

Preuves de protocoles cryptographiques : méthodes symboliques et attaquants puissants

Charlie Jacomme

▶ **To cite this version:**

Charlie Jacomme. Preuves de protocoles cryptographiques : méthodes symboliques et attaquants puissants. Cryptographie et sécurité [cs.CR]. Université Paris-Saclay, 2020. Français. NNT : 2020UPASG005 . tel-02972373

HAL Id: tel-02972373
https://theses.hal.science/tel-02972373

Submitted on 20 Oct 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution| 4.0 International License

université
PARIS-SACLAY

Proofs of Security Protocols

Symbolic Methods and Powerful Attackers

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Unité de recherche : Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV
Référent : ENS Paris-Saclay

Thèse présentée et soutenue à Gif-sur-Yvette le 16 Octobre 2020, par

Charlie JACOMME

Composition du jury:

Sylvie BOLDO Directrice de recherche, Inria Saclay	Présidente
Matteo MAFFEI Professeur, Technische Universität Wien	Rapporteur & Examinateur
Bogdan WARINSCHI Professeur, University of Bristol	Rapporteur & Examinateur
Bruno BLANCHET Directeur de recherche, Inria Paris	Examinateur
Sandrine BLAZY Professeure, Université de Rennes 1	Examinatrice
Hubert COMON Professeur, ENS Paris-Saclay	Directeur
Steve KREMER Directeur de recherche, Inria Nancy	Co-directeur

Thèse de doctorat
NNT2020UPASG005

Abstract

The use of communication protocols has become pervasive at all levels of our society. Yet, their uses come with risks, either about the security of the system or the privacy of the user. To mitigate those risks, we must provide the protocols with strong security guarantees: we need formal, extensive, modular and machine-checked proofs. However, such proofs are very difficult to obtain in practice. In this Thesis, we strive to ease this process in the case of cryptographic protocols and powerful attackers. The four main contributions of this Thesis, all based on symbolic methods, are

1. a methodology for extensive analyses via a case study of multi-factor authentication;
2. composition results to allow modular proofs of complex protocols in the computational model;
3. symbolic methods for deciding basic proof steps in computational proofs, formulated as problems on probabilistic programs;
4. a prototype of a mechanized prover in the Computationally Complete Symbolic Attacker model.

Résumé

L'utilisation des protocoles de communication est omniprésente dans notre société, mais leur utilisation comporte des risques de sécurité ou d'atteinte à la vie privée. Pour réduire ces risques, il faut exiger de solides garanties, i.e. des preuves formelles, approfondies, modulaires et vérifiées par ordinateur. Toutefois, de telles preuves sont très difficiles à obtenir. Nous essayons dans cette thèse de faciliter ce processus dans le cas des protocoles cryptographiques et d'attaquants puissants. Nos contributions principales sont

1. une méthodologie d'analyse approfondies dans le cas de l'authentification multi-facteurs;
2. des résultats de composition permettant des preuves modulaires de protocoles complexes dans le modèle calculatoire;
3. l'automatisation d'étapes élémentaires de preuves calculatoires via des méthodes symboliques appliquées à des programmes probabilistes;
4. un prototype d'assistant de preuve dans le modèle de l'attaquant symbolique calculatoirement complet.

Contents

Abstract i

Résumé iii

Publications xi

1 Introduction 1
- 1.1 Cryptographic Primitives, Security Properties and Protocols . . . 3
- 1.2 Formal Proofs . . . 5
- 1.3 Our Contributions . . . 8

2 Formal Models for Protocols 13
- 2.1 Generic Syntax and Semantics for Protocols . . . 13
  - 2.1.1 Syntax . . . 14
  - 2.1.2 Parameterized Semantics . . . 16
  - 2.1.3 Reachability Properties . . . 19
- 2.2 Symbolic Semantics . . . 20
  - 2.2.1 Interpretation of Terms . . . 20
  - 2.2.2 Attacker Capabilities . . . 21
  - 2.2.3 Symbolic Indistinguishability . . . 21
- 2.3 Computational Semantics . . . 22
  - 2.3.1 Semantics of Terms and Attackers . . . 22
  - 2.3.2 Computational Indistinguishability . . . 23
- 2.4 The BC Logic . . . 26
  - 2.4.1 From Protocols to Terms . . . 26
  - 2.4.2 A Logic over Terms . . . 28

I Extensive 33

3 A Symbolic Model for Multi-Factor Authentication 35
- 3.1 Introduction . . . 35
  - 3.1.1 Our Contributions . . . 36
  - 3.1.2 Related Work . . . 37
- 3.2 Multi-factor Authentication Protocols . . . 37
  - 3.2.1 *Google 2-step* . . . 37
  - 3.2.2 *FIDO's Universal 2nd Factor* - U2F . . . 39
  - 3.2.3 Disabling the Second Factor on Trusted Devices . . . 40
  - 3.2.4 Token Binding . . . 40
- 3.3 Threat Model . . . 41
  - 3.3.1 Malware Based Scenarios . . . 41
  - 3.3.2 Fingerprint Spoofing . . . 43

3.3.3 Human Errors . . . 43
3.3.4 Threat Scenarios Considered . . . 43
3.4 The Formal Model . . . 44
3.4.1 Extension of the Process Calculus with Secret Channels . . . 44
3.4.2 Modelling TLS Communications . . . 46
3.4.3 Modelling Threat Models . . . 47

4 An Extensive Analysis 49
4.1 Introduction . . . 49
4.1.1 Our Contributions . . . 50
4.1.2 Related Work . . . 50
4.2 Analysis and Comparison . . . 51
4.2.1 Properties and Methodology . . . 51
4.2.2 *Google 2-step*: Verification Code and One-Tap . . . 53
4.2.3 Additional Display . . . 54
4.2.4 Conclusion Regarding *Google 2-step* . . . 56
4.2.5 FIDO U2F . . . 56
4.2.6 Token Binding . . . 57
4.2.7 A G2DT$^{\text{DIS}}$ Extension : G2DT$^{\text{EXT}}$ . . . 57
4.2.8 G2DT$^{\text{EXT}}$ Analysis . . . 59
4.3 Validating Attacks in Practice . . . 60
4.3.1 Session Confusion on G2V . . . 60
4.3.2 Session Confusion on G2OT . . . 61
4.3.3 Phishing Attack on *Google 2-step* . . . 62
4.3.4 Action Confusion and Mixing on *Google 2-step* and U2F . . . 64
4.3.5 USB Attack on U2F . . . 65
4.4 Unlinkability . . . 66
4.4.1 On Privacy . . . 66
4.4.2 Formal Analysis . . . 66
4.4.3 Attack against Key Generation . . . 67
4.4.4 U2F with Counters . . . 68
4.4.5 An Attack Based on Global Counters . . . 68
4.4.6 Combining Both Attacks . . . 68
4.4.7 Improvements . . . 69
4.5 *Google 2-step* vs U2F . . . 69
4.5.1 Practical Considerations . . . 69
4.5.2 Final Comparison . . . 70

II Modular 73

5 A Composition Framework in the Computational Model 75
5.1 Introduction . . . 75
5.1.1 Our Contributions . . . 76
5.1.2 Related Work . . . 76
5.2 Protocols and Indistinguishability . . . 78
5.2.1 Stateless Oracle Machines . . . 79
5.3 Simulatability . . . 81
5.3.1 Protocol Simulation . . . 82
5.3.2 Generic Oracles for Tagged Protocols . . . 92

5.4 Main Composition Theorems . . . 94
5.4.1 Composition without State Passing . . . 94
5.4.2 Composition with State Passing . . . 97
5.4.3 Unbounded Replication . . . 100
5.5 Unbounded Sequential Replication . . . 101
5.6 Application to Key Exchanges . . . 102
5.6.1 Our Model of Key Exchange . . . 102
5.6.2 Proofs of Composed Key Exchange Security . . . 103
5.7 Basic Diffie-Hellman Key Exchange . . . 105
5.8 Extension to Key Confirmations . . . 107
5.8.1 Proofs with Key Confirmations . . . 108
5.9 Application to SSH . . . 109
5.9.1 The SSH Protocol . . . 110
5.9.2 Security of SSH . . . 111
5.9.3 SSH with Forwarding Agent . . . 112

6 The Framework in the BC Logic 115
6.1 Introduction . . . 115
6.1.1 Our Contributions . . . 116
6.1.2 Related Work . . . 116
6.2 Oracles in the BC Logic . . . 116
6.2.1 Syntax and Semantics . . . 116
6.2.2 Oracle Soundness . . . 117
6.3 Computational Soundness . . . 120
6.4 Extension to the Model for Unbounded Replication . . . 122

III Automated 127

7 Probabilistic Language and Problems 129
7.1 Introduction . . . 129
7.1.1 Our Contributions . . . 131
7.1.2 Related Work . . . 132
7.2 Probabilistic Programming Language . . . 133
7.2.1 Syntax and Informal Semantics . . . 134
7.2.2 A Core Language . . . 135
7.2.3 Semantics . . . 136
7.3 Decision Problems and Universal Variants . . . 138
7.4 First Results . . . 139
7.4.1 Links between Problems . . . 140
7.4.2 Semantic Characterization of Equivalence . . . 142

8 Complexity and Decidability 145
8.1 Introduction . . . 145
8.1.1 Our Contributions . . . 146
8.1.2 Related Work . . . 148
8.2 Complexity in the Finite Case . . . 149
8.2.1 Conditional Equivalence . . . 149
8.2.2 Independence . . . 151
8.2.3 Majority . . . 152

8.3 The Universal Case . . . 153
8.3.1 General Remarks . . . 153
8.3.2 From Arithmetic Programs without Inputs to LRS . . . 156
8.3.3 Decidability of Universal Equivalence . . . 158
8.4 Program Indistinguishability . . . 161
8.5 Undecidability with Loops . . . 164

9 In Practice 167
9.1 Introduction . . . 167
9.1.1 Our Contributions . . . 168
9.1.2 Related Work . . . 169
9.2 Symbolic Characterization . . . 169
9.2.1 Symbolic Abstraction . . . 170
9.2.2 Symbolic Characterization . . . 172
9.3 Symbolic Methods for Probabilistic Programs . . . 173
9.3.1 Using Deduction to Check Uniformity . . . 173
9.3.2 Deduction Constraints and Unification for Program Equivalence . . . 174
9.3.3 Static Equivalence and Non Equivalence . . . 177
9.4 Extending Symbolic Results . . . 178
9.4.1 Deciding Deducibility for Diffie-Hellman Theories . . . 179
9.4.2 Fields and Commutative Rings . . . 179
9.4.3 From One-Step Deduction Constraints to Originated and Monotone Constraints. . . . 182
9.5 Deriving Heuristics . . . 183
9.5.1 Soundness and Completeness . . . 183
9.5.2 Boolean Algebras: the Linear Case . . . 184
9.5.3 Boolean Algebras: the General Case . . . 184
9.5.4 Extension to More Complex Algebras . . . 185
9.5.5 Interference Witnesses . . . 186
9.5.6 Sampling from Multiple Distributions . . . 186
9.6 Applications . . . 186
9.6.1 Implementation of a Library . . . 186
9.6.2 Integration in MaskVerif . . . 187
9.6.3 Integration in EasyCrypt . . . 188

IV A New Hope 191

10 An Interactive Prover for Indistinguishability Proofs 193
10.1 Introduction . . . 193
10.1.1 Our Contributions . . . 194
10.1.2 Related Work . . . 194
10.2 Overview . . . 196
10.3 A Meta-Logic for Reachability and Equivalence . . . 197
10.3.1 The Meta-Logic . . . 197
10.3.2 Reachability Rules . . . 202
10.3.3 Indistinguishability Rules . . . 204
10.4 Implementation and Case-Studies . . . 205
10.4.1 The Tool . . . 205
10.4.2 Case-Studies . . . 206

Conclusion and Future Work 209

Bibliography 213

A Appendix of Part I 229
A.1 Global Results for MFA 230

B Appendix of Part II 235
B.1 Formal Corollary for Key Exchange 236
B.2 Formal Corollary for Key Confirmations 237
B.3 Proofs of Chapter 5 239
B.3.1 Oracle Simulation 239
B.3.2 Autocomposition Results 241

C Appendix of Part III 257
C.1 Proof of Chapter 7 258
C.2 Proofs of Chapter 8 261
C.3 Proofs of Section 9.4.1 270
C.3.1 Saturation into the Target Group 270
C.3.2 Reduction to Polynomials 270

Publications

Some ideas have appeared previously in the following publications. Remark that our participation w.r.t. [BFG+18] is focused in the Section 4 of this paper.

[BFG+18] Gilles Barthe, Xiong Fan, Joshua Gancher, Benjamin Grégoire, Charlie Jacomme, and Elaine Shi. Symbolic proofs for lattice-based cryptography. In Michael Backes and XiaoFeng Wang, editors, *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS'18)*, pages 538–555, Toronto, Canada. ACM Press, October 2018. URL: `https://dl.acm.org/citation.cfm?doid=3243734.3243825`.

[BGJ+19] Gilles Barthe, Benjamin Grégoire, Charlie Jacomme, Steve Kremer, and Pierre-Yves Strub. Symbolic methods in computational cryptography proofs. In Stéphanie Delaune and Limin Jia, editors, *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'19)*, pages 136–151, Hoboken, NJ, USA. IEEE Computer Society Press, July 2019. DOI: `10.1109/CSF.2019.00017`. URL: `https://hal.inria.fr/hal-02117794`.

[BJK20] Gilles Barthe, Charlie Jacomme, and Steve Kremer. Universal equivalence and majority on probabilistic programs over finite fields. In Naoki Kobayashi, editor, *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'20)*, Saarbrücken. ACM, July 2020. To appear.

[CJS20] Hubert Comon, Charlie Jacomme, and Guillaume Scerri. Oracle simulation: a technique for protocol composition with long term shared secrets. In Jonathan Katz and Giovanni Vigna, editors, *Proceedings of the 27st ACM Conference on Computer and Communications Security (CCS'20)*, Orlando, USA. ACM Press, November 2020. To appear.

[JK18] Charlie Jacomme and Steve Kremer. An extensive formal analysis of multi-factor authentication protocols. In Steve Chong and Stéphanie Delaune, editors, *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 1–15, Oxford, UK. IEEE Computer Society Press, July 2018. DOI: `10.1109/CSF.2018.00008`. URL: `https://ieeexplore.ieee.org/document/8429292/`.

We also rely on the following work under submission:

[BDK+20] David Baelde, Stéphanie Delaune, Adrien Koutsos, Charlie Jacomme, and Moreau Solène. An interactive prover for protocol verification in the computational model, 2020. URL: `https://github.com/squirrel-submission-sp21/squirrel-prover`. Under submission.

1 Introduction

> If you can't give me poetry, can't you give me poetical science?
>
> *(Ada Lovelace)*

Communication technologies have brought a huge shift in our society, and now play a central part in our everyday lives. We now rely on internet and instant communications to manage our bank accounts, our daily communications, our health data, ... Those technologies come with many advantages, but they come at a price. Each technological device that we connect to, if it is ill-designed, may yet become another mean of control, surveillance and discrimination. Major companies have built their empire over their user's data, selling it to the highest bidder and using it to adapt our news feed, to tailor the advertisement we receive and to discriminate over our ability to obtain a job, a loan for a house, a health insurance, or a credit card. Depicted in *Orwell*'s "1984" or *Damasio*'s "Les Furtifs" (among many others), theorized by *Foucault* [Fou75] and *Deleuze* [Del92], an absence of Privacy may lead to catastrophic consequences.

As such, we expect this technology to come with guarantees. Can I have the guarantee that my privacy is not violated by using an application, on my cellphone, e.g., a COVID tracking app? Can I be sure that no one can steal the credentials I use to login on my bank account? Can I trust e-voting systems? Can I hope that the communications with my close ones are secure? Currently, the answer to all those question is *no*. But, all those questions deserve a better answer. And even if it is impossible to provide a perfect solution that comes with strong guarantees, we must strive towards the best possible solution.

Unfortunately, it is difficult to provide guarantees about communication technologies: they are systems that involve communications between multiple entities and rely on many different layers. In such systems, flaws may come from many different places:

- the protocol itself, i.e., the exact ordering of messages and operations followed by the devices to send and receive messages, may be ill-designed;
- the cryptographic primitives, i.e., the mathematical constructs used to encrypt messages, may be too weak;
- the implementation, i.e., the code meant to execute the protocol, could be flawed or contain bugs;
- the operating system, i.e., the environment running the code, might be compromised;
- the hardware, i.e., what is physically running the code; might have back-doors or bugs;
- and finally, the users themselves may simply perform the wrong actions.

Providing guarantees at all those levels is a necessity, but also an overwhelming challenge. In this Thesis, we focus on the protocols and their design. If those are flawed, the rest of the chain is broken.

A major question is:

- *what sort of guarantees can we obtain about protocols?*

We would like to claim that no attacker can ever break the protocol, assuming that the rest of the chain is secure. But what sort of claim is trustworthy? History has shown many times over that designing protocols is a difficult task: many protocols considered or even "proved" secure at one

point have later on been attacked. Thus, it has become widely accepted that for a protocol to be deemed secure, a *formal proof* of its security should be provided. A big question remains:

- *against what kind of attacker do we perform a proof of security?*

If we do not consider attackers with sufficient power, the proof may be useless. If we give too much power to the attacker, it will always be able to break the security of the protocol.

Ideally, we thus want to obtain formal proofs of security for the *strongest possible* attacker. Accordingly, we claim that proofs should be:

- **formal** - we must have a high level of confidence in its correctness;
- **extensive** - the proof should try to consider attackers as strong as possible, looking at all possible threat models and configurations of the protocol.

This kind of proof allows to derive the exact assumptions under which the protocol is secure. It is then possible to formally verify if the implementation and the rest of the chain do satisfy those assumptions.

However, performing such proofs is a difficult challenge that has attracted a lot of attention over the past decades. In this Thesis, we strive to make this process easier. To this end, we focus on the following goals.

- **mechanized** - proofs should be performed in a mechanized prover, thus allowing for a high level of trust in the proof.
- **modular** - proofs of compound protocols (complex protocols involving multiple components) should be obtained from proofs of their components. Otherwise, proving compound protocols can be an overwhelming challenge w.r.t. the size of the proof. Furthermore, a small change in a component of the protocol, which happens often during the standardisation process, would imply to redo the full proof.
- **automated** - proofs should be made easier through automation, removing the most tedious steps for the user. Ideally, proofs in a mechanized prover should follow the high-level intuition.

Outline We strive to ease the process of performing formal and extensive proofs by relying on a mechanized prover that allows to perform formal and modular proofs. Keeping in mind the four paradigms, we

- provide in Chapter 2 the **formal** models we will use to perform security proofs;
- show what an **extensive** and completely automated analysis may look like in Part I;
- develop a composition framework to enable for **modular** proofs in Part II;
- study the **automation** of some basic proof steps in Part III;
- and finally provide a **mechanized prover** in Part IV.

How to read This Thesis can be read at multiple levels, with some content highlighted in dedicated environments:

- each Chapter contains at its beginning a brief *Chapter Summary*, along with dedicated *Related Work*, *Future Work* and *Limitations* when relevant;
- each Section begins with a slightly more detailed *Section Summary*;
- in each Section, the most *Technical details*, not necessary to the general understanding, are isolated;
- finally, proofs that are mostly technical and do not leverage interesting concepts have been pushed in Appendices.

Reading the proofs can always be omitted, and all cross-references can be followed by clicking on them. We tried to write this Thesis in a gender neutral way, notably through the use of the

singular *they*. Sentences such as "The attacker cannot know with which protocol among two they are interacting" will thus appear.

1.1 Cryptographic Primitives, Security Properties and Protocols

Before providing a general state of the art of formal protocol verification, we briefly introduce the central notions of cryptography.

Security properties Different use cases lead to many different expectations in term of security. For illustration purposes, let us consider the case of a user that wishes to consult their bank account online, by connecting from their computer to a website. First, only this specific user should be able to access and operate over the bank account. The server (hosting the website) should then make sure that it is indeed the identified user that is accessing the website. In such cases, we say that a server must *authenticate* the user. Furthermore, we also expect that nobody apart from the user should be able to learn any information about the bank account. In this case, we expect the *confidentiality* of the exchanged messages between the user's computer and the server.

Those two properties, authentication and confidentiality, are typical examples of *reachability properties*. Those properties specify that some bad state should never occur, for instance that the wrong user never accesses the bank account, or that nobody can ever learn the balance of their account.

More complex properties involves the notion of privacy. For instance, we may expect that nobody can identify a user that logged in over the website of a specific bank, and nobody should be able to learn in which bank a given user has an account. To this end, the exchanges between the computer and the server should preserve the *anonymity* of the user. In such a case, we expect that nobody can know if it is the given user or another one that is performing the login. A stronger version of this property expects that it is even impossible to know whether two consecutive logins are from the same user or from two distinct users; this property is called *unlinkability*.

Those properties fall into the class of *indistinguishability* properties. They ask that nobody can distinguish between two given scenarios. Anonymity is generally formalized as: none can distinguish whether a distinguished user, or another user performs the login. Unlinkability is formalized by saying that the scenario in which the same user logs in multiple times is indistinguishable from the scenario where many different users log in.

To provide such security properties, we must have means to hide the content of a message to unauthorized parties, and to certify that a given message was produced by a given party. This is the purpose of the *cryptographic primitives*.

Cryptographic primitives They constitute the building blocks of cryptography. They are functions that allow to hide or authenticate a message (among more complex functionalities). The most classical cryptographic primitives are divided in four categories.

- A *symmetric encryption* is defined by two functions `enc` and `dec`. The encryption should intuitively guarantee that for any message m and bitstring sk, $\texttt{enc}(m, sk)$, called the cyphertext, returns a message that does not leak any information about m, unless one knows sk. Computing $\texttt{dec}(\texttt{enc}(m, sk), sk)$ should return m. sk is called the *secret key*.[1]

[1] This primitive, and the following ones, always come with a key generation algorithm, used to derive fresh

- An *asymmetric encryption* is defined by three functions $\texttt{enc}$, $\texttt{dec}$ and $\texttt{pk}$. It is such that knowing $\texttt{pk}(sk)$ is enough to compute cyphertexts with $\texttt{enc}(m, \texttt{pk}(sk))$. They do not leak any information about m, unless somebody knows sk, in which case they may obtain m by computing $\texttt{dec}(\texttt{enc}(m, \texttt{pk}(sk)), sk)$. $\texttt{pk}(sk)$ is the *public key* associated to sk, which, together, form a *key pair*.
- A *keyed hash function* is defined by a single function h, such that $h(m, sk)$ does not leak any information about m. A hash function is meant to provide a short certificate about a message, and thus $h(m, sk)$ should be concise compared to m. It is thus non invertible, contrary to encryption.
- A *signature* scheme is defined by two functions $\texttt{sign}$ and verify. It is used to authenticate a message. Knowing sk should be the only way to compute $\texttt{sign}(m, sk)$, called a *signature*. Then, anybody knowing the corresponding public key can verify if a given input x is indeed a valid signature, i.e., $\mathsf{verify}(x, m, \texttt{pk}(sk))$ returns true if and only if x is equal to the value returned by $\texttt{sign}(m, sk)$.

Given those building blocks, we must specify how each party is going to use them, and exactly what is the sequence of messages that the parties are going to exchange. This list of instructions is called a *protocol*, and even with secure primitives, protocols can often be insecure.

Protocols They are a sequence of instructions, that specifies at each step exactly which operations a party should perform. We may consider a very simple protocol between two parties, where

1. a party A samples a fresh random value n_r;
2. sends the message $\texttt{enc}(\langle A, n_r\rangle, \texttt{pk}(sk_B))$ to a party B;
3. B decrypts the message using its secret key sk_B.

If B is the only one that knows the secret key sk_B and if the asymmetric encryption behaves as expected, this protocol ensures the secrecy of the random value n_r. However, remark that B has no way of knowing that the message it received does come from A, as anybody could compute a valid cyphertext containing the public identity of A and that B could decrypt. Thus, this protocol does not provide any authentication.

We can modify the protocol as follows:

1. a party A samples a fresh random value n_r;
2. sends the pair of message $\langle \texttt{enc}(n_r, \texttt{pk}(sk_B)), \texttt{sign}(\texttt{enc}(n_r, \texttt{pk}(sk_B)), sk_A\rangle$ to a party B;
3. B receives the pair $\langle x_1, x_2\rangle$, checks that $\mathsf{verify}(x_2, x_1, \texttt{pk}(sk_A))$ returns true, and then decrypts the message x_1 using its secret key sk_B.

If sk_A is only known to A, and sk_B only known to B, this protocol ensures the authentication of A, as B is sure once it verified the signature that the message it received was computed by A. Of course, this holds only if the signature behaves as expected. Remark however that in this protocol A does not have any way to be sure that B did indeed receive its message. Further, B could accept multiple times the same message, while A only sent it once. B would need to store all the randoms n_r he received, in order to avoid this.

We show with this example that to obtain some precise security properties, the design of a protocol based on some cryptographic primitives is a complex operation. This example raises many questions, among them,

- What does it mean for a cryptographic primitive to "behave as expected"? Indeed, how can we specify that a symmetric encryption correctly performs its expected duty.
- How can we be sure that the protocol satisfies some given security property? Even a protocol based on secure cryptographic primitives can be ill-designed, and be insecure. It appears

random secret keys, that may rely on some random seeds. Further, the primitives must be randomized to be able to provide a satisfactory level of security.

difficult to prove that nobody can break the security property, as we cannot know in advance all the capabilities that an attacker may have.
- What does it mean that "nobody" should be able to break the security? If we consider confidentiality, there is always the possibility to simply guess the secret value, and thus break the security of the protocol, even though this is very unlikely to happen. We actually need to consider a restricted class of attackers, with reasonable computational capabilities, but at least as strong as attackers may be in practice.

1.2 Formal Proofs

To answer the previous questions, the notion of provable security was introduced in the 80's, where precise definitions of security and mathematical proofs of security were formalized. Two main paradigms have been considered to show the security of protocols.

- The first one was introduced in the seminal paper of Goldwasser and Micali [GM84]. This is the *computational model*, where a construction is secure if any arbitrary computationally bounded adversary can only break the security with a very small probability. In this model, security proofs often rely on the assumption that some problem is computationally hard.
- The second one is the *symbolic model*. It considers an attacker introduced by Dolev and Yao [DY81], that has full control over the network, has no restriction on the computation power in term of time and memory, but can only perform a fixed set of operations over the intercepted messages. In this model, the cryptographic primitives are often assumed to be perfect.

Computational model Adversaries are any arbitrary probabilistic polynomial time Turing Machines and messages are sequences of bitstrings. In this model, we reason over the probabilities that a given message is sent over the network. Many of the proofs in this model take the form of reductionist arguments in computational complexity, similar to the one used in undecidability or hardness proofs. The security of a construction is always proven secure under a computational assumption, that specifies that some mathematical problem is difficult. This model thus provides strong guarantees, as long as the security of a construction relies on a computational problem widely considered to be difficult, e.g., that nobody can solve for several decades.

In reductionist arguments, both the security goals (e.g., indistinguishability properties) and the computational assumptions are modelled as probabilistic experiments where a challenger interacts with an adversary; such experiments come with a winning condition, which captures the situation where an adversary has broken the security property. In the simple case, only one assumption is involved. The reductionist argument is then given by a method for transforming an adversary $\mathcal{A}$ against the cryptographic construction under consideration into an adversary $\mathcal{B}$ against the computational assumption, and a proof that $p_{\mathcal{B}} \leq f(p_{\mathcal{A}})$, where $p_{\mathcal{B}}$ denotes the probability of $\mathcal{B}$ winning the experiment (against the assumption), $p_{\mathcal{A}}$ denotes the probability of $\mathcal{A}$ winning the experiment (against the construction), and f is a function such that $f(x)$ is "small" whenever x is "small". This rigorous approach is a pillar of modern cryptography, and arguably one of the keys to its success. However, reductionist proofs are becoming increasingly complex, as a consequence of new application scenarios (requiring more complex constructions) and theoretical advances in the field (yielding stronger but more complex constructions). Remark that cryptographic proofs tend to be more complex that classical complexity reductions, due to the complexity of the constructions, the importance of probabilities and the variety of assumptions.

The game-playing technique of Shoup [Sho04] is a popular methodology for proving security of cryptographic constructions. This technique decomposes reductionist arguments into elementary steps that can be justified individually with relative ease. In the simple case above, involving

a single computational assumption, the technique involves defining a sequence of probabilistic experiments (which are called games in this setting), such that the first experiment captures the security of the construction, and the last experiment captures the security assumption. In addition, the technique requires proving for all steps that $p_{\mathcal{A}_{i+1}} \leq f_i(p_{\mathcal{A}_i})$, where $\mathcal{A}_i$ is an adversary and $p_{\mathcal{A}_i}$ is his winning probability in the i-th experiment. One then concludes by applying transitivity of inequality. The game-playing technique is helpful to tame the complexity of reductionist arguments. However, it remains difficult to build and verify game-playing proofs of complex constructions.

The code-based game-playing technique by Bellare and Rogoway [BR06] is a common variant of the game-playing technique where experiments are modelled as probabilistic programs. This approach has been instrumental in the mechanization of reductionist arguments using tools based on program verification [Hal05; Bla06; BGZ09; BGH+11]. These tools have been used for verifying many representative examples of cryptographic constructions. However, they remain difficult to use by cryptographers, because automation is limited and expertise in program verification is required.

Symbolic model Messages are syntactic constructs built over functions symbols, constants and variables, i.e., terms built over a signature. They are equipped with an equational theory, that captures all the possible equality between the terms. Compared to the computational model, $\mathtt{enc}(n_r, \mathtt{pk}(sk_B))$ would not be used to represent a bitstring; it only has a syntactic meaning. However, the equational theory specifies that $\mathtt{dec}(\mathtt{enc}(n_r, \mathtt{pk}(sk_B)), sk_B)$ is equal to n_r. Random samplings are denoted by dedicated constants, e.g., n_r, called *names*. Attackers can perform any operation over the network, but are only given a fixed set of capabilities to manipulate messages, accordingly to the equationnal theory. It implies that, with the previous rule for decryption, the asymmetric encryption is perfect. Notably, if an attacker has $\mathtt{enc}(n_r, \mathtt{pk}(sk_B)), sk_B)$, they may obtain n_r by also having sk_B and computing $\mathtt{enc}(n_r, \mathtt{pk}(sk_B)), sk_B)$. Conversely, this implies that if sk_B is not known to the attacker, n_r is secret. Remark that by adding equations to the theory, the attacker capabilities could be increased.

Proofs in this model are often performed by saturation of the attacker knowledge. As the attacker has a fixed set of capabilities, one can try to explore all its possible sequences of actions. If no attack is found, the protocol is secure. This process can however be very difficult, and even undecidable, notably in the case of protocols with an unbounded number of sessions. In essence, the more complex is the application scenario and the stronger is the attacker, the more difficult is the proof process.

The applied pi-calculus, introduced by Abadi and Fournet [AF01][2], is based on the Dolev-Yao attacker and models protocols using a basic programming language for distributed communicating systems: the applied pi calculus extends the pi-calculus [Mil99] by allowing to exchange terms rather than just names. It is one of the most widely used techniques to reason about protocols in the symbolic model. Another popular representation includes Multi-Set Rewriting rules, where protocols are modelled using atomic reduction rules, used to specify how the protocol may at each step evolve and possibly change the attacker knowledge.

Thanks to this formalization, proofs can be automated by leveraging term rewriting techniques such as unification. Several tools [Bla16; SMC+12; CKR18a] have demonstrated their usefulness and have been used on complex application scenarios.

Comparison A traditional way of comparing the two models is to say that the computational model specifies negatively the adversary while the symbolic model specifies it positively. In the first one, we only say what the attacker cannot do, while in the second one we specify each capability of

[2] See [ABF17] for an up to date presentation of the model.

the attacker. Consequently, if we forget in the symbolic model to give attackers a specific capability that they may have in the real world, we may miss attacks. We thus say that the symbolic model is not *sound* w.r.t. the computational one: a proof may exist in the first one, while an attacks may exist in the second. The symbolic model is however *complete* w.r.t. the computational model, as an attack in the symbolic model is also an attack in the computational model.

However, this distinction between the symbolic and the computational model has been blurred in recent years. Notably, some attackers capabilities have been defined negatively even in the symbolic model, see e.g., [BRS16; JCC+19]. The capabilities that are defined negatively are however based on the logical execution of the protocol, and not on the implementation of the messages.

In this Thesis, we follow the idea that we can split the possible attacks and protocol flaws into two distinct categories.

- We may consider *high-level flaws*: these are logical flaws that exploit ways to combine capabilities provided by the protocol to produce an attack. They often rely on specific ordering of actions, and we can reason over them in an abstract way. The security of a protocol against such flaws may rely on an axiom specifying, for instance in the case of a signature, that whenever a message appears to be an honest signature with a secret key, the message must have been signed previously in the protocol. This axiom can be expressed both in the computational and the symbolic model, and corresponds to a high level capability of the attacker. It is however difficult to discover all possible logical flaws. Indeed, we may need to assume that the attacker can gain control over some part of the equipment of a user, or that an option of the protocol might be enabled and break the security.
- We may consider *low-level flaws*, that exploit flaws in the implementation of some primitives, the actual length of the messages, or some probabilistic arguments. For instance, a protocol might not be secure if the implementation of the pair construct (used to build tuples) can be confused with a name (used to model a single random sampling) (see [Sce15] for a concrete example of this flaw). Or a protocol might not be secure if, depending on some conditional, it encrypts two messages of distinct length. To avoid as much as possible those flaws, it is necessary to only define negatively the capabilities of the attacker, based on realistic assumptions. Only then do we have that all the assumptions about the implementation are explicit, and it may become possible to verify the implementation.

Those two levels of attacks involve distinct reasoning techniques, that may respectively be seen as high-level reasoning, that involves logical reasoning about the protocol, and low-level reasoning, that involves a detailed and sometimes probabilistic reasoning about some specific messages. The symbolic model, with its high-level of automation, is ideal to show that a protocol does not have any high-level flaw. Remark though that studying this for attackers as strong as possible (w.r.t. high-level flaws) is still a challenge, due to the complexity of the scenarios involved. Furthermore, proofs in the symbolic model are limited, as it is impossible to guarantee that all low-level flaws have been taken into account. The computational model allows to find both high and low-level flaws. It is however more difficult to reason in it about the high-level flaws, as the logical reasoning is more cumbersome due to the high level of details. Furthermore, it lacks the automation of the symbolic model, and some low-level proof steps are tedious to perform by hand.

Initiated by Abadi Rogaway [AR00] and pursued by many, (see e.g., [CKW10] for a survey of the field), the *computational soundness* approach tries to derive assumptions under which results obtained in the symbolic models can be transposed to the computational model. It allowed to obtain great insights into both models. However, computational soundness requires both much stronger assumptions on the primitives, as every symbolic proof should yield a computational proof under those assumptions, and a richer symbolic model, which makes the proofs more difficult.

The BC logic A new approach has been introduced in the recent years by Bana and Comon [BC12; BC14a]. It tries to get the best of both worlds, performing proofs of security

in a first-order logic designed so that the proofs are computationally sound. As it allows to work in a first-order logic, we can reason about high-level flaws in a symbolic way, close to the intuition. Furthermore, it allows to abstract away the details that are not required to prove the security, thus simplifying the reasoning. Finally, as it is computationally sound by construction, it ensures that we derive all the assumptions required to prevent any flaw, both high and low level.

This model has been successfully used to perform several case-studies of real life protocols [SR16; CK17; BCE18; Kou19c], and bears great promises. Two Thesis have studied decision procedures for this model, where Scerri [Sce15] studied the case of reachability properties and developed a mechanized prover for this case, and Koutsos [Kou19b] provided a decision procedure for the indistinguishability logic, when restricted to a specific set of axioms. However, due to its relative youth, it suffers the comparison with the symbolic and computational models on several points.

- It does not have any composition result that enables modular proofs, while many composition frameworks exist for the other two models.
- It only allows for proofs of protocols with a bounded number of sessions. This means that we can prove that for any fixed number of sessions, no attacker can break the security. However, if the attacker is first allowed to choose the number of sessions, which we call an unbounded number of sessions, they might be able to break the security. This can be an important point, when real life protocols are used millions of time everyday.
- It does not have any mechanized prover for indistinguishability properties.

Remark that the BC logic is sometimes called the Computationally Complete Symbolic Attacker (CCSA) model in the literature. We denote it by the BC logic, from its author names, both for concision and because in the way we use it, it is neither a complete nor a symbolic attacker, but rather a symbolic proof technique valid against computational attackers.

1.3 Our Contributions

Let us now outline how our contributions fit in this general state of the art[3], by detailing each of the four Parts of our Thesis. In all four Parts, we attempt to make the process of proving security protocols easier by leveraging symbolic methods, to enable formal and mechanized proofs of protocols against attackers as powerful as possible.

Part I - Extensive We consider an application scenario that involves a complex combination of parties and equipment (computer, phone, server, USB token), called Multi-Factor Authentication (MFA). Due to the number of parties, there are many possible threat scenarios and an extensive analysis implies to look at several thousands of threat models. However, this class of protocols does not tend to rely heavily on cryptographic primitives, and most flaws appear to be high-level flaws. MFA is thus a typical application where the symbolic model is perfectly suited to perform a security analysis: it allows to detect most potential flaws and is highly automated.

In this first Part, we provide a symbolic threat model suited for the study of MFA protocols against attackers as powerful as possible, by giving the attacker extensive compromising capabilities. This contribution shows how, even in the symbolic model, we may still consider a wide range of powerful attackers. We use the Proverif [BCA+10] tool, an automated prover in the symbolic model, to study the security of multiple protocols (variants of *Google 2-step* and *FIDO's U2F*) against such attackers.

This extensive analysis has a direct practical interest, as it is a novel study of real life protocols that are widely used. It is however also interesting to note that this study highlights the strengths

[3] In each Chapter we provide a more detailed and dedicated Related Work Section.

and limitations of the symbolic model. Thanks to its automation, we were able to analyse over 6000 distinct scenarios. Such a study is completely out of reach of the computational model or the BC logic. However, the analysis is limited by the precision of the symbolic model. This is why we strive in the remainder of this Thesis to simplify how computational guarantees can be obtained.

Part II - Modular In this part, we provide a composition framework suitable both for the computational and the BC logic. It allows to decompose proofs of protocols into smaller proofs, either for sequential and parallel composition, with or without shared state and long term shared secrets. It notably allows to reduce the security of an unbounded number of sessions to the security of a single one.

While we did not dwell on this point in our state of the art (it is developed in a dedicated related work), this is an interesting contribution for the computational model, as it was still difficult to handle composition for protocols that share long-term secrets.

The main contribution of this part is however the fact that we can use the framework in the BC logic, as it removes two of its main drawbacks:

- the BC logic is now equipped with a composition framework, so that it is easier to use it to study compound protocols;
- the BC logic is not anymore restricted to the study of a bounded number of sessions, but can be used through the composition framework to derive the security of an unbounded number of sessions.

To hint at its applicability, we use our framework to show how one may reduce the security of a signed Diffie-Hellman key exchange (ISO 9798-3 [Iso]) and of the SSH [YL] protocol to smaller single session proofs.

Part III - Automation Automation in the computational model is a challenge. Notably, for protocols that rely heavily on group or bitstring operations, it is necessary to reason about the probability distributions of the messages to prove the security of the protocol. This low-level reasoning can be tedious, and may be necessary in both the computational and the BC logic.

In this part, we study relational properties over probabilistic programs. The properties can be used as low-level proof steps either in the computational model or the BC logic. For several properties, we provide a complexity analysis and/or obtain their decidability. As we show that many of the problems are essentially non tractable, we derive efficient heuristics, based on widely used symbolic methods. Surprisingly, we leverage the completeness of the symbolic methods to prove equality of distributions and thus obtain computational guarantees.

Those heuristics have been implemented as a library that was integrated in two mechanized cryptographic provers in the computational model, EasyCrypt [BGH+11; BDG+13] and MaskVerif [BBD+15], improving their automation. It has not been integrated in the BC logic, but may become of interest if the model is used to study the security of advanced cryptographic primitives or pairing-based protocols.

Part IV - A mechanized prover In this final part, we provide a mechanized prover for the BC logic, called the Squirrel prover. We do not delve too much into the theoretical details, but rather provide a high level overview of the tool. On top of the BC logic, we build a meta-logic that allows to reason on multiple execution traces of the protocol in an abstract way. The tool allows to reason symbolically on the protocols, with proofs close to the intuition, and abstract away the most cumbersome details of the low-level implementation, while providing clear assumptions about

it. A strength of the tool is that it supports both reachability and indistinguishability properties, and reachability properties can be used to simplify indistinguishability proofs.

We use the Squirrel prover to verify the security of protocols involving multiple cryptographic primitives (encryption, hash, signature, xor, Diffie-Hellman exponentiation), and this for multiple security properties (unlinkability, anonymity, strong secrecy). We also use Squirrel to perform the proofs obtained by applying our composition framework to the SSH protocol in Part II.

In a nutshell In Part I, we provide a practical study, pushing the symbolic model to its limits by considering attackers as powerful as possible. We then focus in the the rest of the Thesis on simplifying the process of deriving computational guarantees about a protocol.

In this respect, our main contribution is the development in Part II of a composition framework simplifying proofs in the computational and the BC logic, notably allowing for security proofs of an unbounded number of sessions in the BC logic. We build on this contribution in Part IV, by providing a mechanized prover for the BC logic that allows to obtain for the first time a mechanized proof for an unbounded number of sessions of a protocol through the BC logic.

Our final contribution, in Part III, is a mostly theoretical study of the automation of low-level proof steps, that could be leveraged in game based or BC proofs. We hint at its usability in the computational model through an implementation integrated in the tools EasyCrypt and MaskVerif, but further work is required to use it meaningfully in the context of the tool of Part IV.

External Repositories

- The models and scripts used to generate systematically all threat models for the MFA case-study can be found at [Mfa].
- The Solveq library, performing some Gröbner basis computations and verifying probabilistic properties, can be found at [Seq].
- The Squirrel Prover can be found at [Squ], with the sources and the case-studies.

Dependencies and concepts Not all Chapters heavily depend on one another, and they sometimes use very distinct ideas and concepts. The most important dependencies for each Chapter are given bellow, so that readers may peruse this work more easily.

- Part I only relies on the symbolic model introduced in Chapter 2. It is a formal analysis of real life multi-factor authentication protocols, and contains a very practical part.
  - Chapter 3 introduces Multi-Factor Authentication protocols, and defines a modular threat model in the symbolic model of Chapter 2.
  - Chapter 4 performs an extensive, completely automated case study by leveraging the modular threat model of Chapter 3.
- Part II involves an understanding of the computational model and its proof techniques.
  - Chapter 5 builds on the computational model introduced in Chapter 2. It depends on probabilistic arguments, and reasons about indistinguishability through cryptographic reductions.
  - Chapter 6 is in the framework of Chapter 5 cast into the BC logic presented in Chapter 2.
- Part III studies probabilistic programs, independently from the protocol models. It relies heavily on probability reasoning and algebraic tools.
  - Chapter 7 introduces the setting of probabilistic programs and the multiple problems that we consider.

  - Chapter 8 studies the complexity and decidability of the problems introduced in Chapter 7. It involves complexity classes from the counting hierarchy, and leverages mathematical background about the Local Zeta function and Linear Recurrence Sequences.
  - Chapter 9 tries to derive efficient heuristics for the problems of Chapter 7. It relies heavily on some classical symbolic proof techniques, and leverages Gröbner Basis and the notion of primal algebra.
- Part IV gives an overview of a novel mechanized prover that relies on the BC logic of Chapter 2. For concision, it does not delve into the theory of the tool, but tries to carry the main intuition. It lightly depends on Chapter 6 to perform some case-study in a modular way.

Independent Results Going away from the global picture, this Thesis contains some results that may be of interest outside its context and sometimes outside of the context of cryptography, with:

- a comparison between *Google 2-step* and U2F as multi-factor authentication protocols (Section 4.5);
- an extension of the BC logic to attackers with access to oracles, allowing for simpler expressions of some axioms (Section 6.2);
- the decidability of universal equivalence of probabilistic programs over finite fields (Section 8.3);
- the decidability of deducibility over field, rings and Diffie-Hellman exponentiation (Section 9.4.1);
- a technique to decide if a function is a bijection trough deducibility (Section 9.3).

2 Formal Models for Protocols

> Reife des Mannes: das heißt den Ernst wiedergefunden haben, den man als Kind hatte, beim Spiel
>
> *(Friedrich Nietzsche - Zitate)*

Multiple models with distinct syntax and meanings have been used to express protocols and security properties. In this Thesis, we work in

- the symbolic model, more precisely the applied pi-calculus, based on the Dolev-Yao attacker;
- the computational model, that is game based and consider an arbitrary Probabilistic Polynomial Time attacker;
- the BC logic, based on labelled transition systems and a first-order logic.

For the sake of coherence, we choose to present here a unified syntax to express protocols and messages, and an abstract execution model parameterized by the definition of an attacker and by the message interpretation. Instantiating those parameters yields either symbolic or computational semantics. The BC logic is then introduced, showing how it allows to derive proofs of security valid in the computational semantics, while working with a first-order logic.

We remark that in multiple aspects, our syntax and semantics diverge from the classical ones of the computational and symbolic models. Some of the differences are due to the unified point of view, and others are design choices required by later results.

Chapter Summary

We provide a syntax for protocols, along with an execution model parameterized by an attacker definition. It is instantiated with different attackers, providing a single syntax, and both symbolic and computational semantics. We finally present the BC logic, which is sound with respect to our computational semantics.

2.1 Generic Syntax and Semantics for Protocols

To provide unified semantics between the multiple models, we draw the syntax and semantics of terms from the BC logic.[1]. We use symbols from an alphabet of *names*, to represent the random samplings. The same symbol used twice represents the same (shared) randomness. Those names can be seen as pointers to a specific randomness, where all the randomness has been sampled upfront at the beginning of the protocol. This idea stems from the BC logic [BC14a], from which we re-use exactly the same term semantics.

[1] This is also required to enable composition with long term shared secrets, where we must be able to specify precisely the shared randomness between protocols

TERMS

t	$::=$	n	name
	$\mid$	$n_{\vec{i}}$	indexed name
	$\mid$	x	variable
	$\mid$	$f(t_1, \ldots, t_n)$	function of arity n

Figure 2.1: Terms

2.1.1 Syntax

Figure 2.1 presents the syntax of terms, used to model messages. Messages can be obtained by applying functions to randomly sampled values or variables (constants can be modelled with function symbols of arity 0).

Random samplings are modelled by names, and variables model possible inputs of the protocol. We denote by $\mathcal{N}$ the set of names, $\mathcal{X}$ the set of variables and Σ the set of function symbols. $\mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$ then denotes the set of terms.

Technical Details

We allow for a single kind of sampling, but some protocols may require to sample booleans, finite field elements, bitstrings, ... While it is possible to type names and function symbols, this would increase the complexity of the presentation. However, we do not lose generality, as even with untyped terms, explicit constructors can be used to model types.

A key addition to the BC logic is that some names can be indexed by sequences of integers or index variables in a set I. This is necessary so that we may later on consider the replication of protocols. When a replicated protocol depends on a name n_i for some variable i, the first copy (session) of the protocol uses n_1, the second n_2, ... Intuitively, names without index models randomness shared by all sessions of the protocol. Variables are used to model the attacker inputs, and function symbols allows to model the cryptographic computations. Names in $\mathcal{N}$ are only names with index, or indexed with only integers and no index variable. They represent the names that can be interpreted, while the other names must have their index variables bounds before they can be given an interpretation.

Technical Details

Formally, each symbol name in $\mathcal{N}$ come with an arity, for any name of arity k, $n_{\vec{i}}$ for a sequence $\vec{i}$ of l integers and $l - k$ variables is a name of arity $l - k$ in $\mathcal{N}$. In particular, $n_{i_1,\ldots,i_k}$ with $i_1, \ldots, i_k \in \mathbb{N}^k$ is a name of arity 0 in $\mathcal{N}$. Only names with arity 0 can be interpreted as bitstrings. When the arity is greater than one, the names depends on some variables that needs to be instantiated before one can provide its interpretation.

Elementary protocols The syntax for elementary protocols, which models a thread running on a computer, is depicted in Figure 2.2. For communications, channels are taken out of a set of constants C that are known to the attacker. $\mathbf{in}(c, x)$ denotes an input, binding the variable x to the received value, and $\mathbf{out}(c, t)$ denotes the output of the term t over the channel c. As channels are constants known by the attacker, all communications are public. An extension with secret channels

Elementary Protocols

$$
\begin{array}{rcll}
P_{el} & ::= & \mathbf{in}(c, x).P_{el} & \text{input} \\
& | & \mathbf{out}(c, t).P_{el} & \text{output} \\
& | & \mathbf{let}\ x = t\ \mathbf{in}\ P_{el} & \text{variable binding} \\
& | & \mathbf{if}\ t_1 = t_2\ \mathbf{then}\ P_{el}\ \mathbf{else}\ P_{el} & \text{conditionals} \\
& | & \mathbf{0} & \\
& | & \bot &
\end{array}
$$

Figure 2.2: Elementary Protocol

Protocols

$$
\begin{array}{rcll}
P & ::= & P_{el} & \\
& | & P_{el}; P & \text{sequential composition} \\
& | & P \| P' & \text{parallel composition} \\
& | & \|^{i \leq N} P & \text{parallel replication} \\
& | & \|^{i} P & \text{unbounded replication} \\
& | & P_{el}^{\star_{i \leq N}} & \text{sequential replication} \\
& | & P_{el}^{\star_{i}} & \text{unbounded sequential replication}
\end{array}
$$

Figure 2.3: Protocol Algebra

is made in Section 3.4 for the symbolic case. The **let** construct allows for variable bindings in a process, and **if then else** enables conditional branchings. **0** is a successfully terminated thread and $\bot$ is an aborted thread.

Example 2.1. We use the function symbol `enc` to model a probabilistic symmetric encryption. Consider the elementary protocol

$$P := \mathbf{in}(c, x).\mathbf{out}(c, \mathtt{enc}(x, r, sk)).\mathbf{in}(c, y).\mathbf{out}(c, \mathtt{enc}(y, r', sk)).\mathbf{0}$$

This elementary protocol just encrypts twice an input with a secret key sk.

Protocols The Protocol Calculus is presented in Figure 2.3. $P; Q$ models sequential composition, and $P \| Q$ the parallel one. In a sequential composition, $\mathbf{0}; P$ reduces to P, while $\bot; P$ reduces to $\bot$. In most cases, to increase readability we will omit $\mathbf{0}$. Looking ahead, we make the distinction between elementary protocols and protocols so that our algebra can be seen as a composition algebra, geared toward the design of a composition framework in the computational model. Remark that this distinction is not made in the classical applied pi-calculus, and that the composition operator is not usually provided.

Example 2.2. We use `dec` to denote the decryption function associated to `enc`, and let $Q := \mathbf{in}(c, x).\mathbf{out}(c, \mathtt{dec}(x, sk))$. Q provides a one time decryption oracle to the attacker. $P; Q$ models the protocol where P must be executed before Q, and $P \| Q$ the protocol where they can be executed in any order. In both cases, Q could be used to decrypt one of the messages encrypted by P. However, in $Q; P$, Q must be executed before P and cannot be used to decrypt a message produced by P.

If a protocol P depends on some name indexed by i, given an explicit integer k, $P\{i \mapsto k\}$ denotes P in which all names n_i are replaced by n_k. Then, $\|^{i \leq N} P$ with an explicit integer N corresponds to N parallel copies of P, with the index i instantiated with $1, \ldots, N$. $\|^{i}$ denotes a replication

where the attacker can choose the number of copies, i.e instantiating N himself. $\star^i$ and $\star^{i\leq N}$ are similar, but of sequential replication. $\mathcal{N}(P)$ is further split into the local names $\mathcal{N}_l(P)$, the set of names indexed by variables, and the global names $\mathcal{N}_g(P)$, the names without index, which are shared between all copies of the protocol. In a protocol, all names appearing with index variables must be such that the index variable is under the scope of a binder ($\|^i$, $\|^{i\leq N}$, $\star^i$, $\star^{i\leq N}$).

We allow terms in a protocol to depend on some free variables and, in this case, we denote by $P(x_1, \ldots, x_n)$ a protocol that depends on the free variables $x_1, \ldots, x_n$. $P(t_1, \ldots, t_n)$ denotes the protocol obtained when instantiating each x_i by the term t_i. We denote by $\mathcal{C}(P)$ the set of channels appearing in a protocol, and $\mathcal{N}(P)$ the set of (indexed) names.

Technical Details

Compared to the classical applied pi-calculus, we do not have a **new** construct used to bind fresh names. In our case, names are explicit pointers to a value, which may be shared between protocols: if P and Q are two protocols using the same name n, in $P\|Q$, both protocols will use the same name n. This behaviour is distinct from the behaviour of (**new** $n.P$)$\|$(**new** $n.Q$), where P and Q both use a distinct name. We are able to replace the binding of fresh names through the use of indexed names, which allows to replicate protocols an unbounded number of time. This modelling of names is similar to the BC logic. It is a key point that allows us to cast our composition framework in the BC logic.

Notice also that sequential composition can only contain elementary protocols on the left side. Allowing protocols of the form $((P_1\|P_2); Q)$ would model a behaviour similar to the phases of ProVerif, that are difficult to handle in a composition framework.

Example 2.3. Given a randomized encryption function `enc`, we let $P(x_1, x_2)$ be the protocol $\mathbf{in}(c, x).\mathbf{out}(c, \mathtt{enc}(x, x_1, x_2))$. Given names sk, r representing respectively a secret key and a random seed, $E := \|^i P(r_i, sk)$ is then the protocol providing an encryption oracle for the key sk.

An encryption oracle for five distinct secret keys is expressed with $\|^i\|^{j\leq 5} P(r_{j,i}, sk_j)$.

2.1.2 Parameterized Semantics

Section Summary

The semantics are close to one of the classical semantics of the applied pi-calculus, but parameterized by:

- a domain D for messages interpretation, equipped with an equality $=_D$;
- an interpretation of terms $[\![t]\!]^\sigma : \mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N}) \mapsto D$, with $\sigma : \mathcal{X} \mapsto D$, where $\mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$ is the set of terms and σ is a substitution containing the binding of variables of a protocol;
- an attacker $\mathcal{A}$, from $D^* \mapsto (C \times D) \cup (I \times \mathbb{N})$, where given a sequence of messages in D^* (D^* is the Kleene star, denoting all sequences of elements in the given set), the attacker must either provide an input to the protocol on a channel in C with a message in D, or choose the number of replications of an index in I. No assumptions are made about the attacker, except that it receives and provides values of the correct type. It is instantiated later on by attackers with limited computational power.

Let D be a domain for message interpretation (which must be equipped with an equality relation $=_D$). A (global) state of a protocol consists in a *frame* φ, which is a sequence of messages in D modelling the current attacker knowledge, and a finite multiset of pairs (P, σ), where P is a protocol and σ is a local binding of variables.

Intuitively, the frame contains the sequence of messages output by the protocol. Each of the components of the multi-set is the current state of a running thread. We write such global states $\varphi, (P_1, \sigma_1) \| \cdots \| (P_n, \sigma_n)$, where we assume commutativity and associativity of the operator $\|$.

The transition relation between global states is parameterized by an interpretation of terms and an attacker. For any $t \in \mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$, we denote by $[\![t]\!]^\sigma \in D$ an interpretation of t, if all the variables in t are bound by σ. We denote by $\mathcal{A}$ a function $D^* \mapsto (C \times D) \cup (I \times \mathbb{N})$, modelling an attacker. The attacker chooses which of the threads is going to move and computes, given φ, the input to that thread. These inputs are specified either as a channel along with an input message $(C \times D)$, or with an index and an integer $(I \times \mathbb{N})$ when he must choose the number of replications of a protocol. Given a value in $C \times D$ or $I \times \mathbb{N}$, we denote by π_1 and π_2 the first and second projection.

We give the rules describing the Structural Operational Semantics of the elementary protocols in Figure 2.4. The semantics of elementary protocols assumes that, after an attacker input, the protocol progresses as far as possible until it terminates or has to wait for another input. Formally, we define a relation $\rightarrow$ that does not depend on the attacker. Out adds to the current frame (which intuitively models the attacker knowledge) the interpretation of the output made by the protocol, given the current assignment of variables. Let stores in the local assignment of the variables the interpretation of the new binding. If and Else reduces the protocol according to the intuitive execution of a conditional branching.

The first four rules of Figure 2.4 defines a reduction relation $\rightarrow$, that trivially terminates and has a normal form. We write $\xrightarrow{!}$ for the reduction of a global state to its normal form w.r.t. $\rightarrow$. It corresponds to reducing as much as possible the global state without any action of the attacker. We can then define the transition relation $\underset{\mathcal{A}}{\longrightarrow}$ between configurations, which depends on the attacker $\mathcal{A}$. We will write $\xrightarrow[\mathcal{A}]{*}$ for its reflexive transitive closure. $\underset{\mathcal{A}}{\longrightarrow}$ can reduce protocols if $\xrightarrow{!}$ can, accordingly to Red. To perform an input with In, the attacker must produce a value (c, m) in $C \times D$, such that c corresponds to the channel of some enabled input. m is then bound to the given variable.

Figure 2.5 presents the rules corresponding to the sequential composition, whose composition semantics are straightforward: in $P; Q$, P has to be executed first. Seq models the fact that P can be executed in $P; Q$, and SeqFail and SeqSucc capture the fact that Q is executed only if P succeeded and reduced to $\mathbf{0}$. In case of success, Q inherits the bindings of P. For unbounded sequential replication, SeqRep simply unfold the given number of copies, and the attacker can choose with SeqStar the number of times the protocol will be replicated by providing the index in I to be instantiated with the given integer. Here, we add to the frame some constant **cst**, denoting any fixed constant of D known to the attacker, so that the attacker knows that this action was performed.

Finally, Figure 2.6 presents the rule for parallel composition. ParNull and ParFail allow to remove terminated processes from the configuration. ParC pushes a parallel process in the configuration. ParRep and ParStar behave similarly to the operators for the sequential replication.

Technical Details

The semantics can be non deterministic because of parallel composition, for instance if two parallel processes expect an input on the same channel. Symbolic semantics will be non deterministic, but to provide computational semantics, we will have to consider a restricted class of processes for which the semantics are deterministic (Definition 2.8).

Elementary protocols

$$\textsc{Out}\ \frac{}{\varphi, (\mathbf{out}(c,s).P, \sigma) \ \to\ \varphi \uplus \{[\![s]\!]^\sigma\}, (P, \sigma)}$$

$$\textsc{Let}\ \frac{}{\varphi, (\mathbf{let}\ x = t\ \mathbf{in}\ P, \sigma) \ \to\ \varphi, (P, \sigma \uplus \{x \mapsto [\![t]\!]^\sigma\})}$$

$$\textsc{If}\ \frac{}{\varphi, (\mathbf{if}\ s = t\ \mathbf{then}\ P\ \mathbf{else}\ Q, \sigma) \ \to\ \varphi, (P, \sigma)}\ \text{if}\ [\![s]\!]^\sigma =_D [\![t]\!]^\sigma$$

$$\textsc{Else}\ \frac{}{\varphi, (\mathbf{if}\ s = t\ \mathbf{then}\ P\ \mathbf{else}\ Q, \sigma) \ \to\ \varphi, (Q, \sigma)}\ \text{if}\ [\![s]\!]^\sigma \neq_D [\![t]\!]^\sigma$$

$$\textsc{In}\ \frac{\varphi, (P, \sigma \uplus \{x \mapsto \pi_2(\mathcal{A}(\varphi))\}) \xrightarrow{!} \varphi', (P', \sigma')}{\varphi, (\ \mathbf{in}\ (c,x).P, \sigma) \underset{\mathcal{A}}{\longrightarrow} \varphi', (P', \sigma')}\ \text{if}\ \pi_1(\mathcal{A}(\varphi)) = c \qquad \textsc{Red}\ \frac{\varphi, (P, \sigma) \xrightarrow{!} \varphi', (P', \sigma')}{\varphi, (P, \sigma) \underset{\mathcal{A}}{\longrightarrow} \varphi', (P', \sigma')}$$

Figure 2.4: Operational Semantics of Elementary Protocols

Sequential Composition

$$\textsc{Seq}\ \frac{\varphi, (P, \sigma) \ \to\ \varphi', (P', \sigma')}{\varphi, (P; Q, \sigma) \ \to\ \varphi', (P'; Q, \sigma')} \qquad \textsc{SeqSucc}\ \frac{}{\varphi, (\mathbf{0}; Q, \sigma) \ \to\ \varphi, (Q, \sigma)}$$

$$\textsc{SeqFail}\ \frac{}{\varphi, (\bot; Q, \sigma) \ \to\ \varphi, (\bot, \sigma)} \qquad \textsc{SeqRep}\ \frac{}{\varphi, (P^{\star i \leq N}, \sigma) \ \to\ \varphi, (P\{i \mapsto 1\}; \ldots; P\{i \mapsto N\}, \sigma)}$$

$$\textsc{SeqStar}\ \frac{}{\varphi, (P^{\star i}, \sigma) \underset{\mathcal{A}}{\longrightarrow} \varphi \uplus \mathbf{cst}, (P^{\star i \leq \pi_2(\mathcal{A}(\varphi))}, \sigma)}\ \text{if}\ \pi_1(\mathcal{A}(\varphi)) = i$$

Figure 2.5: Operational Semantics of Sequential Composition

Parallel Composition

$$\textsc{ParNull}\ \frac{}{\varphi, (\mathbf{0}, \sigma) \| E \ \to\ \varphi, E} \qquad \textsc{ParFail}\ \frac{}{\varphi, (\bot, \sigma) \| E \ \to\ \varphi, E}$$

$$\textsc{ParC}\ \frac{\varphi, (P, \sigma) \ \to\ \varphi', E'}{\varphi, (P, \sigma) \| E \ \to\ \varphi', E' \| E} \qquad \textsc{Par}\ \frac{}{\varphi, (P \| Q, \sigma) \ \underset{\mathcal{A}}{\to}\ \varphi, (P, \sigma) \| (Q, \sigma)}$$

$$\textsc{ParRep}\ \frac{}{\varphi, (\|^{i \le N} P, \sigma) \ \to\ \varphi, (P\{i \mapsto 1\} \| \cdots \| P\{i \mapsto N\}, \sigma)}$$

$$\textsc{ParStar}\ \frac{}{\varphi, (\|^{i} P, \sigma) \ \underset{\mathcal{A}}{\to}\ \varphi \uplus \mathbf{cst}, (\|^{i \le \pi_2(\mathcal{A}(\varphi))} P, \sigma)}\ \text{if}\ \pi_1(\mathcal{A}(\varphi)) = i$$

Figure 2.6: Operational Semantics of Parallel Composition

2.1.3 Reachability Properties

Reachability properties characterize properties that are true on every possible executions of the protocol. For instance, we can specify that some value will always be secret, or that some event is always preceded by another one. To formalize the notion of events, we extend the syntax with the construction **event** $e(t_1, \ldots, t_k)$, where $t_1, \ldots, t_k$ is a sequence of terms and e a fresh symbol, with the associated reduction rule $\varphi, (\textbf{event}\ e(t_1, \ldots, t_k).P, \sigma) \ \to\ \varphi, (P, \sigma)$.

Definition 2.1. Given a reduction sequence $A_1 \underset{\mathcal{A}}{\to} \ldots \underset{\mathcal{A}}{\to} A_l$, with $s_1, \ldots, s_k \in \mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$ we say that the event $e(s_1, \ldots, s_k)$ occurs at position i with substitution σ' if $A_i = \varphi, (\textbf{event}\ e(t_1, \ldots, t_k).P, \sigma) \| E$ such that for all $1 \le j \le k$, $[\![t_j]\!]^{\sigma} = [\![s_j]\!]^{\sigma'}$. Given a protocol P, we say that $e(s_1, \ldots, s_k)$ is unreachable if for any attacker $\mathcal{A}$ and all reductions $\emptyset, (P, \emptyset) \overset{*}{\underset{\mathcal{A}}{\to}} \varphi, (P', \sigma)$, $e(s_1, \ldots, s_k)$ does not occur at any position.

We will later on be interested in verifying *authentication properties*. We model them, following [Bla09], as correspondence properties of the form

$$e_1(t_1, \ldots, t_n) \Longrightarrow e_2(u_1, \ldots, u_l)$$

Such a property holds, if in each execution, every occurrence of an instance of $e_1(t_1, \ldots, t_n)$ is preceded by the corresponding instance of $e_2(u_1, \ldots, u_m)$. This property typically represents authentication between two entities, where a server must only accept a connection if a user initiated it.

We denote by $\mathsf{dom}(\sigma)$ the domain of a substitution σ, and say that two substitutions σ, σ' are compatible if for all $x \in \mathsf{dom}(\sigma) \cap \mathsf{dom}(\sigma)$, $x\sigma = x\sigma'$.

Definition 2.2. The property $e_1(t_1, \ldots, t_n) \Longrightarrow e_2(u_1, \ldots, u_m)$ is verified by a protocol P if for all symbolic attacker $\mathcal{A}$ and all reductions $\emptyset, (P, \emptyset) \underset{\mathcal{A}}{\to} A_1 \underset{\mathcal{A}}{\to} \ldots \underset{\mathcal{A}}{\to} A_k$, if $e_1(t_1, \ldots, t_n)$ occurs at position i with substitution σ, then there exists j such that $e_2(u_1, \ldots, u_m)$ occurs at position j with substitution σ', where σ, σ' are compatible.

Technical Details

Contrary to the intuition, the previous definition does not explicitly require j to be smaller than i. Indeed, it is actually implied by the definition: if there are only reductions where e_2 occurs after e_1, as we quantify over all reductions, we can consider the prefix of the reduction that ends with e_1, which does not satisfy the property.

We may actually expect a stronger property, where the event can be matched with a single other one. For such properties, we use *injective* correspondence properties

$$e_1(t_1, \ldots, t_n) \Longrightarrow_{inj} e_2(u_1, \ldots, u_m)$$

that require that each occurrence of e_1 is matched by a *different* preceding occurrence of e_2.

Definition 2.3. The property $e_1(t_1, \ldots, t_n) \Longrightarrow_{inj} e_2(u_1, \ldots, u_m)$ is verified by a protocol P if for all symbolic attacker $\mathcal{A}$ and all reductions $\emptyset, (P, \emptyset) \underset{\mathcal{A}}{\longrightarrow} A_1 \underset{\mathcal{A}}{\longrightarrow} \ldots \underset{\mathcal{A}}{\longrightarrow} A_k$, there exists an injective function f over the integers, such that if $e_1(t_1, \ldots, t_n)$ occurs at position i with substitution σ, $e_2(u_1, \ldots, u_m)$ occurs at position $f(i)$ with substitution σ', where σ, σ' are compatible.

2.2 Symbolic Semantics

Section Summary

In the symbolic model, terms are simply interpreted as terms, with equality modulo an equational theory modeling properties of cryptographic constructions. The attacker is any function producing ground terms through function applications over its knowledge.

2.2.1 Interpretation of Terms

In the Dolev-Yao model, the cryptographic primitives are assumed to be perfect. Terms are interpreted modulo an equational theory modelling the primitives.

An equational theory E is a set of equations $u = v$ where $u, v \in \mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$. The equivalence relation $=_E$ is defined by the equalities of E closed by reflexivity, transitivity, substitutions of variables by terms and application of function symbols, i.e the smallest equivalence relation such that

- $u\phi =_E v\phi$ for any $u, v \in \mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$ and substitution ϕ;
- $u_1 = v_1, \ldots, u_k = v_k \Rightarrow f(u_1, \ldots, u_k) = f(v_1, \ldots, v_k)$ for any $f \in \Sigma$ of arity n.

Example 2.4. With $x, y, z \in \mathcal{X}$, we define the equational theory E that associates to the function symbols $\mathtt{enc}, \mathtt{dec}$ the equation $\mathtt{dec}(\mathtt{enc}(x, y, z), z) = x$. Then, for any ground term (without variables) t, random r and secret key sk, we have $\mathtt{dec}(\mathtt{enc}(t, r, sk), sk) =_E x$

Given E, we instantiate the previous parameters such that D is $\mathcal{T}(\Sigma, \mathcal{N})$ (the set of ground terms) equipped with the equality relation $=_E$, and $[\![t]\!]^\sigma = t\sigma$. Compared to ProVerif, we do not define a notion of constructors and destructors symbols, that allow to model function symbols whose reduction may fail on some inputs.

2.2.2 Attacker Capabilities

The attacker will only be able to produce terms according to the equational theory and its knowledge. To this end, we introduce the classical definition of *deducibility*.

Definition 2.4. Let E be an equationnal theory and $t_1, \ldots, t_k, s \in \mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$. We say that s is deducible modulo E from $t_1, \ldots, t_k$, denoted $t_1, \ldots, t_k \vdash_E s$ if and only if:

$$\exists R \in \mathcal{T}(\Sigma, (x_1, \ldots, x_k), \emptyset).\ R\sigma =_E s$$

where $x_1, \ldots x_n$ are variables disjoint from $\mathcal{X}$ and $\sigma = \{x_1 \mapsto t_1, \ldots, x_k \mapsto t_k\}$.

Intuitively, the term R models the computation of the adversary, and the variables x_i are used as *handles* to refer to the corresponding terms t_i.

Example 2.5. With the equational theory E of Example 2.4, we have that $sk, \mathtt{enc}(m, r, sk) \vdash_E m$ thanks to the term $R(x_1, x_2) := \mathtt{dec}(x_2, x_1)$. However, we have $\mathtt{enc}(m, r, sk) \nvdash_E m$

Notice that when performing a deduction, the attacker is not allowed to use the names modelling the secret random samplings of the protocol. We assume that Σ contains an infinite set of constants, to allow the attacker access to an unbounded number of values. When clear from the context, we omit E.

Given an equational theory, a symbolic attacker $\mathcal{A}$ is then a function $D^* \mapsto (C \times D) \cup (I \times \mathbb{N})$, which given a list of ground terms $t_1, \ldots, t_k$ either gives back an index and and integer (for replication) or a term s along with a channel, such that $t_1, \ldots, t_k \vdash_E s$.

2.2.3 Symbolic Indistinguishability

Indistinguishability captures the fact that no attacker can decide with which protocol among two they interact. We first define *static equivalence*, which specifies when two sequences of terms cannot be distinguished by an attacker.

Definition 2.5. Two sequences $t_1^1, \ldots, t_k^1$, and $t_1^2, \ldots, t_k^2$ of terms in $\mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$ are statically equivalent in E, written $t_1^1, \ldots, t_k^1 \sim_E t_1^2, \ldots, t_k^2$ iff

$$\forall u_1, u_2 \in \mathcal{T}(\Sigma, (x_1, \ldots, x_k)).\quad \begin{array}{c} u_1\sigma^1 =_E u_2\sigma^1 \\ \Leftrightarrow \\ u_1\sigma^2 =_E u_2\sigma^2 \end{array}$$

where $x_1, \ldots x_n$ are variables disjoint from V and $\sigma^i = \{x_1 \mapsto t_1^i, \ldots, x_k \mapsto t_k^i\}$.

Intuitively, two sequences of terms are statically equivalent if the set of equations between terms are the same on both sequences.

Example 2.6. With the equational theory E of Example 2.4, we have that encryption is statically equivalent to a name when the attacker does not have the secret key, i.e., $\mathtt{enc}(m, r, sk), m \sim_E n, m$. However, $sk, \mathtt{enc}(m, r, sk), m \not\sim_E sk, n, m$, as the relation $\mathtt{dec}(x_2, x_1) = x_3$ is true on the left side but not on the right side.

We then define *trace equivalence* to hold for two protocols if they have for any attacker the same set of possible reductions, and the produced frames are always statically equivalent.

Definition 2.6. Two protocols P, Q are trace equivalent if, for any symbolic attacker $\mathcal{A}$, the reductions $\emptyset, (P, \emptyset) \xrightarrow[\mathcal{A}]{*} \varphi_P, E_P$ and $\emptyset, (Q, \emptyset) \xrightarrow[\mathcal{A}]{*} \varphi_Q, E_Q$ have the same length and $\varphi_P \sim_E \varphi_Q$.

Technical Details

We use in Chapter 4 ProVerif to prove results about symbolic indistinguishability. Remark that ProVerif actually proves a stronger relation which implies in particular trace equivalence.

2.3 Computational Semantics

Section Summary

In the computational model, terms are interpreted as bitstrings. Function symbols and names are interpreted through PTTMs taking as input an infinite random tape and a security parameter. The attacker is any PTTM with an infinite random tape.

2.3.1 Semantics of Terms and Attackers

We interpret terms as elements of a set of bitstrings. In the computational model, security is parameterized by the length of the randomly sampled values, called the security parameter. Thus, the interpretation of terms must depend on some security parameter. We must moreover provide a consistent way to interpret names, such that given a security parameter, all names correspond to random sampling of the correct length, and the same name returns the same value. To provide such an interpretation, and in the spirit of the BC logic, we interpret terms through deterministic Polynomial Time Turing Machines (PTTM for short) that are parameterized by an infinite random tape and a security parameter. Providing the same random tape to all PTTMs allows to obtain a consistent interpretation of names.

Messages are thus interpreted through deterministic PTTMs, parameterized by:

- ρ_s, a random tape for secret names (e.g. secret keys);
- 1^η, the security parameter.

We then leverage the notion of a *functional model* $\mathcal{M}_f$, a library implementing the function symbols and names that are used in the protocol: for each function symbol f (encryption, signature,...), $\mathcal{A}_f$ is a PTTM, which we view as a deterministic machine with an infinite random tape and taking as input the security parameter. The functional model also contains a PTTM $\mathcal{A}_n$ for each $n \in \mathcal{N}$, which will extract from the random tape a bitstring of length η. We give η in unary to the PTTMs as they are expected to be polynomial time w.r.t. η in the computational model.

Definition 2.7. A *functional model* $\mathcal{M}_f$ is a set of PTTMs, one for each name and symbol function, such that:

1. if $n \in \mathcal{N}$, n is associated to the machine $\mathcal{A}_n$ that on input $(1^\eta, \rho_s)$ extracts a word of length η from the tape ρ_s. Different names extract disjoint parts of the random tape.
2. if $f \in \Sigma$ is of arity n, f is associated to a machine $\mathcal{A}_f$ which, on input 1^η, expects n more bitstrings, and does not use ρ_s. Intuitively, the functions are completely deterministic, and if randomness is required, it should be given explicitly as an argument to the function symbol.

Given an assignment σ of variables to bitstrings, the random tape ρ_s, a security parameter $\eta \in \mathbb{N}$ and a functional model $\mathcal{M}_f$, the (evaluation of the) interpretation of a term t is inductively defined as follows:

- $[\![n]\!]^{\eta,\sigma}_{\mathcal{M}_f,\rho_s} := \mathcal{A}_n(1^\eta, \rho_s)$ if $n \in \mathcal{N}$
- $[\![x]\!]^{\eta,\sigma}_{\mathcal{M}_f,\rho_s} = (x\sigma)$ if $x \in \mathcal{X}$
- $[\![f(\overline{u})]\!]^{\eta,\sigma}_{\mathcal{M}_f,\rho_s} = \mathcal{A}_f(1^\eta, [\![\overline{u}]\!]^{\eta,\sigma}_{\mathcal{M}_f,\rho_s})$ if $f \in \Sigma$

Given a functional model $\mathcal{M}_f$, a security parameter η and an infinite random tape ρ_s, we thus fix D as $\{0,1\}^*$, the set of bitstrings equipped with syntactic equality, and $[\![t]\!]^\sigma = [\![t]\!]^{\eta,\sigma}_{\mathcal{M}_f,\rho_s}$. The attacker is then any PTTM, parameterized by an infinite random tape ρ_r and a security parameter 1^η.

2.3.2 Computational Indistinguishability

Section Summary

We define protocol oracles that reflect the behaviour of the abstract semantics. Given two protocol oracles, indistinguishability corresponds to any attacker having access to either of the two oracles, deciding with at most negligible probability with which they interact with.

In the classical game-based semantics of the computational model, the interactions between an attacker and a protocol are described using a game, i.e., a list of execution instructions and queries to the attacker. Rather than taking this point of view, we consider the equivalent idea of using oracles given to Turing Machines to model the interactions between the protocol and the attacker. Oracles are more suited for the design of a composition framework. For instance, giving access to two protocol oracles to the attacker is equivalent to giving access to the oracle realizing the parallel composition of the protocols.

However, providing a protocol as an oracle is possible only for a subclass of protocols, that we call *action determinate* ([BDH15]). The idea is that the behaviour of the oracle should be deterministic, i.e., completely defined by the action of the attacker. In particular, no two inputs on the same channel should be possible at any given time.

Definition 2.8. A protocol is action determinate if $\emptyset, (P, \emptyset)$ cannot be reduced w.r.t. $\xrightarrow{!}$, and for any attacker $\mathcal{A}$ there exists a unique reduction $\emptyset, (P, \emptyset) \xrightarrow[\mathcal{A}]{*}$.

Technical Details

This notion of action determinate is not equivalent to the one of [BDH15], but implies it. As we have assumed that internal reduction of processes are always performed as much as possible, we only have to consider input channels to obtain a usable notion of determinism. We choose this version for its simplicity, which still provides a satisfactory level of expressiveness. One of the restrictions implied by the definition is that a protocol cannot start by an output, e.g. all outputs must be preceded by an input (which can be a trivial one).

We instantiate a protocol with an oracle, taking as input the next input computed by the attacker, along with the history of all the previous queries. Looking forward, all definitions and Lemmas below will be extended to support an extra stateless oracle in Section 5.2. In this first definition, protocol oracles can be arbitrary functions, and in particular do not have any computational

restrictions. We provide later the construction of the protocol oracle corresponding to a protocol given in our syntax. Protocol oracles based on actual protocols will by construction run in polynomial time.

Definition 2.9 (Protocol Oracle)**.** A *Polynomial Time Oracle Machine* (PTOM) is a Turing machine $\mathcal{A}^{\mathcal{O}_P}$ equipped with:

- an input/working/output tape (as usual; it is read/write);
- a read-only random tape ρ_r (attacker's coins);
- a protocol oracle read-only random tape ρ_s (not accessible by the Turing Machine);
- a protocol oracle input tape;
- a protocol oracle history tape θ;
- a protocol oracle output tape.

A *protocol oracle* $\mathcal{O}_P$ is a function that takes as input a tuple $(\overline{w}, \theta)$, and is parameterized by a security parameter η and a secret random tape ρ_s. Besides the usual moves of a multiple tape Turing machine (that respect the read/write constraints above), the machine may call the oracle, in which case there is a single move from the current configuration to a configuration, in which only the protocol oracle output tape and protocol oracle history tape are modified (and the control state):

- the content of the oracle output tape is set to $\mathcal{O}_P((\overline{w}, \theta), \rho_s, \eta)$ where $\overline{w}$ is the content of the oracle input tape, θ the content of the oracle history tape, and ρ_s the protocol oracle read-only random tape.
- the history tape is updated by appending the content of the oracle input tape.

Such PTOMs can be written with all the parameters explicit, e.g. as $\mathcal{A}^{\mathcal{O}_P(\rho_s,\eta)}(\rho_r, \rho_s)$, or by omitting some parameters when they are implicit from the context. In general, protocols are stateful, and a way to store this state is required. Rather than providing an explicit notion of state, we choose to store in an history tape the list of all previous inputs. The protocol can then, based on its previous inputs, recompute the corresponding state. This modelling does not correspond the real life behaviour of protocols in term of running time, but allows for a simpler expression of the oracles. Given a protocol P and a functional model $\mathcal{M}_f$, the protocol oracle $\mathcal{O}_P$ is such that given a query m with history h the oracle replies what would be the output of P, given the successive inputs h, m. It also appends the query m to the history tape.

Definition 2.10. Given an action determinate protocol P, a functional model $\mathcal{M}_f$, a security parameter $\eta \in \mathbb{N}$ and a random tape ρ_s, O_P is the protocol oracle, which, given ρ_s and a history $\theta = \{o_1, ..., o_n\} \in (\{0,1\}^*)^n$, on a query m:

- recomputes the control state q of the protocol using the history tape;
- set $\phi := \{x_1 \mapsto o_1, \ldots, x_n \mapsto o_n, x_{n+1} \mapsto m\}$;
- selects the (only) executable input transition of P (defined by action determinism);
- outputs the corresponding output and appends it to the history tape.

An oracle may implement multiple parallel protocols: the oracle $\mathcal{O}_{\langle P_1,\ldots,P_n\rangle}$ first checks which P_i is queried (there is at most one such i, by action determinism) and then replies as $\mathcal{O}_{P_i}$.

Definition 2.11. For any protocols $P_1, \ldots, P_n$ such that $\forall 1 \leq i < j \leq n.\ \mathcal{C}(P_i) \cap \mathcal{C}(P_j) = \emptyset$, we define the oracle $< \mathcal{O}_{P_1}, \ldots, \mathcal{O}_{P_n} > (\rho_s, \theta)$ which on input *query*:

- checks if its input is of the form $query := (channel, mess)$;
- computes i such that $channel \in C(P_i)$, and reject if there is no such i;
- computes the projection θ_i of its history θ such that $\theta_i = \{(channel, mess) \in \theta | channel \in \mathcal{C}(P_i)\}$;
- return the value of $\mathcal{O}_{P_i}(\rho_s, \theta_i)(mess)$.

We will often write $\mathcal{A}^{\mathcal{O}_{P_1},\ldots\mathcal{O}_{P_n}}(\omega, \rho_r)$ for $\mathcal{A}^{<\mathcal{O}_{P_1},\ldots\mathcal{O}_{P_n}>}(\omega, \rho_r)$. This finally allows us to introduce the classical notion of computational indistinguishability.

Definition 2.12. Given a functional model $\mathcal{M}^f$ and protocols P, Q, we write $P \cong Q$ if for every PTOM $\mathcal{A}$, the attacker's advantage $\mathrm{Adv}^{P\cong Q}$ equal to

$$|\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O}_P(\rho_s)}(\rho_r, 1^\eta) = 1\} - \mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O}_Q}(\rho_r, 1^\eta) = 1\}|$$

is negligible in η.

This definition quantifies universally over all polynomial-time attackers. In practice, we assume that no attacker can break a specific cryptographic primitives, and perform an indistinguishability proof under this assumption. Some protocols are however unconditionally indistinguishable.

AS an example, we consider the unforgeability axioms for signatures, called the EUF-CMA axiom [GMR88]. We informally use the classical game based description to match the classical definitions of the axiom.

Definition 2.13. A signature scheme $(\mathsf{Sign}, \mathsf{Vrfy})$ is EUF-CMA secure for an interpretation of keys $\mathcal{A}_{sk}$ if, for any PTTM $\mathcal{A}$, the game described in Figure 2.7 returns true with probability (over ρ_r, ρ_s) negligible in η.

For a fixed signing algorithm Sign and a fixed secret key sk, the attacker is given access to an oracle that performs signatures. The attacker wins the game if they can provide a message that corresponds to a valid signature for Vrfy, and such that it was not queried to the signing oracle. This means that the attacker can compute the forgery of a signature, without having access to the secret key.

Game $\mathrm{EUF\text{-}CMA}^{\Sigma,\mathcal{A}}_{sk}(\eta, \rho_r, \rho_s)$:	**Oracle** $\mathsf{Sign}(\mathsf{m})$:
$\mathsf{List} \leftarrow []$	$\mathsf{List} \leftarrow (\mathsf{m} : \mathsf{List})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow (\llbracket pk \rrbracket_{\rho_s}, \llbracket sk \rrbracket_{\rho_s})$	$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$
$(\mathsf{m}, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}}(\mathsf{pk}, \eta, \rho_r)$	Return σ
Return $\mathsf{Vrfy}(\mathsf{pk}, \mathsf{m}, \sigma) \wedge \mathsf{m} \notin \mathsf{List}$	

Figure 2.7: Game for Unforgeability (EUF-CMA)

Example 2.7. For two names n and m, we have that $\mathbf{out}(c, n) \cong \mathbf{out}(c, m)$. Intuitively, without any further information, no attacker can distinguish two random samplings. if we have a boolean function f over names, and an extra name b, we also have $\mathbf{out}(c, n) \cong \mathbf{if}\ f(b) = \mathsf{true}\ \mathbf{then}\ \mathbf{out}(c, n)\ \mathbf{else}\ \mathbf{out}(c, m)$. Without further assumptions and contrary to the symbolic model, we do not have $\mathbf{out}(c, \mathsf{enc}(m, r, sk)).\mathbf{out}(c, m) \cong \mathbf{out}(c, n).\mathbf{out}(c, m)$.

Technical Details

Notice that this definition extends to multiple protocol oracles. It relies on the fact that indistinguishability of an oracle enabling protocols in parallel corresponds to the indistinguishability of the multiple oracles in parallel. In other term, the oracle implementing the behaviour of multiple oracle in parallel behaves the same as the oracle implementing the parallel of our calculus.

Lemma 2.1. *For protocols P, Q, A, B, and a list $\mathcal{O}_l$ of protocol oracles,*

$$|\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O}_l,\mathcal{O}_{A\|P}(\rho_s)}(\rho_r, 1^\eta) = 1\} - \mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O}_l,\mathcal{O}_{B\|Q}(\rho_s)}(\rho_r, 1^\eta) = 1\}| = |\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O}_l,\mathcal{O}_A(\rho_s),\mathcal{O}_P(\rho_s)}(\rho_r, 1^\eta) = 1\} - \mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O}_l,\mathcal{O}_B(\rho_s),\mathcal{O}_Q(\rho_s)}(\rho_r, 1^\eta) = 1\}|$$

We do not provide a proof of this Lemma, as we will prove a stronger version when tackling composition in Section 5.2.

2.4 The BC Logic

Section Summary

In the BC logic, the knowledge of the attacker obtained during the execution of a protocol is completely modelled in terms, using uninterpreted function symbols to model attacker computations. Then, reasoning about those terms in a first-order logic whose models are PTTMs, we can obtain proofs of computational indistinguishability.

We present a proof technique based on the BC logic that can be used to perform proofs of computational indistinguishability for bounded protocols. Given a protocol, we start by considering abstract executions where the scheduling is fixed, but the attacker messages are left unspecified through uninterpreted function symbols. Then, for each scheduling we obtain a frame, which is simply a sequence of terms. Introducing a logic with a predicate for indistinguishability, we are able to reason over such terms soundly w.r.t. computational indistinguishability. Remark that this proof technique is only valid for bounded protocols, as we need to enumerate their set of possible execution flow. We also choose a more restricted approach compared to the BC logic: we can only prove indistinguishability for protocols that share the same set of execution flows. It is indeed challenging to perform indistinguishability proofs for protocols without the same execution flow. We perform this restriction in order to design the first mechanized approach to the BC logic in Part IV.

2.4.1 From Protocols to Terms

Attacker function symbols For any execution of the protocol, the attacker knowledge is modelled using the frame. In the previous models, given an attacker, this frame was completely defined, either as a sequence of bitstrings (the computational model) or of ground terms (the symbolic model). Here, we want to reason about frames in an abstract way, proving indistinguishability for all attackers. Thus, rather than asking for explicit computations from the attacker, we will create a sequence of terms depending on uninterpreted function symbols, which will be interpreted later on an arbitrary PTTM. We assume from now on that we have a set of function symbols $\mathcal{G}$, used to represent the attacker's computations.

Example 2.8. The terms representing the attacker interactions with the protocol P of Example 2.1 are:

$$\mathtt{enc}(g_0(), r, sk), \mathtt{enc}(g_1(\mathtt{enc}(g_0(), k, r)), r', sk)$$

where $g_0, g_1 \in \mathcal{G}$. The first attacker input $g_0()$ is computed without any prior knowledge, while the second attacker input $g_1(\mathtt{enc}(g_0(), r, sk))$ depends on the first output of the protocol.

Notice from the previous example the recursive behaviour: to model a frame $\phi_n = t_1, \ldots, t_n$, the function modelling the attacker's computation g_i in t_i is given the previous frame ϕ_{i-1} as argument. We build iteratively a sequence of messages depending on abstract function symbols, giving as argument to those function the sequence up to this point. It is then possible to give one PTTM meant to interpret each function symbol, and if they are all parameterized by a shared random tape, each PTTM can completely recompute the state of the previous PTTMs before performing a new computation. Thus, this modelling is equivalent to having a single PTTM computing in sequence all the inputs of the frame and maintaining its internal state: we are essentially cutting a

PTTM into many smaller ones, each performing all the computations of the main one up to some message.

Those uninterpreted function symbols allow to model a frame for a given scheduling of the communications, i.e., a sequence of input channels. If the protocols are finite, e.g., do not contain unbounded replication, it is possible to produce a frame for each possible scheduling. Thus, given a protocol, one can produce a set of frames that models all possible executions of the protocol.

Conditional branchings To correctly model the protocols, one last difficulty remains: conditional branchings. Indeed, given a scheduling, an input on some channel may produce distinct terms based on some conditions on the input. In order to have a single frame corresponding to a scheduling, the idea is then to push the conditionals in the terms. Thus, in addition to the **if** _ **then** _ **else** _ of the protocol syntax, we now extend the syntax of terms so that we may write in them conditionals. In other terms, we do not see conditional branchings as part of the protocol control flow, but only as part of the messages that are produced. To this end, we now assume that Σ always contain the following function symbols:

- constants true and false;
- a symbol ite of arity three;
- a unary symbol $\dot{\neg}$;
- binary symbols $\doteq, \dot{\wedge}, \dot{\vee}, \dot{\rightarrow}$.

All those function symbols are assumed to have a fixed interpretation in all functional model. This means that there semantics always correspond to the one of propositional logic.

For simplicity, we will write $\mathsf{if}\ b\ \mathsf{then}\ t\ \mathsf{else}\ s$ for $\mathsf{ite}(b,t,s)$, but bear in mind that it is not anymore a construct of protocols, but a function symbol that will be interpreted in the natural way in all functional models. If omitted, the **else** branch will contain **cst**.

Example 2.9. We define a protocol that allows to obtain the encryption of an input message x, if for some function symbol f, $f(x)$ is equal to some constant true. We do not give any precise meaning to f, it could for instance check that the first byte of x is equal to 1.

$$\mathbf{in}(c,x); \mathbf{if}\ f(x) = \mathsf{true}\ \mathbf{then}\ \mathbf{out}(c, \mathsf{enc}(x,r,sk))\ \mathbf{else}\ \mathbf{out}(c,\mathsf{fail})$$

Based on the previous discussion, one can see this protocol as the following equivalent protocol:

$$\mathbf{in}(c,x); \mathbf{out}(c, \mathsf{if}\ f(x) = \mathsf{true}\ \mathsf{then}\ \mathsf{enc}(x,r,sk)\ \mathsf{else}\ \mathsf{fail})$$

And the corresponding frame is:

$$\mathsf{if}\ f(g_0()) \doteq \mathsf{true}\ \mathsf{then}\ \mathsf{enc}(g_0(),r,sk)\ \mathsf{else}\ \mathsf{fail})$$

We now use our execution model to build such frames. Similarly to the symbolic model, we interpret terms as terms, but adding attacker function symbols. D is $\mathcal{T}(\Sigma,\mathcal{N})$, $[\![t]\!]^\sigma = t\sigma$, and attackers are function such that $\mathcal{A}(\phi_n) = (c, g_n(\phi_n))$ or $\mathcal{A}(\phi_n) = (i,n)$. Notice that as outlined previously, the execution only depends on the scheduling of the attacker as the computed inputs are abstracted away. We call such an attacker a *scheduler*, completely defined by a sequence of values in $C \cup (I \times \mathbb{N})$. Because the inputs of the attacker given to the protocol are of the form $g_n(\phi_n)$, the In rule will construct frames that are of the previously demonstrated form. Then, the sequence of values in $C \cup (I \times \mathbb{N})$ can be seen as specifying a possible execution flow of the protocol, where the scheduling is fixed, but not the computations of the attacker.

Definition 2.14. A protocol that is action determinate, finite, and without conditional branchings is called *simple*. Given a simple protocol P, a scheduler $\mathcal{A}$ and its corresponding sequence τ of n values in $C \cup (I \times \mathbb{N})$, we call τ an *observable trace* of P if P can be reduced according to this scheduling, i.e., there exists a final configuration $(P_1, \sigma_1) \| \dots \| (P_k, \sigma_k)$ such that:

$$\emptyset, (P, \emptyset) \xrightarrow[\mathcal{A}]{} \dots, \xrightarrow[\mathcal{A}]{} \phi_n, (P_1, \sigma_1) \| \dots \| (P_k, \sigma_k)$$

We denote by ϕ_P^τ the corresponding frame ϕ_n, called a *concrete trace* of P.

Recall that considering that a protocol is without conditional branchings is not a restriction, as we express the conditional in the terms. A formal translation from protocols to protocols without conditional branching is straightforward.

Technical Details

Given a protocol, we produce a sequence of terms modelling the corresponding frame for each scheduling. Thus, we can model all behaviours of the protocol in a finite set of term sequences. In the first paper of the BC logic for indistinguishability ([BC14a]), all the behaviours of the protocol were captured in a single term, by using a technique called folding. The idea is to push the scheduling in the terms, using conditionals over additional uninterpreted function symbols. This produces a large term that can be difficult to read, and manipulate in proofs. Moreover, most proofs in the BC logic start by performing a case study on all possible values of the symbols modeling the scheduling, thus yielding our set of frames. For those reasons, we choose to avoid this folding step, and directly produce a frame for each observable trace. However, we cannot anymore reason about protocols that do not have the same set of observable traces.

Section Summary

Given a simple protocol (a deterministic and finite protocol where all conditionals are modelled in the messages using a dedicated function symbol), one can produce for each scheduling a frame modelling all the possible attacker knowledge that can be obtained for this scheduling.

2.4.2 A Logic over Terms

Now that we can extract from a protocol the sequences of terms that model all possible executions of the protocol, we provide a logic allowing to reason over such sequences, and most notably to prove computational indistinguishability of two protocols by reasoning on their concrete traces in a first order logic. We thus define semantics of terms for this logic. Those semantics should not be confused with the previously provided semantics: in the BC logic we only use the calculus to define the concrete traces of a protocol without interpreting the terms; the semantics of terms of the BC logic corresponds to the semantics of a first order logic.

Semantics of terms

Definition 2.15. A *computational model* $\mathcal{M}$ is an extension of a functional model $\mathcal{M}_f$, which provides an additional PTTM $\mathcal{A}_g$ for each symbol $g \in \mathcal{G}$, that takes as input an infinite random tape ρ_r, a security parameter 1^η and a sequence of bitstrings.

We define the interpretation of extended terms as, given $\mathcal{M}$, η, σ (which is now a mapping from variables to ground terms), ρ_s and ρ_r:

- $[\![n]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r} := \mathcal{A}_n(1^\eta, \rho_s)$ if $n \in \mathcal{N}$
- $[\![x]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r} = [\![x\sigma]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r}$ if $x \in \mathcal{X}$
- $[\![f(\overline{u})]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r} = \mathcal{A}_f([\![\overline{u}]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r})$ if $f \in \Sigma$
- $[\![g(\overline{u})]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r} = \mathcal{A}_g([\![\overline{u}]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r}, \rho_r, 1^\eta)$ if $g \in \mathcal{G}$

The attacker is basically given the interpretation of its different inputs, but also the security parameter and its own randomness. Notice that all PTTMs interpreting the attacker's function symbols are given the same random tape as parameter, thus modelling accurately the behaviour previously defined, where a single attacker against a protocol can be seen as a set of attackers, each recomputing all the previous interactions.

Technical Details

We defined a notion of computational model which is not directly a valid model for a first order logic. Indeed, in a logic, all terms are of the same sort and must be interpreted over the same domain, but here we have PTTMs that do not have the same parameters. The actual domain of interpretation of terms (for the logic) is the set of PTTMs taking as input a security parameter 1^η, two random tapes ρ_s and ρ_r and possibly a sequence of bitstrings. In our case, we further restrict the PTTMS so that for instance, $\mathcal{A}_n$ that interpret the name n does not access ρ_r (the attacker's randomness), and we simply write $\mathcal{A}_n(1^\eta, \rho_s)$ instead of $\mathcal{A}_n(1^\eta, \rho_r, \rho_s)$. Furthermore, we only give the interpretation in the case of a σ that maps variables to ground terms, while for the first order logic it can be any mapping from variables to the domain of interpretation of terms.

We can easily derive a valid notion of models for a first-order logic from our definition of computational models. Our definition is sufficient to give the intuition of the interpretation of terms in the case where η is given. It is also sufficient to give the interpretation of the indistinguishability predicate in all our use case, as in practice we only construct formulas without free variables.

Interpretation of formulas Atomic formulas of the logic are built using a set of predicate symbols $\sim_n$ of arity $2n$. Given terms $t_1, \ldots, t_n, s_1, \ldots, s_n$, the predicate $\sim_n (t_1, \ldots, t_n, s_1, \ldots, s_n)$ will be interpreted as computational indistinguishability between the two sequences of terms. We use infix notation, and always omit n as it is clear from the context, thus denoting the previous equivalence by $t_1, \ldots, t_n \sim s_1, \ldots, s_n$. The first order formulas are then built using the usual logical connectives $\vee, \wedge, \top, \bot, \Rightarrow, \exists, \forall, \neg$.

Definition 2.16. Given a computational model $\mathcal{M}$, two sequences of ground terms $\overline{t}$, $\overline{u}$, and an assignment σ of the free variables of $\overline{t}, \overline{u}$ to ground terms, $\overline{t} \sim \overline{u}$ is satisfied by $\mathcal{M}$ and σ, denoted by $\mathcal{M}, \sigma \models \overline{t} \sim \overline{u}$, if, for every polynomial time oracle Turing machine $\mathcal{A}$,

$$|\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}([\![\overline{t}]\!]^{\sigma,\eta}_{\mathcal{M},\rho_s,\rho_r}, \rho_r, 1^\eta) = 1\} - \mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}([\![\overline{u}]\!]^{\sigma,\eta}_{\mathcal{M},\rho_s,\rho_r}, \rho_r, 1^\eta) = 1\}|$$

is negligible in η. Here, ρ_s and ρ_r are drawn according to a distribution such that every finite prefix is uniformly sampled. (PTIME computable distributions have to be made explicit through function symbols). If $\mathcal{M}, \sigma \models \overline{t} \sim \overline{u}$ holds for all σ, we write $\mathcal{M} \models \overline{t} \sim \overline{u}$. The satisfaction relation is extended to full first-order logic as usual.

Example 2.10. Considering again the two first indistinguishability of Example 2.7, $\mathbf{out}(c, n) \cong \mathbf{out}(c, m)$ is expressed as the formula $n \sim m$, and $\mathbf{out}(c, n) \cong \mathbf{if}\ f(b)\ \mathbf{then}\ \mathbf{out}(c, n)\ \mathbf{else}\ \mathbf{out}(c, m)$ as the formula $n \sim \mathbf{if}\ f(b)\ \mathbf{then}\ n\ \mathbf{else}\ m$. Proving the validity of those formulas requires axioms and logical deduction rules.

The random tape given to the adversary machine (modeling the distinguisher) is identical to the one used for the interpretation of terms, and thus to the one given to the other attacker machines in the terms. Then, the distinguisher and the attackers computing the protocol inputs can once again be seen as modelling a single PTTM. This links our notion of satisfiability with computational indistinguishability, where the computational attacker that distinguishes and computes the inputs of the protocol is split into multiple PTTMs, one for each message and one for the final output.

Axioms and logical rules We outline some of the axioms used to derive the validity of formulas in the BC logic. A more extensive presentation of the rules allowing to reason in this logic is postponed to Part IV. In this part, we only provide intuition about why those axioms can indeed be used in the BC logic to derive computational indistinguishability.

A first example corresponds to the fact that any term is indistinguishable to the same term where all occurrences of a name are replaced by a fresh name. Essentially, we can perform α-renaming on terms. This correctly models the fact that names are only pointers to random samplings, and they are interchangeable.

Example 2.11. The (recursive) set of formulas corresponding to the α-renaming axiom is given by the set of formulas $t \sim t\{n \mapsto n'\}$ for each term $t \in \mathcal{T}(\Sigma, \mathcal{N})$, each name $n \in \mathcal{N}$, and any name n' that does not appear in t.

A widely used technique is that if two sequences of terms are indistinguishable, then applying any deterministic function to the sequences yields two indistinguishable terms.

Example 2.12. The (recursive) set of formulas corresponding to the function application axiom is given by the set of formulas $t_1, \dots, t_n \sim t'_1, \dots, t'_n \Rightarrow f(t_1, \dots, t_n) \sim f(t'_1, \dots, t'_n)$ for any terms $t_1, \dots, t_n, t'_1, \dots, t'_n \in \mathcal{T}(\Sigma, \mathcal{N})$ and function symbol $f \in \Sigma$.

Finally, we present a more complex example, based on the EUF-CMA axiom of Definition 2.13. We can transpose this axiom in the BC logic, saying that any term that is a valid signature, must in fact be equal to the signature of a message appearing in the term. We use the function symbols $\mathtt{sign}$,$\mathtt{pk}$ and $\mathtt{checksign}$, where intuitively $\mathtt{checksign}(x, \mathtt{pk}(sk))$ should only be equal to true if x is equal to a message of the form $\mathtt{sign}(x, sk)$. We denote by $\mathtt{St}(t)$ the set of sub-terms of t, which is defined completely syntactically.

Definition 2.17. Given a name sk, we define the axiom scheme EUF-CMA$_{sk}$ as, for any term t such that sk is only in key position:

$$\begin{array}{l} \textbf{if } (\mathtt{checksign}(t, \mathtt{pk}(sk))) \textbf{ then} \\ \quad \dot{\bigvee}_{\mathtt{sign}(x,sk) \in \mathtt{St}(t)} (t \doteq \mathtt{sign}(x, sk)) \quad \sim \top \\ \textbf{else } \top \end{array}$$

By saying that the formula on the left hand-side is indistinguishable from true, we say that the formula is true with overwhelming probability.

Example 2.13. Let us consider the term $g(\mathtt{sign}(m, sk))$, where g models some computation of the attacker. The only signature appearing as a sub-term of $g(\mathtt{sign}(m, sk))$ is $\mathtt{sign}(m, sk)$. Thus, with the EUF-CMA axiom, we could conclude that

$$\begin{array}{l} \textbf{if } (\mathtt{checksign}(g(\mathtt{sign}(m, sk)), \mathtt{pk}(sk))) \textbf{ then} \\ \quad (g(\mathtt{sign}(m, sk)) \doteq \mathtt{sign}(m, sk)) \quad \sim \top \\ \textbf{else } \top \end{array}$$

Thus, in any protocol (seen as a term) where this message is verified as a valid signature, we could use the fact that it is equal with overwhelming probability to $\mathtt{sign}(m, sk)$.

Computational Soundness Given a functional model $\mathcal{M}_f$, we can consider all possible attackers by considering all computational models $\mathcal{M}$ that extend $\mathcal{M}_f$ by providing interpretations for the attacker's function symbols. We denote this relation by $\mathcal{M} \supset \mathcal{M}_f$, which is formally defined as set inclusion. Given simple protocols P, Q and a functional model $\mathcal{M}_f$, if the protocols have the same set of observable trace, we can try to prove the indistinguishability of each concrete trace in the logic. By requiring that the protocols share the same set of observable traces, we design a proof technique restricted to a subclass of protocols. As we consider simple protocols in this part, this is in fact not a restriction, as we expect that conditionals are in the terms, and protocols that do not have the same set of observable traces are naturally distinguishable.

Lemma 2.2. *Given two simple protocols P, Q with the same set T of observable traces, random tapes ρ_r, ρ_s and a functional model $\mathcal{M}_f$, we have:*

$$\forall \tau \in T, \forall \mathcal{M} \supset \mathcal{M}_f. \quad \mathcal{M} \models \phi_P^\tau \sim \phi_Q^\tau$$

$$\Rightarrow$$

$$P \cong Q$$

Sketch of Proof. We do not provide the proof, that will be performed in Section 5.2 in a more general setting. As we only consider finite protocols the set of observable traces is also finite. While it is of exponential size w.r.t. the protocol, the size is fixed w.r.t. the security parameter, which allows us to perform the proof. Thus, if there exists a distinguisher for P and Q, its advantage must be non negligible for at least one abstract trace. For this given abstract trace, we can construct a model that negates the desired formula by splitting the distinguisher into multiple attackers. ■

This Lemma does not provides us with a usable proof technique, as it is not possible in practice to make a proof for each computational model. However, as we are in a first order logic, if one can find a set of axioms that are satisfied by all computational models, a proof under such axioms implies that the formula is satisfied by all computational models. We write $Ax \models \phi$ if the set of formulas Ax and the formula $\neg\phi$ are inconsistent. We want to consider axioms that hold for a family of computational models, for instance all $\mathcal{M} \supset \mathcal{M}_f$ given a functional model $\mathcal{M}_f$.

Definition 2.18. Given a family of computational models $\mathcal{F}$, a set of first order formulas A is *sound* (w.r.t. $\mathcal{F}$) if, for every $\psi \in A$, every $\mathcal{M} \in \mathcal{F}$, $\mathcal{M} \models \psi$.

We can finally perform a proof of indistinguishability using such axioms.

Theorem 2.1. *Given two simple protocols P, Q with the same set of observable traces T, a set of axioms A and a functional model $\mathcal{M}^f$, if we assume that:*

- *A is sound w.r.t. $\mathcal{F} = \{\mathcal{M} \supset \mathcal{M}^f\}$;*
- *for all $\tau \in T$, $A \models \phi_P^\tau \sim \phi_Q^\tau$*

Then $P \cong Q$.

The proof will be performed in Section 5.2 in a more general setting. In practice, we can for instance assume that all PTTMs satisfy the classical EUF-CMA axiom), and then prove that the first-order formulas of Definition 2.17 define a sound BC version of the EUF-CMA axiom. Performing proofs under such an axiom yields indistinguishability assuming the classical EUF-CMA axiom.

Part I

Extensive

In which we try to demonstrate how one may carry out a so called extensive analysis of a protocol

3 A Symbolic Model for Multi-Factor Authentication

> If you are asked for the password, type in "password".
>
> *(Ockham's Razor)*

3.1 Introduction

To provide strong guarantees about the security of a protocol, we should consider all combinations of attacker capabilities. Ideally, an extensive analysis should provide for each possible threat model either an effective attack against the protocol or a security proof. To achieve this level of precision requires highly automated tool: it can nowadays only be performed in the Symbolic model.

In this chapter, we provide an example of a detailed and modular threat model that can be leveraged to perform an extensive analysis in the symbolic model. We focus on authentication properties, a major concern: users need to authenticate to an increasing number of electronic services in everyday life, such as email and bank accounts, agendas, e-commerce sites, etc. Authentication generally requires a user to present an *authenticator*, that is "something the claimant possesses and controls (typically a cryptographic module or password) that is used to authenticate the claimant's identity" [GGF17]. Authenticators are often classified according to their *authentication factor*:

- what you know, e.g., a password, or a pin code;
- what you have, e.g., an access card or physical token;
- what you are, e.g., a biometric measurement.

Although these different mechanisms exist, passwords are still by far the most widely used mechanism, despite the fact that many problems with passwords were already identified in the late '70s when they were mainly used to grant login into a computer [MT79]. Since then, things have become worse: many people choose the same weak passwords for many purposes, and large password databases have been leaked. Studies have shown that the requirement to add special characters does not solve these problems, and the latest recommendations by NIST [GFN+17] even discourage this practice.

To palliate password weaknesses, multi-factor authentication protocols combine several authentication factors. Typically, instead of using only a login and password, the user proves possession of an additional device, such as there mobile phone or a dedicated authentication token. Two popular protocols are *Google 2-step* [G2s] (which actually regroups several mechanisms) and *FIDO's U2F* [Fid] (the version implemented by Yubico for their Security Keys), which is supported by many websites, including Google, Facebook, and GitHub. In (one version of) *Google 2-step*, the user receives a verification code on their phone that they must copy onto their computer, while *FIDO's U2F* requires the use of a specific USB token that must be plugged into the computer.

Multi-factor authentication (MFA) protocols thus provide complex situations where multiple agents are involved, and with protocols leveraging mechanisms that are very different in nature. To study such protocols, we provide a detailed threat model, trying to capture all possible levels of compromise of the multiple agents. We then explain how one can at a high level model each

component of the threat model in the symbolic model presented previously, after extending it to support communications over secret channels.

Chapter Summary

Multi-factor authentication strengthens authentication mechanisms by adding to passwords additional proofs of identity. They constitute complex protocols involving multiple agents. We present a detailed and modular threat model for those protocols. It takes into account communication through TLS channels in an abstract way, yet modelling interesting details such as session identifiers and TLS sessions with compromised agents. Moreover, we consider different levels of malwares in a systematic way by representing a system as a set of interfaces with access rights. Additionally, we allow the adversary to perform phishing and spoof fingerprints, and consider scenarios where a careless user does not perform expected checks. We formalize this model in the applied pi calculus, defining a formal and modular threat model.

3.1.1 Our Contributions

In classical protocol analysis, the attacker is supposed to control the communication network. However, the protocols we study in this part make extensive use of TLS communications and are supposed to provide security even if some devices are infected by malware.

We therefore propose a novel, detailed threat model for multi-factor authentication protocols which takes into account many additional threats.

- Compromised passwords: our basic assumption is that the user's password has been compromised. Otherwise multi-factor authentication would not be required.
- Network control: we define a *high-level model of TLS channels* that guarantees confidentiality and authentication of messages and additionally ensures, through inclusion of session ids, that messages of different TLS sessions cannot be mixed. Nevertheless, we allow the attacker to delay or block messages. Our model also contains a notion of *fingerprint* that is used in some protocols to identify machines, and we may give the adversary the power to spoof such fingerprints.
- Compromised platforms: we give a structured and fine-grained *model for malwares*. We take an abstract view of a system as a set of input and output interfaces, on which an adversary may have read or write access, depending on the particular malware.
- Human aspects: we take into account that most of these protocols require some interaction with the *human user*. We model that humans may not correctly perform these steps. Moreover, we model that a human may be a victim of *phishing*, or *pharming*, and hence willing to connect to and enter their credentials on a malicious website.
- *"Trust this computer mechanism"*: to increase usability, several websites, including Google and Facebook, offer the possibility to *trust* a given machine, so that the use of a second factor becomes unnecessary on these machines. We add this trust mechanism to our model.

We completely formalize these threat scenarios in the applied pi calculus.

Limitations

We do not generalize the modelling to the computational semantics, but restrict our analysis to the symbolic semantics. Indeed, for the high-level model of TLS, we need dynamic secret channels (secret channels that can become public), which would significantly increase the complexity of the computational semantics. Moreover, as we need to perform an efficient and automated analysis given the number of scenarios, the computational model is ill suited for this purpose at the moment.

3.1.2 Related Work

Bonneau et al. [BHO+12] propose a detailed framework to classify and compare web authentication protocols. They use it for an extensive analysis and compare many solutions for authentication. While the scope of their work is much broader, taking into account more protocols, as well as usability issues, our security analysis of a more specific set of protocols is more fine-grained in terms of malware and corruption scenarios. Basin and Cremers [BC14b] formalizes the notion of a protocol-security hierarchy, which provides for multiple adversaries the strengths and weaknesses of each protocol. Adversaries are defined in a modular way, combining multiple notions of key and state compromise. Our methodology is similar as we identify the largest threat scenarios that a protocol can tolerate through a modular combination of attacker capabilities.

Basin et al. [BRS16] studied how human errors could decrease security. Their model is more evolved than ours on this aspect. However, we consider more elaborate malwares and also check for a stronger authentication property: an attack where both a honest user and an attacker try to log into the honest user's account but only the attacker succeeds is not captured in [BRS16], as they simply check that every successful login was proceeded by an attempt from the corresponding user to login. As a side remark, notice that [BRS16] blurs the line between the Symbolic and Computational models, defining negatively what the human (which is a second attacker) cannot do. In the same vein, [BRS15] studies minimal topologies to establish secure channels between humans and servers. Their goal is to establish a secure channel, while we consider entity authentication. They consider authentic and confidential channels, which we extend by being more fine grained.

3.2 Multi-factor Authentication Protocols

Section Summary

Google 2-step relies on a cellphone as a second factor: it either sends a confirmation code via SMS, or asks for confirmation by touching the screen.*FIDO's U2F* provides a small USB token that can perform signatures after a button press. The signature can be performed over the authentication material and checked by the server.

3.2.1 *Google 2-step*

To improve security of user logins, Google proposes a two factor authentication mechanism called *Google 2-step* [G2s]. If enabled, a user may use there phone to confirm the login. On their website Google recalls several reasons why password-only authentication is not sufficient and states that "2-Step Verification can help keep bad guys out, even if they have your password". *Google 2-step* proposes several variants. Google's mechanisms are not documented, and were reverse engineered. Thus, some distance between our presentation and real life implementation may exist. The default mechanism sends to the user, by SMS, a verification code to be entered into there computer. An alternative is the "One-Tap" version, where the user simply presses a Yes button in a pop-up on there phone. The second version avoids to copy a code and is expected to improve the usability of the mechanism. This raises an interesting question about the trade-off between security and ease of use. We also present a more recent version of "One-Tap" that we dubbed "Double-Tap".

As Google does not provide any detailed specification of the different authentication mechanisms, the following presentations are based on reverse engineering. As the protocols are simple and do not contain complex cryptographic operations, the reverse engineering is rather straightforward, based on the operations visible by the user and behavioral tests. Notice though that we may have omitted some checks performed by the server, based on some information that is not entered

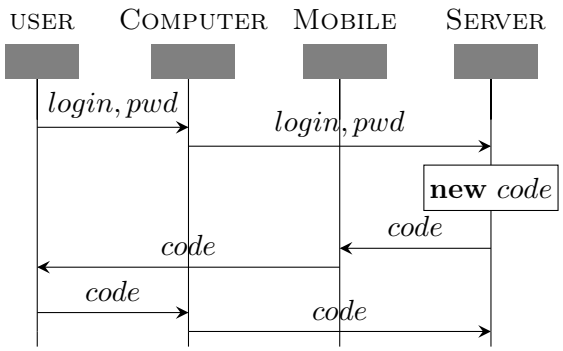


Figure 3.1: G2V Protocol

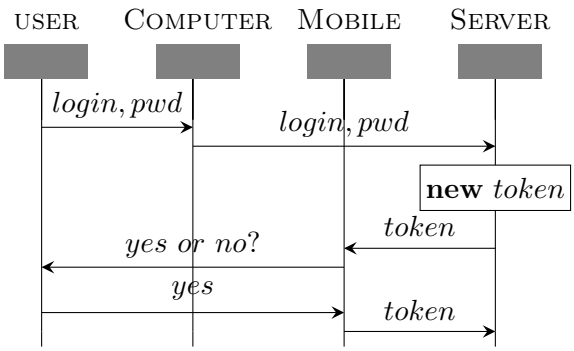


Figure 3.2: G2OT Protocol

by the user, such as the timing of the login. As we validated the attacks found systematically in a laboratory environment, our protocol models appear to be precise enough. All experiences presented in this Part were performed in January and February 2018.

***Google 2-step* with verification codes - g2V** In Figure 3.1 we depict the different steps of the protocol. All communications between the user's computer and the server are protected by TLS. The three main steps of the protocol are:

1. the user enters their login and password into their computer, which forwards the information to the server;
2. upon receiving login and password, the server checks them. In case of success, the server generates a fresh 6 digits code, and sends an SMS of the form *"G-****** is your Google verification code"* to the user's mobile phone;
3. the user then copies the code to their computer, which sends it to the server. If the correct code is received login is granted.

When the password is compromised, the security of the protocol only relies on the code sent on the SMS channel. Thus, if the attacker can intercept the code produced in step (2) before it is received by the server, the attacker could use the code to validate their own session and break the security. This could be done for instance by intercepting the SMS, compromising the phone with a malware, or through a key-logger on the user's computer.

***Google 2-step* with One-Tap - g2OT** In Figure 3.2 we present the One-Tap version of *Google 2-step*, the main steps being:

1. the user enters their login and password into their computer, which forwards the information to the server;
2. the server then creates a fresh random *token* that is sent to the user's mobile phone. Unlike in the previous version, the communication between the server and the phone is over a TLS channel rather than by SMS;
3. the phone displays a pop-up to the user who can then confirm the action or abort it, by choosing "Yes" or "No" respectively;
4. in case of confirmation the phone returns the token and login is granted.

Note that in its most basic version, the user only answers a yes/no question. Google announced in February 2017 [tea17] that the pop-up would also contain in the future a fingerprint of the computer, including information such as IP address, location and computer model. However this new version has yet to be implemented on some of the smartphones we used for tests. In the following we will analyse both versions, with (G2OT^{FPR}) and without (G2OT) the fingerprint. Remark that in steps (2) to (4), the authentication token is never sent to the computer. This is an important difference with the previous version, disabling attacks based on compromising the

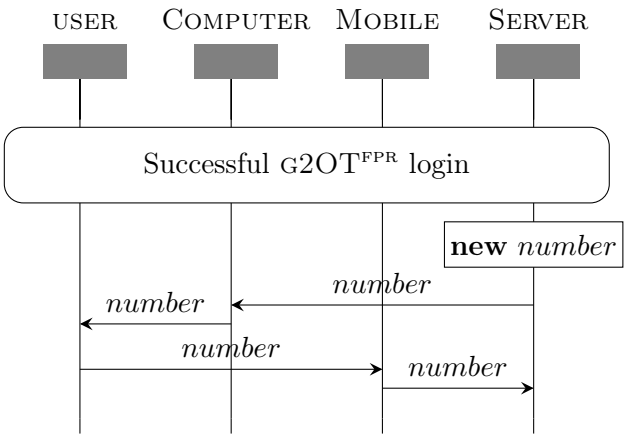


Figure 3.3: G2DT$^{\text{FPR}}$ Protocol

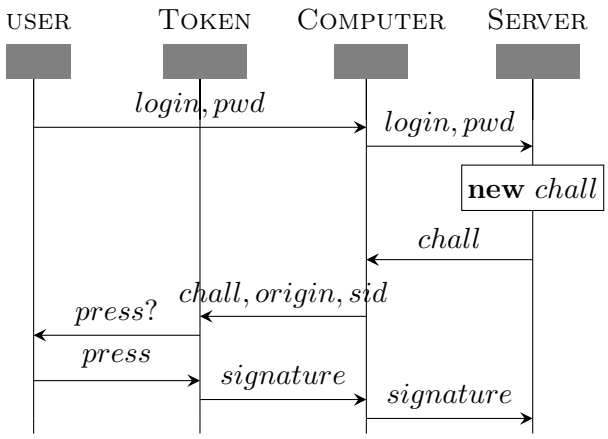


Figure 3.4: U2F Protocol

computer, e.g., a key-logger. The independence of the second factor with respect to the computer then improves the security. Adding a fingerprint to the screen additionally improves the security, as it allows the user to perform a suspicious login from an unknown location.

***Google 2-step* with Double-Tap - g2DT$^{\text{fpr}}$** The issue with One-Tap compared to the code version is that the user is likely to simply press "Yes" without reading any displayed information. To mitigate this issue, Google sometimes uses a version which we call Double-Tap. We were not able to find a public documentation of this variant, but we saw it at work in practice. The first step is the One-Tap protocol previously presented, with the display of the fingerprint. It is then followed by a second step, where a two digit number is displayed on the user's computer screen, and the same number is displayed on the user phone along with two other random numbers. The user is then asked to select on their phone the number displayed on their computer. This selection mechanism mimics the behaviour of a verification code displayed on the computer and that the user should enter on their phone, but with the benefits of greater simplicity and ease of use. If we abstract the selection mechanism used to simplify the user experience and simply consider that the user is entering the data on their phone, the protocol outline is shown in Figure 3.3.

3.2.2 *FIDO's Universal 2nd Factor* - U2F

FIDO is an alliance which aims at providing standards for secure authentication. They propose many solutions under the U2F, FIDO and FIDO2 [FID18; BLVGB+17] (also known as the WebAuthn) standards. We only study partially the Universal 2nd Factor (U2F) protocol [Fid], focusing on the version using a USB token as the second factor. More precisely, we study the implementation of the standard performed by the Yubico company, producing the Yubikey token. The U2F protocol relies on a token able to securely generate and store secret and public keys, and perform cryptographic operations using these keys. Moreover, the token has a button that a user must press to confirm a transaction. To enable second-factor authentication for a website, the token generates a key pair[1] and the public key is registered on the server. This operation is similar to the registration of a phone as a second factor in the case of a Google account. We must assume that this step was performed securely by the user at a time where their password was secure, and ideally at the time of the creation of the account. Else, no security may come from the second factor. Once the registration has been performed, the token can then be used for authenticating; the steps of the authentication protocol are presented in Figure 3.4, and can be explained as:

[1] In the case of the Yubikey token, the key is generated by hashing a fresh random with a fixed secret stored in the token.

1. the computer forwards the user's login and password to the server;
2. the server generates a challenge which is sent to the user's computer;
3. upon reception, the browser generates a payload containing the URL of the server, the challenge and the identifier of the current TLS session to be signed by the token;
4. the user confirms the transaction by pressing the token button;
5. the token signs the payload, and the signature is forwarded to the server for verification.

Compared to G2OT and G2DT$^{\text{FPR}}$, the second factor and the user's computer are not independent, which may lead to attacks base on malware on the computer. However, thanks to the signature of the payload, the signature sent back to the server is strongly linked to the current session, and session confusion is significantly harder. Moreover, as the signature includes the URL seen by the user, this may counter phishing attacks.

3.2.3 Disabling the Second Factor on Trusted Devices

When designing an authentication protocol, as also emphasized in [BHO+12], a key requirement should be usability. On a user's main computer, used on a daily basis, it may not be necessary to use a second factor: for instance, using a second factor each time a user pops there emails on there main laptop would be very cumbersome. This is why several providers, including Google and Facebook, propose to *trust* specific computers and disable the second factor authentication on these particular machines. This is done by checking a *"Trust this computer"* option when initiating a two-factor authenticated login on a given machine. Technically, the computer will be identified by a cookie and its *fingerprint*. A fingerprint typically includes information about the user's IP address, inferred location, OS or browser version, etc. As those elements will obviously change over time, in practice, a distance between fingerprints is evaluated, and if the fingerprint is too far from the expected one, the second factor authentication will be required. To the best of our knowledge, this feature is not documented and the full mechanism has not been studied previously even though it may lead to security issues. To capture such security issues we will include the *"Trust this computer"* mechanism in our analysis.

3.2.4 Token Binding

While cookies are a common mechanism widely used to remember a computer after a successful login, a new protocol called TokenBinding [PNB+18] is under development. Its usage is recommended by the FIDO standards, but providers are free to use it or not. After a successful login, a public key may be bound to the user account, and the corresponding secret key will be used to sign the session identifier of the following TLS sessions. It may be seen as a partial U2F where the keys are directly stored on the computer. We describe the protocol in Figure 3.5. If a computer has been successfully authenticated, the registration part of TokenBinding may be enabled and the computer may generate a new secret key, and simply send the corresponding public key to the server.

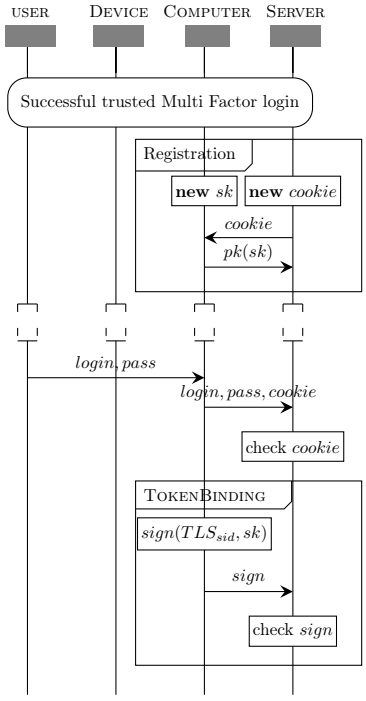


Figure 3.5: TokenBinding

In parallel, the server may send a classical cookie to the computer. For later logins, the server will ask for the cookie but also for the signature of the TLS session identifier by the registered public

key. We remark that the cookie and the signature may actually be sent at the same time, and TokenBinding thus does not require more communications than classical cookie authentication after the registration.

3.3 Threat Model

In order to conduct an in depth analysis of MFA protocols, we consider different threat models, types of attacks and corresponding attacker capabilities. We will consider a Dolev-Yao attacker [DY81] that controls any compromised parts and, classically, the network. However, many of the protocols we study use channels protected by TLS. The attacker may block a message, even if they cannot read or write on such channels. Moreover, as we are studying multi-factor authentication protocols, in order to assess additional protection offered by these protocols, we are interested in the case where the user's password has been compromised. Therefore, the most basic threat scenario we consider is the one where the attacker has (partial) control over the network, and knows the users' passwords.

There are however several ways the attacker can gain more power. Our aim is to present a detailed threat model, reflecting different attacker levels that may have more or less control over the user's computer, the network, or even over the user itself. Those levels aim at capturing the attacker capabilities that are necessary for a given attack.

Section Summary

We define a fine-grained threat model. The attacker may gain read/write or read-only access over all the communication channels of the user platform (USB, display, network, HDD). The user's phone may be compromised, the human may be subject to phishing or not perform some checks such as comparing two values, and the fingerprint of the platform might be spoofed.

3.3.1 Malware Based Scenarios

The first range of scenarios covers malwares that give an attacker control over parts of a user's device, also known as Man In The Machine attacks.

Systems as interfaces To give a principled model of malwares and what parts of a system the malware may control, we take an abstract view of a system as a set of *interfaces* on which the system receives inputs and sends outputs. Some interfaces may only be used for inputs, while other interfaces may be used for outputs, or both. For example the keyboard is an input interface, the display is an output interface, and the network is an input and output interface. Compromise of part of the system can then be formalized by giving an attacker read or write access to a given interface. On a secure system, the attacker has neither read nor write access on any interface. Conversely, on a fully compromised system the attacker has read-write access on all interfaces.

More formally we consider that for each interface the attacker may have no access (NA), read-only access (RO), write-only access (WO), or read-write access (RW). We may specify many different levels of malware by specifying for every interface two access levels, one for inputs and one for outputs on the interface. Obviously, for a given interface not all combinations

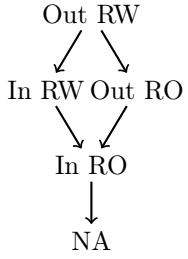


Figure 3.6: Access Lattice

need to be considered: a read-write access will yield a stronger threat model than read-only access, write-only or no access. We consider for each interface five levels, that can be organized as the lattice depicted in Figure 3.6.

Technical Details

We suppose in this work that it is harder to control the outputs of an interface than its inputs: therefore a given access level to the outputs implies the same access level on the interface inputs. Although not a limitation of our model, this choice is motivated by practical considerations. Running for instance a key-logger does not require specific rights, because the keyboard data is completely unprotected in the OS. FIDO devices are identified by the OS as a keyboard (at least on Linux systems). However, reading data sent by an application to a USB device, i.e., having read access on the USB interface's output, may require to corrupt the driver (or in the case of Linux enable the "USBmon" module) which requires specific privileges. Similarly, we suppose that having write access implies having read access.

Malware on a computer For a computer, we will consider four interfaces:

- the USB interface, capturing for instance the keyboard, or a U2F USB key, with all possible types of access;
- the display, the computer screen, with only output interfaces;
- the TLS interface, capturing the network communications, but by always assuming that the attacker has the same level of control over inputs and outputs;
- the hard drive interface, capturing control of the storage of the computer, with all possible types of access.

We can succinctly describe a malware on a computer by giving for each interface the attacker's rights for both inputs and outputs of this interface. We use the notation $\mathcal{M}^{interf}_{in:acc1,out:acc2}$, where interf might be TLS, USB, hdd or dis, and $acc1$ and $acc2$ might be $\mathcal{RO}$ or $\mathcal{RW}$, to denote that the attacker has rights $acc1$ on the inputs, respectively rights $acc2$ on the outputs, of interface interf.

By convention, if we do not specify any access level, it means that the attacker has no access. A key-logger is for instance denoted with $\mathcal{M}^{USB}_{in:\mathcal{RO}}$. If the access level is the same both for the inputs and the outputs, as we always assume for TLS, we may write $\mathcal{M}^{TLS}_{io:\mathcal{RW}}$, thus capturing the fact that the attacker may have full control over the user browser, or that they might have exploited a TLS vulnerability.

Remark that we give a very high-level threat model for TLS, only considering read and write accesses. While this subsumes all possible capabilities, this does not reflect precisely the capabilities that an attacker may gain through XSS or CSRF attacks. However, such attacks tend to be linked to the actual implementation of the web server or of the browser, rather than being protocol specific. Furthermore, capturing such attacks requires a very fine grained model of the web infrastructure, such as the one presented by Fett et al. [FKS14]. Such a fine grained model would break the automation of our analysis, which was already at the limit of ProVerif's capabilities (minor changes to the model lead to non termination of ProVerif).

Malware on a phone For a mobile phone, the type of interface may depend on the protocols, with for instance SMS inputs or TLS inputs. To simplify, we will consider a phone to have only one input and one output interface. We thus only consider a generic device interface called dev, with all possible access levels. $\mathcal{M}^{dev}_{in:\mathcal{RO}}$ then corresponds for instance to the attacker having broken the SMS encryption, or to some malware on the phone listening to inputs.

3.3.2 Fingerprint Spoofing

Whenever a user browses the Internet, the user provides information about him or herself, called their *fingerprint*. Those elements will be very useful later on for additional checks in our protocols, and as we mentioned Google is adding this kind of details to their One-Tap protocol. However, in some cases the attacker might be able to obtain the same fingerprint as a given user. While some elements, such as the OS version, are rather easy to spoof, it is more complicated to spoof the IP address and inferred location. It is nevertheless possible if an attacker either completely controls the network the user connects on, or is connected to the same WiFi, or works in the same office.

3.3.3 Human Errors

The attacker may also exploit vulnerabilities that rely on the user not or wrongly performing some actions, or preferring to ignore security warnings. The assumption that users may not behave in the expected way seems reasonable given that most users are not trained in computer security, and their goal is generally to access a service rather than performing security related actions.

Phishing In our model, we capture that users may be victims of *phishing* attempts, i.e., willing to authenticate on a malicious website. For instance, an untrained, naive user may be willing to click on a link in an email which redirects to a fake web site. While a phishing attack through an e-mail may not fool a trained user, even a more experienced user may be victim to more sophisticated attacks, for instance if they connect to an attacker WiFi hotspot which asks to login to a website in order to obtain free Wii. Therefore, when we consider the *phishing threat scenario* we allow the attacker to choose with whom the user will initiate the protocol. We consider phishing as one of the simplest attacks to mount, and protocols should effectively protect users against it.

However, even though we consider that users might be victim of phishing, we suppose that they are careful enough to avoid it when performing the most sensitive operations: these operations include the registration of the U2F key, and logging for the first time on a computer they wish to trust later on. Indeed, if we were to allow phishing to be performed during those steps, no security guarantees could ever be achieved as the use of a second factor authentication requires a trusted setup.

No compare A protocol may submit to the user a fingerprint and expect the user to continue the protocol only if the fingerprint corresponds to their own. When given a fingerprint and a confirmation button, some users may confirm without reading the displayed information. Thus, when considering the *no compare* scenario, we assume that the user does not compare any value given to him and always answers yes.

3.3.4 Threat Scenarios Considered

In our analysis we consider all the possible combinations of the previously presented scenarios. This yields a fine-grained threat model that allows for a detailed comparison of the different protocols, and to identify the strengths and weaknesses of each protocol, by showing which threats are mitigated by which mechanisms.

By considering those possibilities, we capture many real life scenarios. For instance, when a user connects to a WiFi hotspot in a hotel or train station, the WiFi might be controlled by the

attacker, making the fingerprint spoofing and phishing scenarios realistic, because the attacker can have full control over the network, and thus use the provided IP address or redirect a user to a fake website.

If we try to connect on some untrusted computer, for instance the computer of a coworker, it may contain a rather basic malware, for instance a key-logger ($\mathcal{M}^{\mathsf{USB}}_{in:\mathcal{RO}}$). However, if we connect on a computer shared by many people at some place, for instance at a cybercafe, there could be a very strong malware controlling the display of the computer ($\mathcal{M}^{\mathsf{dis}}_{out:\mathcal{RW}}$) or controlling any TLS connection on this computer ($\mathcal{M}^{\mathsf{TLS}}_{io:\mathcal{RW}}$). Moreover, the network in this unknown place might also be compromised, and we may have some other scenarios combined with the malware, such as phishing (PH) or fingerprint spoofing (FS).

Our different scenarios provide different levels of granularity going from no attacker power at all to complete control over both the network and the platform. Our threat model abstracts away from how the attacker gained this power. Thus, the scenarios we consider will contain at some point all the possible attacks, without the need to specify how they may be performed. Note that we distinguish access to the RAM of the computer and access to the hard drive. For instance, a TLS session key will only be stored in RAM and a cookie will be stored on the hard drive. A side channel attack such as Meltdown [LSG+18] or Spectre [KGG+18] may allow the attacker to read the RAM of the user computer. In the protocols studied in this Part, all values stored in the RAM are received over one of the channels and not generated by the computer. Thus, in our examples the RAM read only access is equivalent to giving read-only access to all the interfaces of the computer ($\mathcal{M}^{\mathsf{USB}}_{io:\mathcal{RO}}$ $\mathcal{M}^{\mathsf{TLS}}_{io:\mathcal{RO}}$ $\mathcal{M}^{\mathsf{dis}}_{io:\mathcal{RO}}$ $\mathcal{M}^{\mathsf{hdd}}_{io:\mathcal{RO}}$). Another threat scenario is pharming, where the attacker can "lie" about the URL that is displayed to the user. This may happen either because of a malware that edits the *hosts* file (on a UNIX system), or by performing DNS ID Spoofing or DNS Cache Poisoning. All of these scenarios are simply captured as $\mathcal{M}^{\mathsf{TLS}}_{io:\mathcal{RW}}$.

3.4 The Formal Model

For our formal analysis, we model protocols in the applied pi-calculus presented in Section 2.1. As required by the modelling of TLS, We first extend the semantics with support for secret channels, and then provide a detailed threat model.

Technical Details

In the next chapter, we use the ProVerif tool, which has a slightly different syntax and semantics. The precise semantics used by ProVerif can be found in [Bla16]. We use our calculus for the sake of coherence with the multiple parts of the Thesis, and the gap between the semantics should not impact our analysis. The main syntactic difference, when comparing the example given here and the ProVerif files of the case study ([Mfa]), is the presence of types.

3.4.1 Extension of the Process Calculus with Secret Channels

To allow for a high level modelling of the behaviour of TLS, we extend the pi-calculus with secret channels. Rather than sampling channels in the fix set $\mathcal{C}$ of constants, we allow channel identifiers to be arbitrary terms in $\mathcal{T}(\Sigma, \mathcal{X}, \mathcal{N})$, where all variables must be bound in protocols. This implies that channels can be dynamically created. This extension is dedicated to the symbolic model, where we can thus assume that we have access to the deducibility relation $\vdash_E$ (Definition 2.4). We

$$
\begin{array}{ll}
\mathit{Server} \hat{=} & \mathit{Platform} \hat{=} \\
\quad \mathbf{in}(a, x); & \quad \mathbf{in}(kb, x_{m1}); \\
\quad \mathbf{if}\ x = \langle \mathit{login}, \mathit{pass} \rangle\ \mathbf{then} & \quad \mathbf{out}(a, x_{m1}); \\
\quad\quad \mathbf{out}(\mathit{sms}, \mathit{code}_i); & \quad \mathbf{in}(kb, x_{m2}); \\
\quad\quad \mathbf{in}(a, x_{\mathit{code}}); & \quad \mathbf{out}(a, x_{m2}). \\
\quad\quad \mathbf{if}\ x_{\mathit{code}} = \mathit{code}_i\ \mathbf{then} & \\
\quad\quad\quad \mathbf{event}\ \mathtt{Login}(\mathit{login}). & \\
 & \\
\mathit{User} \hat{=} & \mathit{Mobile} \hat{=} \\
\quad \mathbf{event}\ \mathtt{Initiate}(\mathit{login}); & \quad \mathbf{in}(\mathit{sms}, x_{\mathit{code}}); \\
\quad \mathbf{out}(kb, \langle \mathit{login}, \mathit{pass} \rangle); & \quad \mathbf{out}(\mathit{phone}, x_{\mathit{code}}). \\
\quad \mathbf{in}(\mathit{phone}, x_{\mathit{code}}); & \\
\quad \mathbf{out}(kb, x_{\mathit{code}}). &
\end{array}
$$

$$\|^i(\mathit{Server} \| \mathit{Platform} \| \mathit{Mobile} \| \mathit{User})$$

Figure 3.7: *Google 2-step* Toy Example

replace the rule Out of Figure 2.4 by the two following rules:

$$\textsc{Out}\ \frac{}{\varphi, (\mathbf{out}(t, s).P, \sigma)\ \rightarrow\ \varphi \uplus \{[\![s]\!]^\sigma\}, (P, \sigma)}\ \text{if}\ \varphi \vdash_E [\![t]\!]^\sigma$$

$$\textsc{Sync}\ \frac{}{\varphi, (\mathbf{in}(t_P, x).P, \sigma_P) \| (\mathbf{out}(t_Q, s).Q, \sigma_Q)\ \rightarrow\ \varphi, (P, \sigma'_P) \| (Q, \sigma_Q)}\ \text{if}\ \begin{cases} [\![t_P]\!]^{\sigma_P} =_E [\![t_Q]\!]^{\sigma_Q} \\ \varphi \not\vdash_E [\![t_P]\!]^{\sigma_P} \\ \sigma'_P = \sigma_P \uplus \{x \mapsto [\![s]\!]^{\sigma_Q}\} \end{cases}$$

The rule Out only applies if the attacker can deduce the identifier of the channel, once it has been interpreted w.r.t. to the local binding variables of the process. Sync allows an input and an output on the same channel to be reduced in a synchronous way, without leaking anything to the attacker. It can only be executed if the term of the channel cannot be deduced, and if the communications are on the same channel.

We provide an example of a process in Figure 3.7. a is a constant known to the attacker, modelling an insecure Internet communication. All other string identifiers are names: *sms* models the sms channel, *kb* the keyboard between the user and the platform and *phone* the screen of the mobile. A user process *User* wants to authenticate to some server *Server*. To do so, the user sends their *login* and password *pass* to their platform which are then forwarded to the server. The *Server* generates a fresh code_i for each session which sent to the user's *Mobile*. The code is then forwarded to the user, and back to the server through the platform.

Considering this example, we model the correspondence property (Definition 2.2) that any accepted login was actually initiated by the user

$$\mathtt{Login}(x) \Longrightarrow \mathtt{Initiate}(x)$$

This property is satisfied here, thanks to the *sms* channel which is private. In this case, the property is even injective.

3.4.2 Modelling TLS Communications

Most web protocols rely on TLS to ensure the secrecy of the data exchanged between a client and a server. In order to formally analyse online authentication protocols, we thus need to model TLS sessions and corresponding attacker capabilities. A possibility would of course be to precisely model the actual TLS protocol and use this model in our protocol analysis. This would however yield an extremely complex model, which would be difficult to analyse. A more detailed model of TLS would mostly be of interest for the analysis of TLS itself, rather than the protocol that make use of it. Therefore, for this Part, we opt to model TLS at a higher level of abstraction. In essence we model that TLS provides

- confidentiality of the communications between the client and the server, unless one of them has been compromised by the adversary;
- a session identifier that links all messages of a given session, avoiding mixing messages between different sessions.

To model this in the applied pi calculus, we use what is called private function symbols. Terms can be built using those symbols, but they are not available to the attacker for the deduction.

Technical Details 1

Private function symbols are built-ins of ProVerif. We may encode them formally in our model by defining the syntactic sugar $\texttt{TLS}(x, y) \mapsto \texttt{TLS}(x, y, n_f)$, where TLS a function symbol of arity 3 and n_f is a secret name, that is completely fresh for the protocols.

We then model TLS as follows:

- we define a private function `TLS(id,id)` where `id` is a user defined type of identities, and use the channel `TLS(c,s)` for communications between client c and server s;
- we define a *TLS manager* process that given as inputs two identities id_1 and id_2 outputs on a public channel the channel name $\texttt{TLS}(id_1, id_2)$, if either id_1 or id_2 are compromised;
- we generate a fresh name of type `sid` for each TLS connection and use it as a session identifier, concatenating it to each message, and checking equality of this identifier at each reception in a same session.

However, even if the communication is protected by TLS, we suppose that the adversary can block or delay communications. As communications over private channels are synchronous we rewrite each process of the form $\mathbf{out}(\texttt{TLS}(c, s), M).P$ into a process $\mathbf{out}(\texttt{TLS}(c, s), M)|P$. This ensures that the communications on TLS channels are indeed *asynchronous*. We provide the new elements of our previous toy example in Figure 3.8. We use pattern matching to bind the variable on inputs, e.g., $\mathbf{in}(c, (x, y))$ is a shortcut for testing if the input is a pair, and assigning the projection to the variables. If the input received does not follow the pattern, the thread goes into a failure state.

The TLS manager essentially allows the attacker to have a valid TLS session as long as the communication is not between the honest user and the server. This means that, even though we consider a single honest user, the attacker can perform all actions corresponding to sessions involving other users. Hence, in our model we consider a single honest user in parallel with an arbitrary number of corrupted users. As the corrupted user may behave honestly, considering a single honest user is not a limitation. Note however, that we assume that there are no interactions between the user's computer and phone and the equipment of other users.

$$
\begin{array}{ll}
\multicolumn{2}{l}{Platform \hat{=}} \\
\quad \mathbf{in}(kb, \langle x_{login}, x_{pass}, x_{idS} \rangle); & \\
\quad \mathbf{out}(\mathtt{TLS}(idP, x_{idS}), \langle x_{login}, x_{pass} \rangle; & \\
\| \quad \mathbf{in}(kb, x_{code}); & \\
\quad \mathbf{out}(\mathtt{TLS}(idP, x_{idS}), x_{code}). & \\
 & \\
\multicolumn{2}{l}{TLS_{manager} \hat{=}} \\
\quad \mathbf{in}(a, \langle x_{idc}, x_{ids} \rangle); & \\
\quad \mathbf{if}\ \mathtt{not}(x_{idc} = idP) \| \mathtt{not}(x_{ids} = idS)\ \mathbf{then} & \\
\quad\quad \mathbf{out}(a, \mathtt{TLS}(x_{idc}, x_{ids})). &
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{Server \hat{=}} \\
\quad \mathbf{in}(a, x_{id}); & \\
\quad \mathbf{in}(\mathtt{TLS}(x_{id}, idS), \langle x \rangle); & \\
\quad \mathbf{if}\ x = \langle login, pass \rangle\ \mathbf{then} & \\
\quad\quad \mathbf{out}(sms, code_i); & \\
\quad\quad \mathbf{in}(\mathtt{TLS}(x_{id}, idS), x_{code}); & \\
\quad\quad \mathbf{if}\ x_{code} = code_i\ \mathbf{then} & \\
\quad\quad\quad \mathbf{event}\ \mathtt{Login}(login). & \\
 & \\
\multicolumn{2}{l}{User \hat{=}} \\
\quad \mathbf{event}\ \mathtt{Initiate}(login); & \\
\quad \mathbf{out}(kb, \langle login, pass, idS \rangle); & \\
\quad \mathbf{in}(phone, x_{code}); & \\
\quad \mathbf{out}(kb, x_{code}). &
\end{array}
$$

$$\|^i (Server \| Platform \| TLS_{manager} \| Mobile \| User)$$

Figure 3.8: *Google 2-step* Toy Example with TLS

3.4.3 Modelling Threat Models

We will now present how we model the different scenarios discussed in Section 3.3 in the applied pi calculus.

Malware As discussed in Section 3.3.1, we view a system as a set of interfaces. By default, these interfaces are defined as private channels. Let a be a public channel (i.e., a constant). A malware providing read-only access to an interface ch is modelled by rewriting processes of the form $\mathbf{in}(ch, x).P$ into processes of the form $\mathbf{in}(ch, x).\mathbf{out}(a, x).P$, respectively $\mathbf{out}(ch, M).P$ into $\mathbf{out}(a, M).\mathbf{out}(ch, M).P$, depending on whether inputs or outputs are compromised. Read-write access is simply modelled by revealing the channel name ch, which gives full control over this channel to the adversary. We provide in Figure 3.9 an example where the input received on the keyboard channel kb is forwarded to the attacker. The modified part of the process is highlighted.

Fingerprint and spoofing When browsing, one may extract information about a user's location, computer, browser and OS version, etc. This fingerprint may be used as an additional factor for identification, and can also be transmitted to a user for verification of its accuracy. We model this fingerprint by adding a function $\mathtt{fpr}(id)$ which takes an identity and returns its corresponding fingerprint. Given that all network communications are performed over a TLS channel $\mathtt{TLS}(c, s)$ the server s can simply extract the fingerprint $\mathtt{fpr}(c)$. However, in some cases we want to give the attacker the possibility to spoof the fingerprint, e.g., if the attacker controls the user's local network. In these cases we declare an additional function $\mathtt{spoof}_{fpr}(x)$ and the equation

$$\mathtt{fpr}(\mathtt{spoof}_{fpr}(\mathtt{fpr}(c))) = \mathtt{fpr}(c)$$

which provides the attacker with an identity whose fingerprint is identical to $\mathtt{fpr}(c)$, and allows the attacker to initiate a communication on a channel $\mathtt{TLS}(\mathtt{spoof}_{fpr}(\mathtt{fpr}(c)), s)$.

$Platform \hat{=}$
in$(kb, \langle x_{login}, x_{pass}, x_{idS}\rangle)$;
out$(a, \langle x_{login}, x_{pass}, x_{idS}\rangle)$;
out$(\texttt{TLS}(idP, ids), \langle x_{login}, x_{pass}\rangle)$;
$\cdots$

Figure 3.9: Key-logger

$User \hat{=}$
event $\texttt{Initiate}(login)$;
out$(kb, \langle login, pass, idS\rangle)$;
in$(phone, \langle x_{code}, x_{fingerprint}\rangle)$;
if $x_{fingerprint} = \texttt{fpr}(idP)$ **then**
out(a, x_{code}).

Figure 3.10: Fingerprint

$User \hat{=}$
event $\texttt{Initiate}(login)$;
in(a, x_{id});
if $x_{id} = idS$ **then**
out$(kb, \langle login, pass, x_{id}\rangle)$;
in$(phone, \langle x_{code}, x_{fingerprint}\rangle)$;
if $x_{fingerprint} = \texttt{fpr}(idP)$ **then**
out(kb, x_{code}).

Figure 3.11: Phishing

We show in Figure 3.10 an example where the *User* also receives from their phone the *fingerprint* of the platform seen by the server, and checks that the fingerprint does match the fingerprint of their platform.

Human errors - No compare Our model contains dedicated processes that represent the expected actions of a human, e.g., initiating a login by typing on the keyboard, or copying a received code through the display interface of their computer or phone. A user is also assumed to perform checks, such as verifying the correctness of a fingerprint or comparing two random values, one displayed on the computer and one on the phone. In the *No Compare* scenario we suppose that a human does not perform these checks and simply remove them. The corresponding process is obtained from Figure 3.10, by simply removing the highlighted conditional "**if** $x_{fingerprint} = \texttt{fpr}(idP)$ **then** ".

Human errors - Phishing In our model of TLS we simply represent a URL by the server identity idS, provided by the human user, as it was shown in Figure 3.8. This initiates a communication between the user's computer, with identifier idC, and the server over the channel $\texttt{TLS}(idC, idS)$. This models that the server URL is provided by the user and may be the one of a malicious server, which their machine is then connecting to. We let the adversary provide the server identity x_{id} to the user in order to model a basic *phishing* mechanism. We distinguish two cases: a trained user will check that $x_{id} = idS$, where idS is the correct server, while an untrained user will omit this check and connect to the malicious server. The updated *User* process is provided in Figure 3.11, where we highlight the line to be removed under phishing.

4 An Extensive Analysis

Les rêves, ça ne se compare pas.

(Le roi Arthur - Kaamelott)

4.1 Introduction

We want to have a detailed and modular threat models, so that we may consider attackers as powerful as possible. We defined such a model in the previous Chapter, yet, once a detailed threat model has been defined, performing the corresponding analysis on concrete examples is challenging. Given a protocol, one needs to explore all combinations of attacker capabilities, which can yield thousands of distinct analyses to perform. We perform such an analysis on the MFA protocols presented in the previous chapter, thanks to the highly automated tool Proverif [BCA+10].

The analysis of multiple protocols designed to provide the same guarantees allows to obtain a high level understanding of the weaknesses and strengths of each protocol. Also, the precise security guarantee provided by a given mechanism in a more complex security protocol can be clearly identified. Such results can be used to help system designers to choose the protocol that best suits their needs, given their specifications.

We perform the extensive analysis by ranging over all possible combinations of attacker capabilities (but without exploring the scenarios already subsumed), and exploring multiple versions of a protocol. However, remark that we only consider authentication properties. To perform a truly extensive analysis, one would need to consider all the possibly interesting security properties provided by a given protocol. This can be of importance in contexts where we expect a protocol to provide multiple properties. E-voting protocols can for instance be considered, where many properties can be expected. In the context of authentication protocols, one may want to consider privacy properties such as unlinkability. We briefly address this subject on the *FIDO's U2F* protocol, where the specific setting only yield one interesting scenario.

Chapter Summary

We use the Proverif tool to systematically and automatically analyse several versions of *Google 2-step* and U2F in an extensive way, considering all possible threat combinations and analysing over 6000 (incomparable) scenarios. The resulting protocols comparison highlights strengths and weaknesses of the different mechanisms, and allows us to propose some simple variants, adding actions to the displayed information or linking the URL to the payload, which improves security. We also study unlinkability of *FIDO's U2F* protocol. Using our formalism, we rediscover two previously reported attacks. While these attacks have been considered non critical we argue that both attacks can be combined into a more dangerous one. Finally, we validate our models and findings by demonstrating the feasibility of several attacks, in laboratory conditions. We conclude with a final comparison between *FIDO's U2F* and *Google 2-step* approaches.

4.1.1 Our Contributions

We analyse several variants of the *Google 2-step* and *FIDO's U2F* protocols in a detailed threat model. The analysis is completely automated, using scripts to generate systematically all combinations of threat scenarios for each of the protocols and using the ProVerif tool for automated protocol analysis. Even though we eliminate threat scenarios as soon as results are implied by weaker scenarios, the analysis required over 6 000 calls to ProVerif, yet finishes in only a few minutes. Our analysis results in a detailed comparison of the protocols which highlights their respective weaknesses and strengths. It allows us to suggest several small modifications of the existing protocols which are easy to implement, yet improve their security in several threat scenarios. In particular, the existing mechanisms do not authenticate the *action* that is performed, e.g., a simple login may be substituted by a login enabling the *"trust this computer"* mechanism, or a password reset. Adding some additional information to the display may thwart such attacks in many of our threat scenarios. We propose the variant G2DT^{DIS} of the previously introduced G2V^{FPR} protocol, where such information is displayed. We also propose a new variant of *Google 2-step* building on ideas from *FIDO's U2F* protocol.

To validate our model and analysis we verify that the weaknesses we found can indeed be put into practice. We report on our experiments with the google mail server and a FIDO USB token, implementing *FIDO's U2F* protocol. Even though our experiments are performed in a laboratory environment they confirm the relevance of our models and analyses.

In addition to authentication, we also study *unlinkability*. The *FIDO's U2F* specification claims that it should not be possible to link two accounts that use the same second factor token. Modelling unlinkability in the applied pi calculus, we are able to find two attacks. Both attacks we found appeared to be known. However, we argue that they may be combined into a more relevant one.

Limitations

Our analysis is performed in the symbolic model, and thus lacks the precision of the computational model. Moreover, the analysis on unlinkability is performed for a single threat model.

Essentially, we are limited by the current state of the art of automated analysis both in the symbolic and computational models. Notably, while we may want to perform analysis for other properties or other protocols (involving more complex primitives), we believe that we may have reached the limits of ProVerif. Indeed, small changes to the modelling for the authentication study causes non termination of the analysis.

4.1.2 Related Work

Regarding the practical extensive analysis, many studies were performed by generating multiple protocols or multiple scenarios systematically. We only mention three of the most recent work following this idea, based on distinct provers. [CGT18] studies with ProVerif three distinct security properties of a single e-voting protocol, given a fixed threat model, but with a protocol parameterized by an integer n and instantiating the parameter with multiple values. Each scenario was produced from a single file in a systematic way. Still in the e-voting protocol area, [CDD+17] provides machine checked proofs of privacy related properties using EasyCrypt for several hundred variants of an e-voting protocol. [GHS+20] studies the Noise framework by analysing many different protocols of the framework and systematically deriving the maximal threat model supported by each of the protocol using the Tamarin prover.

Concerning MFA, other attempts to automatically analyse MFA protocols were made, including

for instance the analysis of *FIDO's U2F* [PRW17], the *Yubikey One Time Password* [KK16; KS13], the analysis of MFA combined with Single sign-on with SATMC [SCR+18] and the *Secure Call Authorization* protocols [ACZ13]. However, those analyses do not study resistance to malware, nor do they capture precisely TLS channel behaviour or fingerprints.

? Future Work

As a direction for future work, it would be interesting to perform an in depth analysis of U2F [FID18] and FIDO2 [BLVGB+17], also known as WebAuthn, standards, using our fully mechanized approach.

As another direction, we consider the use of *enclaves* in trusted execution environments: such environments could provide execution certification and a way to enable secure login on a completely untrusted computer, if the computer is equipped with a trusted module. One could then use a phone as a U2F token assuming that we also have an efficient way to establish a channel between the computer and the phone in order to pass the payload. The U2F keys could be stored on the phone, and the next natural step would be to merge G2DT$^{\text{DIS}}$ and U2F by performing a U2F on the phone in parallel of the G2DT$^{\text{DIS}}$. The user would only see the G2DT$^{\text{DIS}}$ part, which would even be simplified without the double tap, because thanks to the channel between the phone and the computer, there would not be any need to ask the user to select the correct random. G2DT$^{\text{DIS}}$ combined with for instance the storage of the keys using a trusted execution environment, such as TrustZone would then palliate the issue of keys being revealed due to malware on the phone.

4.2 Analysis and Comparison

Section Summary

We use the formal framework presented in Chapter 3 to analyse several MFA protocols, focusing on authentication properties. The analysis is completely automated using the ProVerif tool, along with a script which generate all possible combinations of attacker capabilities from a single modelling file. All scripts and source files used for these analyses are available at [Mfa]. The results are summarized in tables (systematically produced from the analysis), allowing to compare the respective guarantees provided by distinct protocols, depending on the considered scenario.

4.2.1 Properties and Methodology

Properties We focus on authentication properties and consider that a user may perform 3 different actions:

- an *untrusted login*: the user performs a login on an untrusted computer, i.e., without selecting the *"trust this computer"* option, using second-factor authentication;
- a *trusted login*: the user performs an initial login on a trusted computer, and selects the *"trust this computer"* option, using second-factor authentication;
- a *cookie login*: the user performs a login on previously trusted computer, using their password but no second factor, and identifying through a cookie and fingerprint.

For each of these actions we check that whenever a login happens, the corresponding login was requested by the user. We therefore define three pairs of events

$$(init_x(\mathtt{id}), accept_x(\mathtt{id})) \qquad x \in \{u, t, c\}$$

The $init_x(\mathtt{id})$ events are added to the process modelling the human user, in order to capture the user's intention to perform the login action. The $accept_x(\mathtt{id})$ events are added to the server process. The three properties are then modelled as three injective correspondence properties:

$$accept_x(id) \Longrightarrow_{inj} init_x(id) \qquad x \in \{u, t, c\}$$

When the three properties hold, we have that every login of some kind accepted by the server for a given computer matches exactly one login of the same kind initiated by the user on the same computer.

Methodology For every protocol, we model the three different types of login, and then check using ProVerif whether each security property holds for all possible (combinations of) threat scenarios presented in Section 3.3. As we consider trusted and untrusted login, we provide the user with two platforms: a trusted platform on which the user will try to perform trusted logins, and an untrusted platform for untrusted logins. We will thus extend the notation for malwares presented in 3.3.1 by prefixing the interface with t if the interface belongs to the trusted computer, and u if it belongs to the untrusted computer. For instance, $\mathcal{M}^{\mathtt{u}-usb}_{in:\mathcal{RO}}$ corresponds to a key-logger on the untrusted computer. A scenario is described by a list of considered threats that may contain

- phishing (PH);
- fingerprint spoofing (FS);
- no comparisons by the user (NC);
- the malwares that may be present on the trusted and untrusted platform.

For instance, "PH FS $\mathcal{M}^{\mathtt{t}-usb}_{io:\mathcal{RW}}$" denotes the scenario where the attacker can perform phishing, fingerprint spoofing, and has read-write access to the inputs and outputs of USB devices of the trusted computer. "NC $\mathcal{M}^{\mathtt{u}-tls}_{io:\mathcal{RW}}$ $\mathcal{M}^{\mathtt{u}-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{\mathtt{u}-dis}_{io:\mathcal{RW}}$" models a human that does not perform comparisons and an attacker that has read-write access to the inputs and outputs of the TLS, USB and display interfaces of the untrusted device.

We use a script to generate the files corresponding to all scenarios for each protocol and launch the ProVerif tool on the generated files. In total we generated 6 172 scenarios that are analysed by ProVerif in 8 minutes on a computing server with twelve Intel(R) Xeon(R) CPU X5650 @ 2.67GHz and 50Go of RAM. We note that we do not generate threat scenarios whenever properties are already falsified for a weaker attacker (considering less threats or weaker malware). The script generates automatically the result tables, displaying only results for minimal threat scenarios that provide attacks, and maximal threat scenarios for which properties are guaranteed. In the following Sections we present partial tables with results for particular protocols. Full results for all protocols are given in Tables A.1 and A.2 in Appendix.

The result tables use the following notations:

- results are displayed as a triple $u\,t\,c$ where u, t, c are each ✗ (violated) or ✓ (satisfied) for the given threat scenario; each letter in the set $\{u, t, c\}$ gives the status of the authentication property for *u*ntrusted login, *t*rusted login and *c*ookie login respectively;
- ✖ and ✔ are shortcuts for ✗✗✗ and ✓✓✓;
- signs are greyed when they are implied by other results, i.e., the attack existed for a weaker threat model, or the property is satisfied for a stronger adversary;
- we sometimes use blue, circled symbols to emphasize differences when comparing protocols.

Even if ProVerif can sometimes return false attacks, we remark that any ✗ corresponds to an actual attack where ProVerif was able to reconstruct the attack trace.

Threat Scenarios		G2V	G2OT	G2OT^{FPR}
		✔	✖	✔
PH		✖	✖	✔
NC		✔	✖	✖
FS		✔✔✔	✖	✖
	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✔✔✔	✖	✔
	$\mathcal{M}^{dev}_{in:\mathcal{RO}}$	✖	✖	✔
	$\mathcal{M}^{t-dis}_{io:\mathcal{RO}}$	✔	✖	✔
	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✔
	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✖	✔
	$\mathcal{M}^{dev}_{in:\mathcal{RW}}$	✖	✖	✖
	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✗✔✗
FS	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✔✔✗	✖	✖
	$\mathcal{M}^{u-hdd}_{in:\mathcal{RO}}$	✔	✖	✔
	$\mathcal{M}^{u-dis}_{io:\mathcal{RO}}$	✔	✖	✔
	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✖	✔
	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✖	✖	✔
	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✖	✔✗✗
	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✖	✖	✔✗✔

Table 4.1: Analysis of the Basic *Google 2-step* Protocols

Threat Scenarios		G2DT^{FPR}
NC		✔
FS		✔
NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✔
NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✔
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✔
NC	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✔
NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✔
FS	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✔
FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✔

Table 4.2: Analysis of the *Google 2-step* Double-Tap

4.2.2 *Google 2-step*: Verification Code and One-Tap

In this Section we report on the analysis of the currently available *Google 2-step* protocols: the verification code (G2V, described in Section 3.2.1), the One-Tap (G2OT, described in Section 3.2.1) with and without fingerprint, and the Double-Tap (G2DT^{FPR}, described in Section 3.2.1). The results are summarized in Tables 4.1 and 4.2.

g2V In the G2V protocol the user must copy a code received on their phone to their computer to validate the login. We first show that G2V is indeed secure when only the password of the user was revealed to the attacker: as long as the attacker cannot obtain the code, the protocol remains secure. If the attacker obtains the code, either using a key-logger ($\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$), or by reading the SMS interface ($\mathcal{M}^{dev}_{in:\mathcal{RO}}$), or any other read access to an interface on which the code is transmitted, the attacker can use this code to validate their own session. Looking at Table 4.1, it may seem surprising that a malware on a trusted platform may compromise an untrusted login. This is due to the fact that a code of a trusted session may be used to validate an untrusted session and vice-versa. Moreover, if the attacker can access the hard disk drive ($\mathcal{M}^{t-hdd}_{in:ro}$), they may steal the cookie that allows to login without a second factor, and then perform a login if they can also spoof the platform fingerprint (FS).

We have tested on the Google website that a code generated for a login request can indeed be used (once) for any other login, demonstrating that such attacks are indeed feasible. Interestingly, this also shows that in the actual implementation, the verification code is not linked to the TLS session. Not linking codes to sessions is actually useful as it allows to print in advance a set of codes, e.g., if no SMS access is available. Moreover, we note that linking the code to a session does not actually improve security in our model, as the code of the attacker session will also be sent to the user's phone and could then be recovered. In practice, if the code is linked, an attack can be produced only if the attacker's code is received first, i.e., if the attacker can login just before or after the user.

We remark that the results for G2V are also valid for another protocol, *Google Authenticator*. On this protocol the phone and the server share a secret key, and use it to derive a one time password (OTP) from the current time. In all the scenarios where the SMS channel is secure, G2V can be

seen as a modelling of *Google Authenticator* where the OTP is a random value "magically" shared by the phone and the server.

g2OT In the G2OT protocol a user simply confirms the login by pressing a yes/no button on their phone. We first consider the version that does not display the fingerprint, and which is still in use. Our automated analysis reports a vulnerability even if only the password has been stolen. In this protocol, the client is informed when a second, concurrent login is requested and the client aborts. However, if the attacker can block, or delay network messages, a race condition can be exploited to have the client tap yes and confirm the attacker's login. We have been able to reproduce this attack in practice and describe it in more detail in Section 4.3. While the attack is in our most basic threat model, it nevertheless requires that the attacker can detect a login attempt from the user, and can block network messages (as supposed in the Dolev-Yao model).

g2OT^{fpr} We provide in the third column of Table 4.1 the analysis of G2OT^{FPR}. To highlight the benefits of the fingerprint, we color additionally satisfied properties in blue. In many read only scenarios ($\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$, $\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$,$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$, $\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$), and even in case of a phishing attempt, the user sees the attacker's fingerprint on there phone and does not confirm. However, if the user does not check the values (NC) or if the attacker can spoof the fingerprint (FS), G2OT^{FPR} simply degrades to G2OT and becomes insecure. Some attacks may be performed on the cookie login, for instance for scenarios $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$ or $\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$, as the attacker may initiate a login from the user's computer without the user having any knowledge of it, and then use it as a kind of proxy.

Because of the verification code, in scenarios FS or NC, G2V provides better guarantees than G2OT^{FPR}. It is however interesting to note that G2OT^{FPR} resists to read only access on the device as there is no code to be leaked to the attacker. One may argue that an SMS channel provides less confidentiality than a TLS channel, i.e., the read-access on the SMS channel may be easier to obtain in practice. Indeed, SMS communications between the cellphone and the relay can be made with weaker encryption (A5/0 and A5/2) than TLS, and the SMS message will anyway be sent over TLS between the relay and the provider's servers. While this argument is in favour of G2OT^{FPR}, one may also argue that G2V has better resistance to user inattention, as a user needs to actively copy a code.

g2DT^{fpr} To palliate the weakness of G2OT compared to G2V, Google proposes G2DT^{FPR} where a comparison through a second tap is required. The additional security provided by the second tap is displayed in Table 4.2, where we highlight in blue the differences between G2OT^{FPR} and G2DT^{FPR}. The attacker must be able to have their code displayed and selected on the user's device in order to successfully login. Therefore, FS or NC scenarios with some additional read only access, are secure. Interestingly, in the NC scenario, we are now as secure as G2V, while having greater usability. We note that we are still not secure in the PH FS scenario. This means that an attacker controlling the user's network or some WiFi hotspot could mount an attack against G2DT^{FPR}.

4.2.3 Additional Display

In this Section, we propose and analyse small modifications of the previously presented protocols. Given the benefits discussed in Section 3.2.1, we first add a fingerprint to G2V.

In *Google 2-step* some attacks occur because the attacker is able to replace a trusted login by an untrusted one, e.g., under $\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$. If this happens, the attacker can obtain a session cookie for their own computer and perform additional undetected logins later on. A user might expect that

Threat Scenarios				G2V^{FPR}	G2V^{DIS}	G2OT^{DIS}	G2DT^{DIS}
PH				✔	✔	✔	✔
PH	FS			✖	✗✓✓	✖	✗✓✓
PH	FS	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$		✖	✗✓✗	✖	✗✓✗
PH	FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$		✖	✖	✖	✗✓-
PH	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$		✖	✖	✖	✗✓✓
PH	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$		✖	✗✓✓	✖	✖
PH	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✖	✖	✗✓✗
PH	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✗✓✗	✖	✖
		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$		✔	✔	✓✓✓	✔
		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$		✔	✔	✔	✔
		$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$		✗✓✗	✓✓✗	✓✓✗	✓✓✗
	FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$		✖	✓✗✗	✖	✓✓-
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$		✖	✓✗✗	✖	✓✓✓
	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$		✓✓✓	✓✓✓	✖	✓✗✗
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✓✗✗	✖	✓✓✗
	FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$		✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$		✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✓✗	✓✓✗	✖	✓✗✗
	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✓✗✗	✖	✓✗✗
		$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$		✔	✔	✓✓✓	✔
		$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$		✔	✔	✓✓✓	✔
		$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$		✓✗✗	✔	✓✓✓	✓✓✓
		$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$		✓✗✓	✔	✓✓✓	✓✓✓
	FS	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$		✖	✗✓✓	✖	✔
	FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$		✖	✗✓✓	✖	✔
	FS	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$		✔	✔	✖	✗✓✓
	FS	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$		✖	✗✓✓	✖	✗✓✓
	FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$		✖	✗✓✓	✖	✗✓✓
	FS	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✗✓✓	✖	✗✓✓
	FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✖	✗✓✓	✖	✗✓✓

Table 4.3: *Google 2-step* Protocols with Additional Display

by using a second factor, they should be able to securely login once on an untrusted computer and be assured that no additional login will be possible.

We now study a variant of each of the protocols where the user's action (trusted or untrusted login) is added to the display. This addition may create some harmless "attacks" where the attacker replaces a trusted login with an untrusted login. However, such attacks indicate that an attacker may change the type of action, such as password reset, or disabling second-factor authentication.

We call G2V^{FPR} the protocol version that additionally displays the fingerprint, and G2V^{DIS}, G2OT^{DIS} and G2DT^{DIS} the versions that additionally display the action, and provide in Table 4.3 the results of our analysis. To highlight the benefits of our modifications, we color additionally satisfied properties in blue, when considering G2V and G2V^{FPR}, G2V^{FPR} and G2V^{DIS}, G2OT^{FPR} and G2OT^{DIS} and G2DT^{FPR} and G2DT^{DIS}.

It appears that adding the action - and the fingerprint in the G2V case - performs as expected: the protocols become secure in all the scenarios where the only possible attack was a mixing of actions.

Threat Scenarios				G2V^{DIS}	G2DT^{DIS}
PH	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✗✓-
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✗✓✓
PH	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✗✓✓	✖
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✗✓✗
PH	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✗✓✗	✖
			$\mathcal{M}^{dev}_{in:\mathcal{RO}}$	✖	✔
		NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✔
		NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✔
		NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✔	✖
	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✗✗	✓✓-
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✓✗✗	✓✓✓
	FS	NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓-
	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✓✓✓	✓✗✗
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✗✗	✓✓✗
	FS	NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✓✓✗
	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✓✗	✓✗✗
	FS	NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✓✗	✖
		NC	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✔
		NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✖	✔
		NC	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✔	✖
	FS		$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✗✓✓	✓✓✓
	FS		$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✗✓✓	✓✓✓
	FS		$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✓✓✓	✗✓✓

Table 4.4: Comparison of *Google 2-step* with Code or Tap

Threat Scenarios		U2F
		✔
PH		✔
	FS	✓✓✓
	$\mathcal{M}^{dev}_{in:\mathcal{RO}}$	- - -
	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✓✓
	$\mathcal{M}^{t-dis}_{io:\mathcal{RO}}$	✔
	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✓
	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✔
	$\mathcal{M}^{dev}_{io:\mathcal{RO}}$	✖
	$\mathcal{M}^{dev}_{in:\mathcal{RW}}$	✖
	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖
	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖
	FS $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✓✗
	FS $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗
	$\mathcal{M}^{u-hdd}_{in:\mathcal{RO}}$	✔
	$\mathcal{M}^{u-dis}_{io:\mathcal{RO}}$	✔
	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✔
	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✓✓✓
	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖
	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✓✗✓
	$\mathcal{M}^{u-usb}_{io:\mathcal{RW}}$	✗✗✗

Table 4.5: U2F Results

4.2.4 Conclusion Regarding *Google 2-step*

Currently, Google proposes G2V, G2OT, G2OT^{FPR} and G2DT^{FPR}. Adding the type of login being performed (trusted or untrusted) to the display would provide additional security guarantees.

Among the studied mechanisms, G2V^{DIS} and G2DT^{DIS} provide the best security guarantees in our model, having each advantages and disadvantages. In Table 4.4, we provide a comparison between these two mechanisms. We observe that G2V^{DIS} performs better than G2DT^{DIS} only in scenarios where we have $\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$, which may be considered as a powerful malware.

G2DT^{DIS} provides better guarantees in many simpler threat scenarios, with for instance read-only access to the phone. As the code is sent back to the server from the phone rather than the computer, this mechanism is more resilient to malware on the computer. Moreover, the code is sent through a TLS channel rather than via SMS, which may arguably provide better security.

Finally, even though *Google 2-step* may significantly improve security, phishing attacks combined with fingerprint spoofing are difficult to prevent. This seems to be inherent to the kind of protocol, where the security is only enforced through the 2nd factor. As we will see in the next section the FIDO U2F protocol may provide better guarantees for these threat scenarios.

4.2.5 FIDO U2F

FIDO's U2F adds cryptographic capabilities to the mechanism through its registration mechanism. As explained previously, the URL of the server the user is trying to authenticate to is included in

the query made to the FIDO USB token, and also in the signature returned by the token. The server will then only grant login if the signature contains their own URL.

We present the results of our formal analysis in Table 4.5. U2F is secure under many threat scenarios, including some that combine phishing and fingerprint spoofing. However, an attack is found when the computer runs malware that controls the USB interface of the trusted computer ($\mathcal{M}^{\mathsf{t}-usb}_{\mathsf{io}:\mathcal{RW}}$). Indeed, the malware can then communicate with the U2F token, and thus send a request generated for an attacker session. Also, if the attacker can control the TLS interface ($\mathcal{M}^{\mathsf{t}-tls}_{\mathsf{io}:\mathcal{RW}}$ or $\mathcal{M}^{\mathsf{u}-tls}_{\mathsf{io}:\mathcal{RW}}$), they may change the intended action and replace an untrusted login with a trusted one. As a consequence, a login on an untrusted computer with U2F may enable future attacker logins on this computer. This contradicts claims that Yubikeys (an implementation of U2F token) guarantee protection against "phishing, session hijacking, man-in-the-middle, and malware attacks." While the claim indeed holds for the first threats, malware attacks are still possible. Moreover, one might expect an external hardware token to allow users to securely log on an untrusted computer. However, this enables an attacker to submit their own request to the user's token. Even though a user has to press the token button to accept each request, as noted previously, a malware controlling the TLS connection will allow several attacker logins for one user press due to the *"trust this computer"* mechanism.

U2F may lead to another problem that is out of the scope of our analysis: a Yubikey does not have any way to provide feedback for a successful press. When the computer submits two requests in a row to the token and the user just presses once, the user may believe that the press failed, and press once more. This is reminiscent of the problem identified during the analysis of the One-Tap mechanism: success and failure of the second factor should not be silent.

To summarize, one might expect U2F to protect against malware, as it is based on a secure hardware token providing cryptographic capabilities. Thus, even if U2F does provide a better security than most existing solutions, it does not uphold this promise completely. However, the U2F mechanism providing protection against phishing is very interesting. What appears to be lacking from U2F is some feedback capabilities, i.e., a screen, to notify failures, successes, and maybe information such as the fingerprint of the computer.

4.2.6 Token Binding

We previously studied the security of the protocols combined with the *"trust this computer"* mechanism where a cookie is used to authenticate a computer on the long term. We provide in Table 4.6 the results of the formal analysis of TokenBinding combined with U2F and G2DT$^{\text{DIS}}$, and highlight in blue the security gained with respect to the classical cookie version. It provides protection against a read only access to the TLS interface, because it is not any more sufficient to steal the cookie. We do not gain protection against control of the computer memory, as the secret key is stored the same way as the cookie. The attacker needs to be able to access the private key of the user which was generated on their platform but never sent over the network.

4.2.7 A g2DT$^{\text{dis}}$ Extension : g2DT$^{\text{ext}}$

Core idea We propose an extension of G2DT$^{\text{DIS}}$ based on ideas from U2F. Our goal is to provide a protocol which will have the same user experience as G2DT$^{\text{DIS}}$, but will provide a stronger protection against phishing by using the second factor to confirm the origin and the TLS session id.

To protect against phishing, the URL seen by the user must be authenticated. This requires

Threat Scenarios				U2F	U2F$_{\text{TB}}$	G2DT$^{\text{DIS}}$	G2DT$^{\text{DIS}}_{\text{TB}}$
PH	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✗✓-	✗✓✓
PH	FS	NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✖	✖
PH	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✖	✖
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✖	✖
	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✓✓-	✓✓✓
	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✓✗✗	✓✗✗
	FS	NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✖	✖
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✓✗✗	✓✗✗
	FS	NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✖	✖

Table 4.6: Results for the TokenBinding Extension

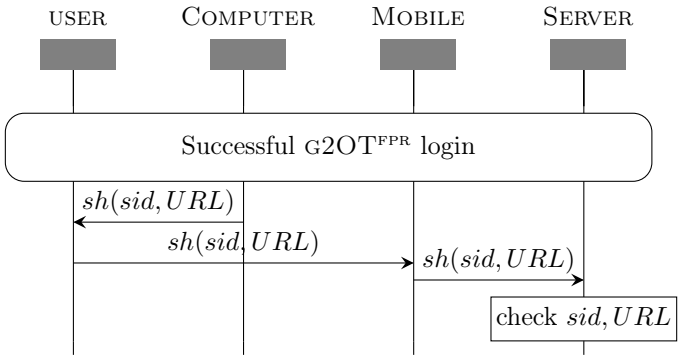


Figure 4.1: G2DT$^{\text{EXT}}$ Outline

an external intervention. For instance, in U2F the browser extracts the URL and sends it to the token. With a phone serving as a second factor, we may mimic this behavior by having the browser transmit the URL to the phone through some secure channel. The phone can then transmit the URL to the server on an independent channel, allowing the server to check that the URL seen by the user corresponds to their own URL. This however requires an efficient way to transmit data from the computer to the phone, ideally without any particular setup. NFC like technologies may provide a promising means for such a channel. Without this channel the selection mechanism of G2DT$^{\text{DIS}}$ may be used: to verify that a user has seen some data on their computer the phone displays the data among other random data, and asks the user to select the correct one.

Of course, an efficient and easy to deploy channel would be preferable to the selection mechanism. Yet, the selection mechanism of G2DT$^{\text{DIS}}$ allows for a protocol with the same user experience, rather easy to deploy, but with greater security.

Extension description The extension is similar to G2DT$^{\text{DIS}}$, except that the server does not send a freshly generated digit to the computer. The user's browser extracts the URL and the TLS session identifier and produces a short hash of those values that is displayed in a pop-up outside the web page. The server computes the same hash and sends it to the phone. The phone displays the hash among two other random values. If the user selects the correct value, the phone confirms the login to the server.

Intuitively, instead of using a signature to transmit securely the URL and the TLS session identifier, the protocol relies on a confirmation on the user's phone. The outline of the protocol is displayed in Figure 4.1.

Currently, G2DT$^{\text{DIS}}$ uses only a 2-digit integer. Hence, an attacker has probability 1/100 to guess the integer, which is much higher than usually accepted. If in G2DT$^{\text{EXT}}$ we were to use 2 digit hash

Threat Scenarios				G2DT^{DIS}	G2DT^{EXT}
PH		NC		✖	✔
PH	FS			✗✓✓	✓✓✓
PH	FS		$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✗✓✗	✓✓-
PH	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✗✓-	✓✓✗
PH	FS	NC	$\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✓✓-
PH	FS	NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓✗
PH	FS		$\mathcal{M}^{t-hdd}_{in:\mathcal{RW}}$	✗✓✗	✓✓✗
PH	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✓✗✗
PH	FS	NC	$\mathcal{M}^{t-hdd}_{in:\mathcal{RW}}$	✖	✓✓✗
PH	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✓✓✗
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✖	✔
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✓✓✗
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓✗
		NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✗✗✗	✓✓✗
		NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✖	✔
	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✓✗✗	✓✓✗
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✓✗✗	✓✓✓
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✗✗	✓✓✗
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✗✗	✓✓✗
	FS	NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✓✓✗
	FS	NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓✗
		NC	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✔
		NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✖	✓✗✓
		NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✓✗✗
	FS		$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✗✓✓	✓✓✓
	FS		$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✗✓✓	✓✓✓

Table 4.7: Comparison between G2DT^{DIS} and G2DT^{EXT}

Threat Scenarios			U2F	G2DT^{EXT}
		$\mathcal{M}^{dev}_{in:\mathcal{RO}}$	- - -	✔
		$\mathcal{M}^{dev}_{io:\mathcal{RO}}$	✖	✔
		$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✓✓✗
	NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✔	✖
		$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖	✔
FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✓✓✓	✓✗✗
FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✓✗	✓✗✗
FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✗✗
FS	NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✓✓✗	✖
FS	NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✖
FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✓✗✗
FS		$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-hdd}_{in:\mathcal{RO}}$	✖	✓✓✗
FS		$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓✗
FS		$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✓✗✗
		$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✔
	NC	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✔	✖
		$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✓✗✓	✓✓✓
	NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✓✗✓	✖
		$\mathcal{M}^{u-usb}_{io:\mathcal{RW}}$	✗✗✗	✔
	NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✓✗✗
	NC	$\mathcal{M}^{u-usb}_{io:\mathcal{RW}}$	✖	✓✗✓
FS		$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✓✓✓	✗✓✓
FS		$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✗✓✓
FS		$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✓✗✓	✗✓✓
FS		$\mathcal{M}^{u-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✖	✗✓✓

Table 4.8: Comparison between U2F and G2DT^{EXT}

values, an attacker could easily find collisions. To maintain usability and improve the security by transmitting more information, it might be worth exploring different mechanisms, such as using images or visual hashes. The only conditions are that the domain should be large, and a human should be able to instantly pick the correct value out of the three proposals.

4.2.8 g2DT^{ext} Analysis

We provide in Table 4.7 the advantages of G2DT^{EXT} when compared to G2DT^{DIS}, where we highlight the newly secure cases in blue. We ensure higher security guarantees in some common scenarios such as phishing combined with a distracted user who does not compare values. This means that G2DT^{EXT} can be used to effectively protect untrained people against phishing. Moreover, it is also secure in the case of phishing and fingerprint spoofing. Hence, the protocol provides secure login even when connecting on an untrusted network. The comparison between U2F and G2DT^{EXT} is displayed in Table 4.8, where we highlight differences in blue. We do not display the device malware scenarios, that are not relevant for U2F, but in which case it naturally provides better security. To summarize, U2F is more secure against an attacker who can manipulate the display of the computer, or of course the phone itself. G2DT^{EXT} is more secure against an attacker who can manipulate the USB ports of the computer or the network. It is difficult to say which protocol provides the best security as it depends on more practical considerations, that we discuss in the Section 4.5.1.

4.3 Validating Attacks in Practice

Section Summary

We provide below a more practical description of a few selected types of attacks and weaknesses. Demonstrating that these attacks can be put in practice, albeit in laboratory conditions, validates our protocol and attacker models. Some of these attacks were found with PROVERIF, while others were discovered during the reverse engineering of the protocols. We do not claim novelty of those attacks, which are not particularly complex. We provide for each type of attacks

- the outline of the required steps of the attacks,
- high level comments about the severity of the attacks and an understanding of how they work and why they are possible;
- a description of how the attack was validated in a lab environment, that explains how exactly those attacks might be performed, by whom and what are the required capabilities.

Each attack was reproduced in a laboratory setting with laptops and an internet connection, using a dedicated Google account for the G2OT attacks, and the first version of the "Security Keys series by Yubico" along with an open source API[a] for the FIDO attacks. Remark that some of the following weaknesses might also be combined into stronger attacks.

[a] https://github.com/Yubico/libu2f-server

4.3.1 Session Confusion on g2V

Outline

The different steps of the attack are as follows:

1. The user enters their email and password, initiating a user session;
2. the browser informs the user that a code will be received on their phone;
3. the attacker enters the user's login and password on another computer, initiating an attacker session;
4. the attacker intercepts the code intended for the user session;
5. the attacker uses the code of the user session to validate the attacker session.

Comments

The fact that the code generated to validate the user session can be used to validate the attacker session may be surprising. It implies that the attacker does not need to intercept the code intended for their own session, but can use the code of any user session. This is an important observation: if the attacker uses for instance a key-logger, the code that the user enters on their computer is the first one received, which is most likely the code for the first session, i.e., the user session. If the codes were linked to the sessions on the server side, the code entered on their computer by the user would be useless to the attacker. We also remark that, as previously mentioned, the SMS channel might not provide a high level of security, at least compared to TLS. Hence, it might be possible for an attacker to obtain the verification code through a weakness of the SMS channel. This weakness does not directly lead to a severe attack but it may facilitate performing some of the following attacks.

Attack Validation

We created a fresh Google account and enabled the second factor authentication by associating a previously unregistered phone. Using two distinct computers, we initiated a first login attempt on the first one and received a first code. We then initiated a second session on the second computer and received a second code. The second code was then used to validate the first session, and conversely. This confirms that the code sent is not linked to a specific login attempt.

4.3.2 Session Confusion on g2OT

Outline

The different steps of the attack are as follows:

1. the user enters their password and email, initiating a user session;
2. the browser displays a request to confirm the request on their phone;
3. the attacker detects that the user contacted the server. After the first reply from the server the attacker blocks all further messages;
4. the attacker enters the user's login and password on another computer, initiating an attacker session;
5. depending on the timing, two things may then happen:
   - the user presses yes, nothing happens on their screen, and the attacker is logged in;
   - or the user presses yes, nothing happens on their screen, but another yes/no pops up on their phone. If the user presses yes once more, the attacker is logged in.

Comments

A robust implementation should reject any kind of simultaneous login from different sessions, or at least display it clearly on the phone, as it is done in the browser. We believe it to be plausible that users, after having pressed yes on their phone without a successful login, would press yes a second time. This attacks relies inherently on a lack of feedback given to the user, and a lack of a strong link between the computer that starts the session and the phone that validates it. This attack is concerning because of its simplicity. Google is implementing G2OT$^{\text{FPR}}$, but G2OT is still deployed on older mobile phones. It might be advisable to disable G2OT entirely.

Attack Validation

This attack was easy to reproduce in practice as it does not involve any complex manipulation. Using again a dedicated Google account, we

1. initiated two sessions for the same user on two distinct computers,
2. disconnected one computer from the network to reflect that the attacker blocks the network, and
3. validated the session of the other computer on the phone.

Sometimes we had to confirm twice on the phone to validate the malicious session, and sometimes only once. In a basic version of this attack, which does not require to block the network, an attacker observing the target user could initiate a session just a few moments after the user, and be logged in when the target validates on their phone. The target would see an error of the type *"Something went wrong"* on their computer and might retry to login. However, the error message

may appear before the user validates, as it appears as soon as a simultaneous login attempt is made. Thus, the more evolved version of this attack implies to block the Internet connection of the target computer after the user started their login. In our experiment, we simply temporarily disabled in step (2) the WiFi connection of the computer to disconnect the computer from the network. This effectively models an attacker that has the control over the network. Indeed, if the attacker controls the network, the attacker can detect all connections to the server IP address, and block all but the first connections through a firewall rule (although we did not implement the experiment detecting automatically the connection).

4.3.3 Phishing Attack on *Google 2-step*

Outline

The different steps of the attack are as follows:

1. the attacker directs the user to some malicious web page;
2. the attacker initiates a login attempt with the server, and the malicious web page simply forwards every information and query from the login attempt to the user through the malicious web page.

Comments

In [BHO+12], G2V was deemed secure with respect to phishing because they only considered passive phishing, where the attacker cannot for instance forward the query of the verification code to the user. We believe that it is necessary to consider active phishing as it is a reasonable capability nowadays. This kind of attack can be performed on a large scale without targeting a specific user. We argue that second factor authentication should efficiently protect against phishing, and even phishing combined with fingerprint spoofing, which are likely scenarios under which a user may wish to perform a secure login. Ideally, a second factor should even provide protection against this attack for a completely untrained human only following basic instructions.

Attack Validation

The core of this phishing attack is a *man-in-the-middle* attack. Interestingly, the interception can be completely invisible for the user. In our different examples, we will consider a user who wishes to login on *google.com*. Several user behaviors may be problematic when dealing with phishing:

- the user follows any untrusted link close enough to the authentic one, e.g. *google-security.com*;
- the user ignores an HTTPS warning;
- the user does not check that the protocol is HTTPS, but accepts HTTP.

We believe that most untrained users may be victims of the first two, and that even trained users do not always check that they are protected by HTTPS before providing their credentials. Depending on the attacker capabilities, many different kinds of phishing attacks might be performed, some of them difficult to avoid even for experienced users.

The most basic phishing attack is to get the user to click on a malicious link which is close to the official one. It can be performed for instance through a mail which invites the user to login

on *google-security.com* to solve some security issue. Here, the attacker may have obtained a valid certificate for the malicious domain, and the user will see a valid HTTPS connection.

Suppose we connect to a malicious WiFi Network. Different kinds of attacks can be performed. First, the malicious network may act as a free WiFi Hotspot network which requires third party authentication. Changing the DNS, for instance to contain the line "google.com IN A 192.168.0.1", the *google.com* domain will be redirected to an IP address controlled by the attacker. This may or may not raise an HTTPS error depending on the state of the cache of the user's browser. More precisely, as most websites, *http://google.com* contains a 301 redirect code to *https://google.com*. This redirection is cached by the browser according to the headers, which contain "Cache-Control: max-age=2592000". This means that the 301 redirect from HTTP to HTTPS is cached by the user browser for 2592000 seconds, i.e., 30 days. If the cache is still valid when the user connects to *google.com*, their browser remembers the 301 and connects to *https://google.com*. As the attacker cannot provide a valid TLS certificate the user sees an HTTPS warning, that the user may choose to ignore. If the cache has expired, which happens once every month, the user connects through HTTP to the malicious server, and believes to be on *google.com* without any warning displayed. If the user does not check for HTTPS, the phishing attempt succeeds.

To confirm this behavior, we forced the DNS client of a Linux machine to resolve the url "google.com" to a local IP address (editing the */etc/hosts* files). Then, when trying to connect to "http://google.com" in a completely fresh browser session, a local dummy page was displayed without any warning. In a browser session that was used previously to visit the honest Google website in the past 30 days, we obtained the HTTPS warning as the browser remembered the 301 redirect code.

We can design an even stronger attack, by setting up the WiFi network as a network which requires authentication through a captive portal. This is a feature classically supported by most access points, which can be provided with an URL or an IP address to which all users who try to login should be redirected. When performing an actual test for instance on Firefox, the browser detects that we are on a network which requires authentication, and proposes in a pop-up to redirect us to the captive portal. Even a trained user is likely to follow the link to have an Internet connection. The attacker can then redirect the user to a link of their choice: the attacker may redirect the user to *https://google-security.com* with a valid HTTPS certificate, or redirect through basic HTTP to a subdomain of *google.com* that does not exist, for instance *api.login.google.com*, and reconfigure the DNS as previously. As the subdomain does not exist, the attacker is ensured that the user's browser does not have any cached 301 redirection for this site. The user then connects to the attacker server via HTTP on a seemingly legitimate URL. Using a DNS redirection such that all websites are resolved to the captive portal (this is a classical implementations of the captive portal for WiFi hotspots), we were able to redirect the user to an arbitrary page, containing any arbitrary link, notably to a fake google page. The fake page corresponding to *http://api.login.google.com* was successfully displayed without warning. To complete the attack with *https://google-security.com*, we would need to register a TLS certificate for this domain. This could have be done for instance using the "Let's encrypt" certification system, although we did not perform this registration.

Some attacks could be avoided or at least complicated through the use of DNSSEC (which enforces a signature validation system for DNS requests) or HSTS (which declares that some website should only be accessed through HTTPS), but this is not supported by most websites, including *google.com*.

The phishing attacks can be made perfect (the user sees *google.com* under HTTPS but is connected to the attacker server) if the attacker can install a malicious HTTPS certificate on the user computer. On a computer running a Debian Linux distribution with libnss3-tools installed, this was achieved through the command *certutil -d sql:$ HOME/.pki/nssdb -A -t TC -n "mitm" -i malicious_ cert.pem*. We then successfully reproduced a valid HTTPS connection over a malicious

server by using the *mitmproxy* tool on Linux that allows to intercept all connections and mimic the behavior of a man-in-the-middle.

4.3.4 Action Confusion and Mixing on *Google 2-step* and U2F

Outline

The different steps of the attack are as follows:

1. the user initiates an untrusted login;
2. the attacker transforms the untrusted login into a trusted one;
3. the attacker uses the acquired cookie to perform other logins;

or

1. the user initiates an untrusted login;
2. the attacker initiates a trusted login;
3. the attacker uses the multi-factor actions made for the untrusted login to validate their session.

Comments

Using multi-factor authentication, a user may expect that after a successful login, once the user has disconnected from the computer, even an attacker with full control over the malicious computer should not be able to perform other logins. If the attacker can obtain a cookie through transforming an untrusted login into a trusted login, this property is violated. Note that this change may be completely invisible to the user, and hence the user may not check the list of trusted devices in the preferences of their account.

The core of this attack is an action confusion, where an intended action, the untrusted login, is transformed into another action, a trusted login. Another instance of action confusion occurs when a verification code intended for a login attempt is used by the attacker to reset the user's password. Note that every SMS from Google has the same content, independent of the action type. We recommend the SMS or the display of the second factor to provide the user with the intended action that is currently being validated. Untrusted login, Trusted login, Password Reset and deactivation of Multi-factor authentication are sensitive actions that should require multi-factor authentication and not be confusable.

This is particularly complicated for U2F, which does not provide the user with feedback through the second factor.

Attacker Validation

We were able to perform several sequences of actions that can lead to action confusion.

First, after a successful second factor login, multi factor parameters of the account can be accessed by only retyping the password. Hence, the second factor protection can be disabled. Although an e-mail is sent to notify the user about this change, as the attacker is already logged in, the attacker can simply delete the e-mail. Once the second factor protection is disabled, the attacker can login

from any untrusted computer. This is not *per se* an action confusion, but we note that a code intended for a login also allows an attacker to change the authentication settings of the user.

Second, many browser extensions, e.g., *Greasemonkey* or *TamperMonkey*, allow the behavior of specific pages to be changed. On the Google login page, we were able thanks to a five line JavaScript code to hide the *"I trust this computer"* check-box by adding the attribute *style="visibility:hidden;"* to its *"div"*. As the box is checked by default, we were able to perform a login with a second factor enabled where the box was invisible. The platform then became trusted, and we were able to perform a second login where only the password was required, and the second factor was not asked.

Third, we recall that a login validation and a password reset action yield the same SMS. By initiating simultaneously a login attempt and a password reset, we receive two similar SMS. An attacker may thus initiate a password reset while the user is trying to log into their account. The user will receive the code for the password reset, that the attacker may intercept, e.g., using a key-logger, and change the user's password. We reproduced those attacks using a dedicated Google account and a phone as a second factor.

4.3.5 USB Attack on U2F

Outline

The different steps of the attack are as follows:

1. the user initiates a login;
2. the attacker initiates another login;
3. the attacker sends to the token the payload corresponding to their session;
4. the attacker sends to the token the payload corresponding to the user session;
5. the user presses once the button of the token;
6. the attacker get backs the signed data, and completes their login;
7. after the first press, the token keeps blinking without any change;
8. the user presses again, and validates their session.

Comments

This weakness is inherent to the fact that the U2F factor does not provide feedback to the user. Therefore, the user is unable to know which action is actually validated when pressing the token button. Moreover, when submitted two queries in a row, the token will simply keep blinking after the first press. Given that at least some tokens have touch buttons (and not a press button) the user may have the impression that the press was unsuccessful.

The weakness is also related to the fact that U2F does not have an independent communication channel with the server, and cannot provide any security when plugged into a malicious computer. We believe however that providing a secure one time only login on a malicious computer is a reasonable user expectation.

Attack Validation

We performed our tests on the Yubikey U2F FIDO security key, which is equipped with a touch button. Using an open source API[1], we were able to submit two simultaneous signature requests to the token, in a similar way that the requests are submitted by the browser. The token then started blinking as usual, expecting a user touch to confirm the signature. After touching once the button, it kept blinking as if the press were unsuccessful. Pressing once again, we received both signatures through the API. Hence, we could implement a malware that would detect an honest request and would submit a second request at the same time. Then, the user may press once, believe that the press was not registered and thus press once again, thus validating without their consent for the two requests. This problem could be avoided by forbidding two simultaneous requests, and simply dropping any request as long as the current one is not validated.

4.4 Unlinkability

Section Summary

We analyse unlinkability of the U2F protocol, but not of *Google 2-step* as it is directly linked to Google and the cellphone. The formal analysis allows to rediscover two known attacks. However, we remark that these two attacks can actually be combined into a more severe attack.

4.4.1 On Privacy

So far, our analysis did not cover privacy considerations. As we assumed that the attacker knew the user password, we also assumed that they knew the user's identity. *Google 2-step* does not provide privacy guarantees as users provide their phone number to the server. However, U2F is advertised as enforcing privacy: it can be read on Yubiko's website that by using a fresh key pair for every account "Example.com cannot know whether User1 and User2 shares the same device." [2]. The FIDO alliance also backs up some claims about the unlinkability of U2F by advertising "No linkability between services or accounts" [3].

We analysed unlinkability in the U2F registration and authentication protocols and were able to observe two weaknesses. We noted that, independently, both weaknesses were already acknowledged in the FIDO specification, although deemed either not critical or too costly to fix. We will describe the two attacks on privacy, and argue that, when combined, their impact may be higher. We propose several possible fixes to be considered.

4.4.2 Formal Analysis

Unlinkability captures the fact that a server should not be able to tell if two users share the same device or not. ProVerif supports verification of indistinguishability properties, modelled as an observational equivalence between processes. Observational equivalence expresses the fact that the attacker cannot know with which of the two process they are interacting with.

[1] https://github.com/Yubico/libu2f-server
[2] `https://developers.yubico.com/U2F/Protocol_details/Overview.html`, section 3
[3] `https://fidoalliance.org/about/what-is-fido/`

$T_{reg} \hat{=}$
  let $fsk = \mathtt{smac}((rnd, sk_i), sk)$ **in**
  let $handle = (\mathtt{mac}((tsk, URL), sk_i), rnd, \mathtt{pk}(fsk))$ **in**
  out$(USB, handle)$.

$T_{auth} \hat{=}$
  in$(USB, (handle))$;
  let $fsk = \mathtt{smac}((rnd, sk_i), sk_i)$ **in**
  if $handle = (\mathtt{mac}((tsk, URL), sk), rnd, \mathtt{pk}(fsk))$ **then**
    out$(USB, \mathtt{sign}((URL, challenge), fsk))$.

Figure 4.2: U2F Token Modelling

A U2F token may generate a fresh key for every new registration. We model here the key generation process of the Yubikey [4], which follows *FIDO's U2F*'s guidelines. It has been designed so that the device does not need any writable memory. To register to a new server, the U2F token proceeds as follows, where sk is the token's global secret key, and mac denotes a MAC function:

- generate a fresh random rnd;
- compute the fresh secret key $fsk := mac((rnd, sk), sk)$;
- compute the corresponding public key fpk;
- compute the key handle, which is the triple $mac(fsk, sk)$, rnd, fpk;
- output the key handle.

Including the random rnd in the key handle allows the token to recompute the secret key; the MAC guarantees the authenticity of the secret key, i.e., it ensures that the secret key was indeed issued by the token.

When receiving a login request the server replies by sending a challenge together with the key handle. The token responds with a signature on the challenge and the URL. The model of the U2F token, with one process for the registration and another for authentication, is given in Figure 4.2.

The process parameters are the token's secret key sk_i, used to derive new keys, and the server's URL. The input of the authentication process is given by the attacker, modeling a malicious server. tsk is the new secret key registered on the server side through the corresponding public key $\mathtt{pk}(tsk)$ and the key $handle$. Given these processes, we can express our security property as

$$\|^i(T_{reg} \| T_{auth}) \cong \|^j(\|^{i \leq 1}(T_{reg} \| T_{auth}))$$

where $\simeq$ denotes observational equivalence. Observational equivalence models the attacker's inability to distinguish the left-hand and the right-hand processes [ABF17]. The above property expresses the fact the server, identified by URL, cannot know if it is interacting with a single token, as in the left hand-side, or with multiple different tokens, as on the right-hand side.

4.4.3 Attack against Key Generation

The FIDO overview [5] documents an attack where a server may check if two users share a same token, i.e., if two accounts do in fact correspond the same person. Using the above modelling, ProVerif allows us to capture this attack. Considering two accounts on the server, one with login $log1$ and key handle $kh1$, and the other $log2$ and $kh2$, the server may

[4] `https://developers.yubico.com/U2F/Protocol_details/Key_generation.html`
[5] `https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.pdf`

- receive a login attempt from $log1$, and answer with $kh2$
  - if both $kh1$ and $kh2$ were generated by the same token, the token will accept $kh2$ and provide the corresponding signature;
  - else, it will fail;
- if the server receives a signature, it links $log1$ and $log2$, and, otherwise, differentiates them.

This attack in itself may not be critical, as the server must target two particular users that it wishes to distinguish.

4.4.4 U2F with Counters

The U2F protocol uses global counters that are incremented at every signature made by the token with the counter value included in the signature. These counters tracking the number of signatures issued by the token allow to detect device cloning or key leakage: if a server observes two login attempts with inconsistent counter values, they may have been produced by two different tokens, i.e., the original token and a clone.

Related privacy considerations are briefly mentioned in *FIDO's U2F webauthn* standard[6]. They mention the risk that a global counter for all key pairs can produce correlation attacks. Indeed, even an honest but curious server may check that the login sequences, ordered by timestamp, made by two of its users form a strictly increasing sequence of counter values. If this is verified for a large number of values, the two accounts are likely to share the same device. Providers sharing their information may also allow this attack. Using separate counters for each key pair would avoid this attack, yet this increases the device cost, and Yubikeys currently use a single counter.

4.4.5 An Attack Based on Global Counters

We integrated the use of counters in our formal model. We use private channels to "store" the current value of a counter, and model integers by a zero constant $zero$ and a successor function $s()$. ProVerif finds an attack which corresponds to the basis of the correlation attack. Consider two accounts, with respective logins $log1$ and $log2$, with no previous login attempt. The server may:

- receive a first login request for the login $log1$, and obtain the corresponding token's counter value, which is 0;
- receive a first login request for the ling $log2$, and obtain the corresponding token's counter value, which is either 0 or 1;
- If the value is 1, they share the same token, else they do not.

If $log1$ and $log2$ share the same token, its counter is increased by both logins, which is not the case when different tokens are used.

We also modeled the version where tokens maintain different counters for each key pair instead of a global one. ProVerif proves that in this case the U2F tokens are indeed unlinkable.

4.4.6 Combining Both Attacks

We note that the two previous attacks might be combined. While the first attack is "visible", i.e., the fact that the server replies with a wrong key handle is observable, the second attack is not

[6] `https://www.w3.org/TR/webauthn/#sign-counter`

observable, but does not allow to link users with certainty. A server that does not wish to be detected while misbehaving, often modelled by a *honest-but-curious* server, is unlikely to perform the first attack. However, combining the attacks, a server may try to link together accounts probabilistically using their counters, and then confirm its suspicions using the first attack. This scenario is currently possible on Yubikey's implementation.

4.4.7 Improvements

In the attack against the key generation, the root cause is that the key pair is not strongly linked to the corresponding account. For instance, if the user login was also to be included in the MAC provided in the key handle, the token could detect a mismatch and provide an error. This requires that the browser is capable of forwarding the login name to the token, it must therefore be able to extract the login of the user from the login page. This might be possible through an authentication API that the login page must follow, and might be included for instance in the webauthn API.

The attack based on counters can be avoided by using a different counter for each key pair. This however requires more costly tokens. Counters could also be completely removed, losing the possibility to detect cloning. We note that cloning can only be detected; it does not prevent an attacker from performing a login. Typically an attacker having cloned a device may use a very large counter. This counter is likely to be consistent, i.e., larger, than the counter on the original token. In that case cloning will only be detected when the honest user performs a login after the attacker. Hence, the use of one or no counters is a question of priority of whether one favors privacy or clone detection.

4.5 *Google 2-step* vs U2F

Section Summary

We provide a final comparison of the *Google 2-step* and U2F approaches to MFA. We first look at some practical considerations that are outside the scope of our threat model, and then conclude based on both our analysis and those practical considerations. The comparison does not say which approach is the best, but provides insight on the key points to look at if one desires to choose between the two solutions in a given scenario.

4.5.1 Practical Considerations

As mentioned previously, there are some interesting aspects that are outside of the scope of our threat models and formal analysis. We therefore discuss below some additional thoughts and findings.

Independence of the Second Factor

When trying to log into an account from a compromised computer, we observed that the U2F token might be used by the attacker if the attacker controls the channel used for communication with the second factor. Therefore, the U2F approach cannot provide strong protection against malwares on the user computer. The risk is mitigated by the fact that the attacker may only perform a single action authenticated by the second factor, but if this action can be used to deactivate

the second factor, or reset the user password, the user account may be completely compromised by this single action. The approach of *Google 2-step* provides a second communication channel that is independent from the computer, and may enable security even on a completely untrusted computer.

On the Need for Feedback

An advantage of the phone over the U2F token is the feedback provided to the user. In particular, on *FIDO's U2F*, two consecutive button presses may remain unnoticed. On the phone, a success or failure confirmation after pressing the button is easily provided. Moreover, the phone can be used to produce improved versions of U2F, where the display is enriched with additional information, as we did for G2DT$^{\text{DIS}}$. We note that FIDO proposes the "secure transaction" mechanism, which specifies that second factors might use a display. However, the message content is not included in the standardization.

Storing the Keys on a Dedicated Secure Token

An advantage of the U2F token is that, even if a computer is compromised, the number of attacker logins is limited by the number of times the button is pressed. This is due to the fact that keys are stored on a token and not completely compromised. If keys are stored on a computer or a smartphone, a malware may extract them. As discussed previously, U2F does not provide perfect security either. Although keys are more difficult to compromise, one should be careful about how the token is used to ensure that no unwanted computer becomes trusted, or that a user does not press the button twice in a row. A solution to mitigate key leakage for computers or smartphones could be to consider an Isolated Execution Environment, such as Intel SGX, ARM TrustZone or a Trusted Platform Module.

Carrying Additional Authenticators

An important aspect of multi-factor protocols is of course usability. From that point of view, the need to buy and carry an additional token may be cumbersome. Nowadays, more and more people possess and constantly carry their phone, making it a natural choice for a second factor.

Disabling the Second Factor

On some websites, for instance GitHub, disabling the second factor (and then changing the password) does not require the use of the second factor, once a login was performed. It seems advisable to require a second factor authentication to disable the mechanism.

4.5.2 Final Comparison

It is difficult to compare the two approaches which are quite different. We try to provide a brief summary of the main advantages of both (we consider here U2F with a dedicated USB token):

- *Google 2-step* provides an independent channel of communication with the server, and feedback through the display;

- *Google 2-step* may be compromised by malware on the phone;
- U2F provides privacy (if implemented correctly);
- U2F may suffer from key leakage or device cloning;
- U2F requires an additional device;
- U2F does not provide enough feedback.

If we consider U2F where the phone is used as the dedicated token to store the keys and perform the cryptographic operations, U2F may provide enough feedback to the user (fingerprint, trusted login attempt,...) and would not require carrying another device. We would however potentially lose the privacy, the key storage would need to be completely secured and isolated, it could be a victim of malware, and we need a convenient mechanism to set up a channel between a phone and a computer.

Against both versions of U2F, *Google 2-step* provides better security against some critical scenarios (connection to a dishonest network or on a corrupted computer). Yet, *Google 2-step* is currently unable to provide unlinkability.

Part II

Modular

In which we try to bring some modularity to our proofs

5 A Composition Framework in the Computational Model

> One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.
>
> *(Ada Lovelace)*

5.1 Introduction

Our goal is to strive for formal and mechanized proofs of complex protocols, against attackers as powerful as possible. In the previous Part, we exposed what an extensive analysis can look like. It was performed in the symbolic model, and for a class of protocols that does not involve complex primitives. Performing such an analysis for more complex protocols, e.g., an e-voting protocol, or in the computational model is currently out of reach. To tackle complex protocols in the computational model, we must find ways to simplify the proofs. To this end, we consider in this Part security proofs of composed protocols, where the proof of the protocol is derived from multiple simpler proofs.

We consider the security property $P\|R \cong Q\|R$. It is well known that when R does not share any secrets, i.e., any randomness with P and Q, we have that $P \cong Q$ implies that $P\|R \cong Q\|R$. In this case, the idea is that the attacker can actually simulate R by sampling itself the values of the names used only by R, and produce messages that have exactly the same distribution as the one produced by R. In essence, R does not provide any useful information to try to break $P \cong Q$. Thus, when trying to (de-)compose security properties, the main difficulty comes from the fact that different protocols may share some secrets. For instance, R could provide a decryption oracle for a secret key appearing in P and Q, thus breaking the security of P. In practice, situations arise where we want to split a protocol into components that share some secrets.

This is typically the case for multiple sessions of the same protocol, or for key exchange protocols, which result in establishing a shared secret that will be later used in another protocol. Protocols may also share long term secrets, for instance the same signing key may be used for various authentication purposes. Another example is the SSH protocol with the forwarding agent feature [YL], which we will consider later. The forwarding feature allows to obtain, through previously established secure SSH connections, signatures of fresh material required to establish new connections. It raises a difficulty, as signatures with a long term secret key are sent over a channel established using the same long term secret key.

When decomposing the security of a composed protocol into the security of its components, we would like to break a complex proof into simpler proofs, while staying in the same proof framework. This is also a difficulty since the attacker on a protocol component might use the other components: we need a proof with respect to a stronger attacker. In [BDF+18], such a strong attacker can be simulated by a standard one, because there is no shared long term secret.

Chapter Summary

We design a method that allows to decompose a security property of a compound protocol into security properties of its components. This works for parallel composition, but also sequential composition and replication: we designed a reduction from the security of multiples copies of a protocol to a security property of a single copy. Our method works even if the various components share secrets and (in case of sequential composition) when a state is passed to the other component.

We illustrate our composition results showing how to split the security of any (multi-session with shared long term secret) composed key exchange into smaller proofs. Actual proofs of protocols are delayed to Part IV, where the proofs are mechanized.

We generalize the application to key exchanges performing key confirmations, i.e., using the derived key in the key exchange (as in TLS). The generalization is simple, which is a clue of the usability of our framework.

5.1.1 Our Contributions

We provide a composition framework that reduces the security of a compound protocols to the security of its components. We allow both state passing and shared long term secrets. The framework relies on the definitions of protocols and the computational semantics of Figure 2.3 (its protocol algebra was designed in order to provide a suitable composition algebra for our framework). Our main composition Theorems are generic: the classical game based setting can be used to prove the sub-goals, or the BC logic, as it will be shown in Chapter 6.

The starting idea is simple: if we wish to prove the security of a composed protocol $P\|Q$, it is sufficient to prove the security of P against an attacker that may simulate Q, maybe with the help of an oracle. If $\overline{n}$ are the secrets shared by P and Q, this simulation has to be independent of the distribution of $\overline{n}$. This is actually an idea that is similar to the key-independence of [BFS+13].

Therefore, we first introduce the notion of $\mathcal{O}$-simulation, in which an oracle $\mathcal{O}$ holds the shared secrets: if Q is $\mathcal{O}$-simulatable and P is secure against an attacker that has access to $\mathcal{O}$, then $P\|Q$ is secure. Intuitively, $\mathcal{O}$ defines an interface through which the secrets can be used (e.g., obtaining signatures of only well tagged messages). $\mathcal{O}$ simulatable protocols conform to this interface.

We extend this basic block to arbitrary parallel and sequential compositions, as well as replication of an unbounded number of copies of the same protocol. In the latter case, the security of a single copy of P against an attacker that has access to an oracle allowing to simulate the other copies, requires to distinguish the various copies of a same protocol. In the universal composability framework, this kind of properties is ensured using explicit session identifiers. We rather follow a line, similar to [KR17], in which the session identifiers are implicit.

5.1.2 Related Work

The security of composed protocols has been widely studied in the last two decades. For instance, Universal Composability (UC) and simulation based reductions [Can00; CR03; BPW07; HS15; BDH+08; CKK+19] and other game-based composition methods [Mau11; BFS+13; Bla18; BDF+18] address this issue. While the former proceed in a more bottom-up manner (from secure components in any environment, construct secure complex protocols), the latter proceed in

a more top-down way: from the desired security of a complex protocol, derive sufficient security properties of its components. Such "top-down" proofs design allows more flexibility: the security requirements for a component can be weaker in a given environment than in an arbitrary environment. The counterpart is the lack of "universality": the security of a component is suitable for some environments only.

We follow here the "top-down" approach. While we aim at designing a general methodology, our target is the management of formal security proofs in the BC logic [BC14a]. We do not review here the composition results in the symbolic model, as we are directly in the computational model. As far as we know, the existing composition results that follow the "top-down" approach cannot be used in situations where there is both a "state passing", as in key exchange protocols, and shared long term secrets. For instance, in the framework of [BDF+18], the same public key cannot be used by several protocols, a key point for reducing security of multiple sessions to security of one session.

We introduce the composition problem through a process algebra: protocols are either building blocks (defined, e.g., with a transition system) or composed using parallel and sequential composition, and replication. This prevents from committing to any particular programming language, while keeping a clean operational semantics. This approach is also advocated in [BDF+18], which follows a similar approach. Other works on composition (e.g., [Mau11; BDH+08]) rely on specific execution models.

Our starting idea, to prove a component w.r.t. a stronger attacker that has access to the context, is not new. This is the basis of many works, including [BFW+11; BFS+13; Bla18; BDF+18]. The main difference, that we wish to emphasize, is that these works do not support long term shared secrets, used in different components. Notably, the oracles of [BDF+18] are only used to decompose protocols with state passing. Our notion of simulatability allows sharing long term secret by granting the attacker access to oracles that depend on the secrets (for instance, signing oracles). It also allows a symmetric treatment for proofs of a protocol and proofs of its context.

For several specific problems, typically key exchanges, there are composition results allowing to prove independently the key exchange protocol and the protocol that uses the exchanged key [BFW+11; BFS+13; Bla18; FG14; KR17]. In such examples, the difficulty also comes from the shared secret, especially when there is a key confirmation step. In that case, the derived key is used for an integrity check, which is part of the key exchange. Then the property of the key exchange: "the key is indistinguishable from a random" does not hold after the key confirmation and thus cannot be used in the security proof of the protocol that uses this exchanged key. In [BFS+13], the authors define the notion of key independent reduction, where, if an attacker can break a protocol for some key distribution, they can break the primitive for the same distribution of the key. This is related to our notion of simulatability, as interactions with shared secrets are captured by an oracle for fixed values of the key, and thus attacks on the protocol for a fixed distribution are naturally translated into attacks against the primitive for the same distribution. Key exchanges with key confirmation are therefore a simple application of our composition results. Along the same line, [FG14] extends [BFW+11] to multi staged key exchanges, where multiple keys might be derived during the protocol. While we do not directly tackle this in our work, our framework could be used for this case.

The authors of [Bla18] also provide results allowing for the study of key renewal protocols (which we capture with the sequential replication Theorem), and has the advantage to be in a mechanized framework, while we only cast our results in a mechanizable framework. It does not however consider key confirmations.

The UC framework initiated by [Can00] and continued in [CKK+19; HS15; BPW07] is a popular way of tackling composition. As explained above, this follows a "bottom-up" approach, in which protocols must be secure in any context, which often yields very strong security properties, some of

which are not met in real life protocols. Moreover, to handle multiple sessions of a protocol using a shared secret, joint-state theorems are required. This requires a tagging mechanism with a distinct session identifier (sid) for each session. Relaxing this condition, the use of implicit session identifiers was established in [KT11] for the UC framework, ideas continued in [KR17] for Diffie-Hellman key exchanges, where they notably provide a proof of the ISO 9798-3 [Iso] protocol.

We do not consider a composition that is universal: it depends on the context. This allows us to relax the security properties regarding the protocol, and thus prove the compositional security of some protocols that cannot be proved secure in the UC sense. We also rely on implicit sids to prove the security of multiple sessions. Some limitations of the UC framework are discussed in [BFW+11, Appendix A].

In [BDH+08], the authors also address the flexibility of UC (or reactive simulatability) showing how to circumvent some of its limitations. The so-called "predicates" are used to restrict the order and contents of messages from environment and define a conditional composability. Assuming a joint-state conditional composability theorem, secret sharing between the environment and the protocol might be handled by restricting the accepted messages to the expected use of the shared secrets. However, the framework does not cover how to prove the required properties of (an instance of) the environment.

Protocol Composition Logic is a formal framework [DMP03] designed for proving, in a "Dolev-Yao model", the security of protocols in a compositional way. Its computational semantics is very far from the usual game-based semantics, and thus the guarantees it provides [DDM+05] are unclear. Some limitations of PCL are detailed in [Cre08].

Summing up, our work is strongly linked to previous composition results and captures analogues of the following notions in our formalism: implicit disjointness of local session identifiers [KT11], single session games [BFW+11], key-independent reductions [BFS+13] and the classical proof technique based on pushing part of a protocol in an attacker, as recently formalized in [BDF+18]. We build on all these works and additionally allow sharing long term secrets, thanks to a new notion of $\mathcal{O}$-simulatability. This fits with the BC model: the formal proofs of composed protocols are broken into formal proofs of components. All these features are illustrated by a proof of SSH with (a modified) forwarding agent.

? Future Work

The framework could be used to consider more complex protocols, such as for multi-party computation protocols. While we chose to use later on the BC logic to perform cases studies leveraging our framework, it could also be used to help proving complex protocols in EasyCrypt [BGH+11] for example, as security w.r.t. an attacker accessing an oracle can be formalized in this tool.

5.2 Protocols and Indistinguishability

Section Summary

Classical indistinguishability specifies that an attacker interacting with either oracle $\mathcal{O}_1$ or $\mathcal{O}_2$, both oracles modelling protocols, cannot know with which of the two oracles they are interacting. We extend the definitions of Section 2.3.1 by giving the attacker access to a stateless oracle $\mathcal{O}$ to increase their capabilities. Classical indistinguishability is implied by the $\mathcal{O}$-indistinguishability, for any given oracle $\mathcal{O}$.

5.2.1 Stateless Oracle Machines

For reasons that have been explained in the introduction, we wish to extend the semantics of protocols and their indistinguishability (Chapter 2) to attackers that have access to an additional stateless oracle. At this stage, we need stateless oracles in order to be compositional. Let us explain this. Assume we wish to prove a property of R in the context $P\|Q\|R$. The idea would be to prove R, interacting with an attacker that simulates $P\|Q$. This attacker is itself a composition of an attacker that simulates P and an attacker that simulates Q. The protocols P, Q, R share primitives and secrets, hence the simulation of P, Q requires access to an oracle that holds the secrets. If such an oracle were to be stateful, we could not always build a simulator for $P\|Q$ from simulators of P, Q respectively, since oracle replies while simulating Q could depend on oracle queries made while simulating P, for instance.

We now refer to stateless oracles as simply oracles, that should not be confounded with protocol oracles (Definition 2.9) that are stateful through their history tape. The oracles depend on a security parameter η (that will not always be explicit), (secret) random values and also draw additional coins: as a typical example, a (symmetric key) encryption oracle will depend on the key k and use a random number r to compute $\mathtt{enc}(m, r, k)$ from its query m. Therefore, the oracles can be seen as deterministic functions that take two random tapes as inputs: ρ_s for the secret values and $\rho_{\mathcal{O}}$ for the oracle coins.

Formally, oracles take as input tuples $(\overline{m}, r, s)$ where $\overline{m}$ is a finite sequence of bitstrings, r is a handle for a random value and s is a handle for a secret value. r and s are respectively used to extract the appropriate parts of $\rho_{\mathcal{O}}, \rho_s$ respectively, in a deterministic way: the randomness extracted from $\rho_{\mathcal{O}}$ is uniquely determined by $\overline{m}, r, s$ and the extractions for different values do not overlap.

In what follows, we only consider oracles that are consistent with a given functional model $\mathcal{M}_f$. Such oracles only access ρ_s through some specific names. This set of names is called the *support* of the oracle.

Example 5.1. An encryption oracle for the key k (corresponding to the handle "1"), successively queried with $(m, 1, 1), (m', 2, 1)$, $(m, 3, 1)$, $(m, 1, 1)$, $(m', 2, 2), \ldots$ will produce respectively the outputs $\mathtt{enc}(m, r_1, k)$, $\mathtt{enc}(m', r_2, k)$, $\mathtt{enc}(m, r_3, k)$, $\mathtt{enc}(m, r_1, k)$, $\bot, \ldots$ Here r_1, r_2, r_3 are non-overlapping parts of $\rho_{\mathcal{O}}$ (each of length η). The support of this oracle is $\{k\}$.

Technical Details

The formal definition of stateless oracles is a bit involved, notably to formally specify the randomness extraction. This construction is required to ensure the determinism of the oracles. Determinism is required to build a single simulator for two parallel protocols from the individual simulators for the two protocols.

For instance, for an oracle performing randomized encryption, rather than always encrypting with a fresh nonce, this system allows multiple attackers to obtain an encryption of a message with the same random.

Definition 5.1 ((Stateless) Oracle)**.** An *oracle* $\mathcal{O}$ is a triple of functions that have the following inputs

- a sequence of bitstrings $\overline{w} \in (\{0,1\}^*)^n$ and two bitstrings r, s: the query, consisting of an *input query* $\overline{w}$, an *input tag* r, an *input key* s;
- a random tape ρ_s for the (secret) random values;
- the security parameter η;
- a random tape $\rho_{\mathcal{O}}$ for the oracle's coins.

The first function assigns to each $\overline{w}, s, r$ an integer $n(\overline{w}, s, r) \in \mathbb{N}$ and is assumed injective. $n(\overline{w}, s, r)$ is used to extract a substring $e_1(n(\overline{w}, s, r), \eta, \rho_{\mathcal{O}})$ from $\rho_{\mathcal{O}}$, which is uniquely determined by the input. We assume that the length of the substring extracted by e_1 only depends on η, and substrings extracted with e_1 are disjoint for different values of n.

The second function e_2 assigns to each s a sequence $\overline{p}(s)$ of natural numbers, that are used to extract secret values from ρ_s: $e_2(s, \eta, \rho_s)$ is a sequence of bitstrings. It is also assumed to be injective.

The third function takes $\eta, \overline{w}, r, s, e_1(n(\overline{w}, s, r), \eta, \rho_{\mathcal{O}}), e_2(s, \eta, \rho_s)$ as input and returns a result (a bitstring) or a failure message.

Example 5.2. Expanding upon Example 5.1, the encryption oracle is given by the triple of functions (e_1, e_2, e_3) such that:

- $e_1(n(\overline{w}, s, r), \eta, \rho_{\mathcal{O}})$ extracts the substring r at position range $[n(\overline{w}, s, r) \times \eta, (n(\overline{w}, s, r) + 1) \times \eta]$ from $\rho_{\mathcal{O}}$.
- $e_2(s, \eta, \rho_s) = \begin{cases} [\![k]\!]^{\eta}_{\rho_s} \text{ if } s = 1 \\ 0 \text{ else} \end{cases}$
- $e_3(\eta, \overline{w}, r, s, e_1(n(\overline{w}, s, r), \eta, \rho_{\mathcal{O}}), e_2(s, \eta, \rho_s)) = [\![\mathtt{enc}(y, r, x)]\!]^{\eta}_{\{y \mapsto \overline{w}, r \mapsto r, x \mapsto e_2(s, \eta, \rho_s)\}}$

Given η, and a sequence of bitstrings m, we call r_1 the sequence of bitstrings at position range $[n(m, 1, 1) \times \eta, (n(m, 1, 1) + 1) \times \eta]$ from $\rho_{\mathcal{O}}$. Then, on input $(m, 1, 1)$, $e_1(n(m, 1, 1), \eta, \rho_{\mathcal{O}}) = r_1$, $e_2(1, \eta, \rho_s) = [\![k]\!]^{\eta}_{\rho_s}$ and the oracle returns $e_3(\eta, m, 1, 1, r_1, [\![k]\!]^{\eta}_{\rho_s}) = [\![\mathtt{enc}(y, r, k)]\!]^{\eta}_{y \mapsto m, r \mapsto r_1}$.

We now replace the previous Definition 2.9 of PTOMs by adding tapes and access to the oracle.

Definition 5.2 (Protocol Oracle)**.** A *Polynomial Time Oracle Machine* (PTOM) is a Turing machine denoted by $\mathcal{A}^{\mathcal{O}, \mathcal{O}_P}$ and equipped with:

- an input/working/output tape (as usual; it is read/write);
- a read-only random tape ρ_r (attacker's coins);
- an oracle input tape $\rho_{\mathcal{O}}$;
- an oracle output tape, which is read-only.
- a protocol oracle and oracle read-only random tape ρ_s (not accessible by the Turing Machine);
- a protocol oracle input tape;
- a protocol oracle history tape θ;
- a protocol oracle output tape.

Note that once the oracle's random tape is fixed, we ensure that all our oracles are deterministic.

As previously, we distinguish between the inputs that the machine can access and the inputs that can be accessed by the oracle only; we use the notation $\mathcal{A}^{\mathcal{O}(\rho_s, \rho_{\mathcal{O}}), \mathcal{O}_P(\rho_s)}(\omega, \rho_r)$ for a PTOM with access to the oracle $\mathcal{O}$ and the protocol oracle $\mathcal{O}_P$. We will often omit to specify the oracles

argument, and simply write:

$$\mathcal{A}^{\mathcal{O},\mathcal{O}_P}(\omega,\rho_r)$$

Similarly to protocol oracles, these definitions extend to multiple oracles $\langle\mathcal{O}_1,\ldots,\mathcal{O}_n\rangle$, prefixing the query with an index in $\{1,\ldots,n\}$. We will often write $\mathcal{A}^{\mathcal{O}_1,\ldots\mathcal{O}_k}(\omega,\rho_r)$ for $\mathcal{A}^{<\mathcal{O}_1,\ldots\mathcal{O}_k>}(\omega,\rho_r)$. We may finally consider multiple oracles that combine protocols oracles and stateless oracles. $\mathcal{A}^{\langle\mathcal{O}_1,\ldots,\mathcal{O}_m\rangle,\langle\mathcal{O}_{P_1},\ldots,\mathcal{O}_{P_n}\rangle}$ is also written $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_m,\mathcal{O}_{P_1},\ldots,\mathcal{O}_{P_n}}$.

We can then extend Definition 2.12 to the new PTOMs.

Definition 5.3. Given a functional model $\mathcal{M}^f$, an oracle $\mathcal{O}$ and protocols P,Q, we write $P\cong_{\mathcal{O}} Q$ if for every PTOM $\mathcal{A}^{\mathcal{O}}$, the attacker's advantage $\mathrm{Adv}^{P\cong_{\mathcal{O}}Q}$ equals to

$$|\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O},\mathcal{O}_P(\rho_s)}(\rho_r,1^\eta)=1\}-\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O},\mathcal{O}_{Q_1}(\rho_s)}(\rho_r,1^\eta)=1\}|$$

is negligible in η.

Remark that by giving an oracle to the distinguisher, we strictly increase its power. Thus, we trivially have that for any protocols P,Q and oracle $\mathcal{O}$, $P\cong_{\mathcal{O}} Q\Rightarrow P\cong Q$.

Technical Details

Once again, the new definition actually depends on the matching between the oracles modelling parallel protocols and the actual behaviour of parallel protocols.

Lemma 5.1. *For protocols P,Q,A,B, an oracle $\mathcal{O}$ and a list $\mathcal{O}_l$ of protocol oracles,*

$$\begin{array}{l}|\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O},\mathcal{O}_l,\mathcal{O}_{A\|P}(\rho_s)}(\rho_r,1^\eta)=1\}\\ \quad-\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O},\mathcal{O}_l,\mathcal{O}_{B\|Q}(\rho_s)}(\rho_r,1^\eta)=1\}|\end{array}=\begin{array}{l}|\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O},\mathcal{O}_l,\mathcal{O}_A(\rho_s),\mathcal{O}_P(\rho_s)}(\rho_r,1^\eta)=1\}\\ \quad-\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}^{\mathcal{O},\mathcal{O}_l,\mathcal{O}_B(\rho_s),\mathcal{O}_Q(\rho_s)}(\rho_r,1^\eta)=1\}|\end{array}$$

Proof. For protocols P,Q such that $\mathcal{C}(P)\cap\mathcal{C}(Q)=\emptyset$, for any message m, random tape ρ_s and history tape θ, we have by definition of the semantic of $\|$ and the definition of the parallel oracles:

$$\mathcal{O}_{P\|Q}(\rho_s,\theta)(m)=<\mathcal{O}_P,\mathcal{O}_Q>(\rho_s,\theta)(m)$$

The desired result then immediately follows. ■

5.3 Simulatability

Section Summary

We define a notion of "perfect" simulation, where a protocol depends on some secrets that the attacker can only access through an oracle, and an attacker must be able to produce exactly the same message as the protocol. This means that an attacker, given access $\mathcal{O}$ but not to a set of secrets $\overline{n}$, can completely simulate the protocol P (using $\mathcal{O}$ to have a partial access to the secrets), i.e., produce exactly the same distribution of message.

Formally, given a set of names $\overline{n}$, an oracle $\mathcal{O}$ and a protocol P. We say that $\nu\overline{n}.P$ is $\mathcal{O}$-simulatable, if there exists a PTOM $\mathcal{A}^{\mathcal{O}}$ such that for any attacker $\mathcal{B}^{\mathcal{O}}$, the sequences of messages produced by $\mathcal{B}^{\mathcal{O},\mathcal{O}_P}$ has exactly the same probability distribution as the on produced by $\mathcal{B}^{\mathcal{O}}$ interacting with $\mathcal{A}^{\mathcal{O}}$ instead of $\mathcal{O}_P$.

Assume that $Q \cong_{\mathcal{O}} R$ and $\nu\overline{n}.P$ is $\mathcal{O}$-simulatable, where $\overline{n}$ contains the secrets shared by P, Q and R. Any distinguisher against $Q \cong_{\mathcal{O}} R$ can also produce any message that would produce P in this context, and can therefore be transformed into a distinguisher against $Q\|P \cong_{\mathcal{O}} R\|P$. In other terms, $Q \cong_{\mathcal{O}} R$ and $\nu\overline{n}.P$ is $\mathcal{O}$-simulatable implies that $Q\|P \cong R\|P$.

5.3.1 Protocol Simulation

The goal in the rest of the Chapter is to use this notion of simulatability to obtain composability results. Suppose one wants to prove $P\|Q \cong P\|R$, knowing that $Q \cong_{\mathcal{O}} R$ and P is $\mathcal{O}$-simulatable. The way to obtain a distinguisher for $Q \cong_{\mathcal{O}} R$ from one on $P\|Q \cong P\|R$ is to "push" the (simulated version) of P within the distinguisher. A protocol P is then simulatable if there exists a simulator $\mathcal{A}^{\mathcal{O}}$ that can be "pushed " in any distinguisher $\mathcal{D}$. We formalize this construction below, where a protocol is simulatable if and only if any distinguisher $\mathcal{D}$ behaves in the same way if the protocol oracle $\mathcal{O}_P$ is replaced by its simulator $\mathcal{A}^{\mathcal{O}}$. We define formally $\mathcal{D}[\mathcal{A}^{\mathcal{O}}]^{\mathcal{O}}$ the replacement of $\mathcal{O}_P$ in $\mathcal{D}^{\mathcal{O},\mathcal{O}_P}$.

Definition 5.4. Given an oracle $\mathcal{O}$, a functional model $\mathcal{M}_f$, a protocol P, PTOMs $\mathcal{D}^{\mathcal{O},\mathcal{O}_P}(\rho_{r_{\mathcal{D}}}, 1^\eta)$ and $\mathcal{A}^{\mathcal{O}}(\cdots, 1^\eta)$, we define $\mathcal{D}[\mathcal{A}^{\mathcal{O}}]^{\mathcal{O}}(\rho_r, 1^\eta)$ as the PTOM that:

1. Splits its random tape ρ_r into ρ_{r_1}, ρ_{r_2}
2. Simulates $\mathcal{D}^{\mathcal{O},\mathcal{O}_P}(\rho_{r_2}, 1^\eta)$ by replacing every call to $\mathcal{O}_P$ with a computation of $\mathcal{A}^{\mathcal{O}}$: each time $\mathcal{D}$ enters a state corresponding to a call to $\mathcal{O}_P$, $\mathcal{D}[\mathcal{A}^{\mathcal{O}}]$ appends the query m to a history θ (initially empty), executes the subroutine $\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\rho_{r_1}, \theta, 1^\eta)$ and behaves as if the result of the subroutine was the oracle reply.
3. Prefixes each random handle of an oracle call of $\mathcal{D}$ with 0 and random handle of an oracle call of $\mathcal{A}$ with 1.
4. Outputs the final result of $\mathcal{D}$.

$\mathcal{D}[\mathcal{A}^{\mathcal{O}}]^{\mathcal{O}}$ must simulate $\mathcal{A}^{\mathcal{O}}$ and $\mathcal{D}$ so that they do not share randomness. To this end, $\mathcal{D}[\mathcal{A}^{\mathcal{O}}]^{\mathcal{O}}$ first splits its random tape ρ_r into ρ_{r_1} (playing the role of $\rho_{\mathcal{O}}$) and ρ_{r_2} (playing the role of $\rho_{\mathcal{D}}$). The oracle queries are prefixed by distinct handles for the same reason. $\mathcal{D}^{\mathcal{O},\mathcal{O}_P}$ has access to the shared secrets via both $\mathcal{O}$ and $\mathcal{O}_P$, while $\mathcal{D}[\mathcal{A}^{\mathcal{O}}]^{\mathcal{O}}$ only has access to them through the oracle $\mathcal{O}$. Remark that if $\mathcal{A}^{\mathcal{O}}$ and $\mathcal{D}^{\mathcal{O},\mathcal{O}_P}$ has a run-time polynomially bounded, so does $\mathcal{D}[\mathcal{A}^{\mathcal{O}}]^{\mathcal{O}}$.

To define the central notion of $\mathcal{O}$-simulatability, the distribution produced by any distinguisher interacting with the simulator must be the same as the distribution produced when it is interacting with the protocol. However, as we are considering a set of shared secrets $\overline{n}$ that might be used by other protocols, we need to ensure this equality of distributions for any fixed concrete value $\overline{v}$ of the shared secrets. Then, even if given access to other protocols using the shared secrets, no adversary may distinguish the protocol from its simulated version.

Definition 5.5. Given an oracle $\mathcal{O}$ with support $\overline{n}$, a functional model $\mathcal{M}^f$, a protocol P, a sequence of names $\overline{n}$, then, $\nu\overline{n}.P$ is *$\mathcal{O}$-simulatable* if and only if there exists a PTOM $\mathcal{A}_P^{\mathcal{O}}$ such that for every PTOM $\mathcal{D}^{\mathcal{O},\mathcal{O}_P}$, for every η, every $\overline{v} \in (\{0,1\}^\eta)^{|\overline{n}|}, c \in \{0,1\}^\star$,

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}\\ &\quad = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O}}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}\end{aligned}$$

Note that our definition of simulatability is a very strong one as it requires a perfect equality of distributions, as opposed to computational indistinguishability. This is intuitively what we want: $\mathcal{O}$-simulation expresses that P only uses the secrets in $\overline{n}$ as $\mathcal{O}$ does. This notion is not intended to capture any security property.

In practice, let us consider the security property $P\|Q \cong P\|Q'$, where P is simulatable by $\mathcal{A}_P^{\mathcal{O}}$. The idea of the later composition result is that an attacker $\mathcal{D}$ that distinguishes between $\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}$ and $\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_{Q'}}$ can be turned into an attacker that distinguishes between $\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O},\mathcal{O}_Q}$ and $\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O},\mathcal{O}_{Q'}}$. Notice that here, Q and P may share some secrets, and their distributions are not independent. The intuition is that Q is fixing a specific value for the shared name between P and Q, and P then needs to be simulatable for this fixed value. This is why the notion of simulatability asks that a protocol is simulatable for any fixed value of a set of secret names. The formalization of this proof technique is given by the following Proposition.

Proposition 5.1. *Given an oracle $\mathcal{O}$ with support $\overline{n}$, a functional model $\mathcal{M}_f$, protocols P, Q such that $\mathcal{N}(P) \cap \mathcal{N}(Q) \subseteq \overline{n}$, then, for any PTOM $\mathcal{A}_P^{\mathcal{O}}$, $\nu\overline{n}.P$ is $\mathcal{O}$-simulatable with $\mathcal{A}_P^{\mathcal{O}}$ if and only if for every PTOM $\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}$, for every η, every $\overline{v} \in (\{0,1\}^\eta)^{|\overline{n}|}, c \in \{0,1\}^\star$,*

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}\\ &\quad = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}\end{aligned}$$

It then implies that:

$$\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}(\rho_r, 1^\eta) = c\} = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = c\}$$

While this Definition intuitively captures the proof technique used to allow composition, it does not provide insight about how to prove the simulatability. Another equivalent definition states that a protocol is simulatable if there exists a simulator that can produce exactly the same distribution of messages as the protocol interacting with any attacker. We formalize in the following Technical Details this second Definition, and prove that the two Definitions are equivalent, which also yields the proof of Proposition 5.1.

Technical Details

For this second Definition of simulation to be realizable, we need to ensure that simulator's oracle calls and attacker's oracle calls use a disjoint set of random coins for the oracle randomness. We thus assume, w.l.o.g., that the random handles r of simulator's queries are prefixed by 1. This ensures that, as long as adversaries only make oracle calls prefixed by 0 (this can be assumed w.l.o.g. since it only constrains the part of the oracle's random tape where the randomness is drawn) the oracle randomness used by the simulator is not used by the adversary. We provide later in Example 5.4 a complete example illustrating both simulation and the need of the prefix and a formal definition of prefixed models.

Definition 5.6. Given a functional model $\mathcal{M}_f$, a sequence of names $\overline{n}$, an oracle $\mathcal{O}$ and a protocol P, we say that $\nu\overline{n}.P$ is $\mathcal{O}$-*simulatable* if the support of $\mathcal{O}$ is $\overline{n}$ and there is a PTOM $\mathcal{A}^{\mathcal{O}}$ (using random handles prefixed by 0) such that, for every $c \in \{0,1\}^\star$, for every $\overline{v} \in (\{0,1\}^\eta)^{|\overline{n}|}$, for every $m \geq 1$, for every PTOM $\mathcal{B}^{\mathcal{O}}$ (using random handles prefixed by 1),

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\rho_{r_1},\theta^1_m,1^\eta)=c \mid [\![\overline{n}]\!]^\eta_{\rho_s}=\overline{v}\}\\ =\;&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{O}_P(\rho_s,\theta^2_m)=c \mid [\![\overline{n}]\!]^\eta_{\rho_s}=\overline{v}\}\end{aligned}$$

where

$$\begin{aligned}\phi^2_{k+1}&=\phi^2_k,\mathcal{O}_P(\rho_s,\theta^2_k)\\ \phi^1_{k+1}&=\phi^1_k,\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\rho_{r_1},\theta^1_k,\eta)\\ \theta^i_{k+1}&=\theta^i_k,\mathcal{B}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\rho_{r_2},\eta,\phi^i_{k+1})\end{aligned}$$

for $0 \leq k < m$ and $\phi_0 = \emptyset$, $\theta_0 = \mathcal{B}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\rho_{r_2},\eta,\emptyset)$.

The machine $\mathcal{A}^{\mathcal{O}}$ can be seen as the simulator, while $\mathcal{B}$ is an adversary that computes the inputs: the definition states that there is a simulator, independently of the adversary. We asks for equality of distributions, between the sequence of messages θ^2, corresponding to the interactions of $\mathcal{B}^{\mathcal{O}}$ with $\mathcal{O}_P$, and the sequence of messages θ^1, corresponding to the interactions of $\mathcal{B}^{\mathcal{O}}$ with $\mathcal{A}^{\mathcal{O}}$.

Note that our definition of simulatability is a very strong one as it requires a perfect equality of distributions, as opposed to computational indistinguishability. This is intuitively what we want: $\mathcal{O}$-simulation expresses that P only uses the shared secrets as $\mathcal{O}$ does. This notion is not intended to capture any security property.

The two definitions are indeed equivalent. To prove this, a first technical Lemma is required. It shows that $\mathcal{O}$-simulation, whose definition implies the identical distributions of two messages produced either by the simulator or by the oracle, implies the equality of distributions of message sequences produced by either the oracle or the simulator. It is proved essentially via an induction on the length of the sequence of messages. For any sequence of names $\overline{n}$ and parameter η, we denote $D^\eta_{\overline{n}} = \{[\![\overline{n}]\!]^\eta_{\rho_s} | \rho_s \in \{0,1\}^\omega\}$ the set of possible interpretations of $\overline{n}$. We reuse the notations of Definition 5.6.

Lemma 5.2. *Given a functional model $\mathcal{M}^f$, a sequence of names $\overline{n}$, an oracle $\mathcal{O}$ with support $\overline{n}$ and a protocol P, that is* $\mathcal{O}$-simulatable *with $\mathcal{A}^{\mathcal{O}}$, we have, for every $\overline{x},\overline{y},c,r_2,r_{\mathcal{B}} \in \{0,1\}^\star$, every $\overline{v} \in D^\eta_{\overline{n}}$, for every $m \geq 1$, for every PTOM $\mathcal{B}^{\mathcal{O}}$ (using tags prefixed by 1):*

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^1_m=\overline{x},\phi^1_m=\overline{y}|\ [\![\overline{n}]\!]^\eta_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\ =\;&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^2_m=\overline{x},\phi^2_m=\overline{y}|\ [\![\overline{n}]\!]^\eta_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\end{aligned}$$

where we split $\rho_{\mathcal{O}}$ into $\rho^{\mathcal{A}}_{\mathcal{O}} \uplus \rho^{\mathcal{B}}_{\mathcal{O}}$ such that $\mathcal{O}$ called by $\mathcal{B}$ only accesses $\rho^{\mathcal{B}}_{\mathcal{O}}$ and $\mathcal{O}$ called by $\mathcal{A}$ only accesses $\rho^{\mathcal{A}}_{\mathcal{O}}$ (which is possible thanks to the distinct prefixes).

We now prove that Definition 5.6 implies Definition 5.4, i.e that the simulatability implies that we can replace a protocol oracle by its simulator.

Lemma 5.3. *Given an oracle $\mathcal{O}$ (with support $\overline{n}$), a functional model $\mathcal{M}_f$, a sequence of names $\overline{n}$, P, Q protocols, such that $\nu\overline{n}.P$ is $\mathcal{O}$-simulatable in the sense of Definition 5.6 with $\mathcal{A}_P^{\mathcal{O}}$ and $\mathcal{N}(P) \cap \mathcal{N}(Q) \subseteq \overline{n}$ then, for every PTOM $D^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}$ (prefixed by 1), every η, every $\overline{v} \in D_{\overline{n}}^{\eta}$ and every $c \in \{0,1\}^\star$,*

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}\} \\ &\quad = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}\}\end{aligned}$$

The idea is to use the definition of $\mathcal{O}$-simulatability, using a PTOM $\mathcal{B}^{\mathcal{O}}$ that behaves exactly as $\mathcal{D}$ when it computes the next oracle queries from the previous answers. The difficulty is that $\mathcal{D}$ may call the oracle $\mathcal{O}_Q$, while $\mathcal{B}$ has no access to this oracle. We know however that shared names are included in $\overline{n}$, whose sampling can be fixed at once (thanks to the definition of $\mathcal{O}$-simulation). The other randomness in Q can be drawn by $\mathcal{B}$ from ρ_r, without changing the distribution of $\mathcal{O}_Q$'s replies.

Proof. Fix η and the interpretation $[\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}$.

Given $\mathcal{D}$, we let $\mathcal{D}_m$ be the machine that behaves as $\mathcal{D}$, however halting after m calls to $\mathcal{O}_P$ (or when $\mathcal{D}$ halts if this occurs before the mth call) and returning the last query to $\mathcal{O}_P$.

We have that $\mathcal{D}_m$ first executes $\mathcal{D}_{m-1}$, then performs the oracle call $\mathcal{O}_P(\rho_s, \theta_{m-1})$, getting u_{m-1} and performs the computation of the next oracle call v_m (if $\mathcal{D}$ makes another oracle call), updates the history $\theta_m := (v_1, \ldots, v_m)$ and returns v_m if there is one or the output of $\mathcal{D}$ otherwise. $\mathcal{D}_m[\mathcal{A}_P^{\mathcal{O}}]$ first executes $\mathcal{D}_{m-1}[\mathcal{A}_P^{\mathcal{O}}]$, then performs the computation $\mathcal{A}_P^{\mathcal{O}}(\mathcal{M}_f, \rho_{r_1}, \theta'_{m-1})$ of u'_m, computes the next oracle call v'_m (if one is performed), updates $\theta'_m := (v'_1, \ldots, v'_m)$ and outputs either v_m of the output of $\mathcal{D}$.

We wish to use the definition of $\mathcal{O}$-simulation in order to conclude. However, we cannot directly use the $\mathcal{O}$-simulation, as $\mathcal{D}$ has access to an extra oracle $\mathcal{O}_Q$.

Part 1
We first prove that, assuming $\mathcal{A}_P^{\mathcal{O}}$ is a simulator of $\mathcal{O}_P$:

$$\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = c\} = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O}}(\rho_r, 1^\eta) = c\}$$

This is a straightforward consequence of Lemma 5.2. Writing respectively $p_1^1(c) = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = c\}$ and $p_1^2(c) = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O}}(\rho_r, 1^\eta) = c\}$, Using ρ_{r_1}, ρ_{r_2} as in Definition 5.4, we have

$$\begin{aligned}p_1^1(c) &= \sum_{r_{\mathcal{B}}, r_2} \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = c \mid ([\![\overline{n}]\!]_{\rho_s}^{\eta}, \rho_{\mathcal{O}}^{\mathcal{B}}, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\} \\ &\qquad \times \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{([\![\overline{n}]\!]_{\rho_s}^{\eta}, \rho_{\mathcal{O}}^{\mathcal{B}}, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\} \\ p_1^2(c) &= \sum_{r_{\mathcal{B}}, r_2} \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O}}(\rho_r, 1^\eta) = c \mid ([\![\overline{n}]\!]_{\rho_s}^{\eta}, \rho_{\mathcal{O}}^{\mathcal{B}}, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\} \\ &\qquad \times \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{[\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\end{aligned}$$

We let

$$p_2^1(r_{\mathcal{B}}, r_2, \overline{v}, c) = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = c \mid ([\![\overline{n}]\!]_{\rho_s}^{\eta}, \rho_{\mathcal{O}}^{\mathcal{B}}, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\}$$

and

$$p_2^2(r_{\mathcal{B}}, r_2, \bar{v}, c) = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O}}(\rho_r, 1^\eta) = c |\ ([\![\overline{n}]\!]_{\rho_s}^{\eta}, \rho_{\mathcal{O}}^{\mathcal{B}}, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\}$$

We use Definition 5.6 with $\mathcal{B}^{\mathcal{O}}(\rho_{r_2}, \eta, \phi)$ as the machine that simulates $\mathcal{D}_m$ for $m = |\phi|$ and using ϕ instead of querying the oracle. Let us define ϕ_m^i, θ_m^i for $i = 1, 2$ as in Definition 5.6. Note that with the definition of $\mathcal{D}$, $\mathcal{B}$ uses prefixes for oracle calls, disjoint from those used in $\mathcal{A}_P$, hence randomness used for oracle calls in $\mathcal{A}$ and $\mathcal{B}$ are disjoint. Let v_m^i be the last message of θ_m^i. By definition of $\mathcal{D}$ and $\mathcal{B}$ we have $v_m^1 = v_m$ and $v_m^2 = v'_m$. Choosing m such that $\mathcal{D}$ makes less than m oracle calls, we have

$$p_2^i(r_{\mathcal{B}}, r_2, \bar{v}, c) = \textstyle\sum_{\bar{x} \text{ s.t. } x_m = c, \bar{y}} \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta_m^i = \bar{x}, \phi_m^i = \bar{y} |\ ([\![\overline{n}]\!]_{\rho_s}^{\eta}, \rho_{\mathcal{O}}^{\mathcal{B}}, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\}.$$

Lemma 5.2 yields for all $r_{\mathcal{B}}, r_2, c$ that $p_2^2(r_{\mathcal{B}}, r_2, c) = p_2^1(r_{\mathcal{B}}, r_2, c)$, which concludes part 1.

Part 2
We now prove that:

$$\begin{array}{c} \forall \mathcal{D}.\ \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = c\} = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O}}(\rho_r, 1^\eta) = c\} \\ \Rightarrow \\ \forall \mathcal{D}.\ \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}(\rho_r, 1^\eta) = c\} = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = c\} \end{array} \quad (1)$$

We are thus going to show that, with the interpretation of $\overline{n}$ fixed, we can simulate $\mathcal{O}_Q$ in some $\mathcal{D}'$ by sampling in ρ_r instead of ρ_s. However, both computations of $\mathcal{O}_P$ and $\mathcal{O}_Q$ depend on ρ_s. This is where we need the assumptions that $\overline{n}$ contains the shared secrets between P and Q, as well as the splitting of ρ_r.

For any machine $\mathcal{M}^{\mathcal{O},\mathcal{O}_Q}$, we let $[\mathcal{M}]_{\overline{n}}^{\mathcal{O}}$ be the machine that executes $\mathcal{M}$, simulating $\mathcal{O}_Q$ for a fixed value $\overline{v}$ of $\overline{n}$. The machine samples the names appearing in Q and not in $\overline{n}$ and hard codes the interpretation of $\overline{n}$.

More precisely, we write $\mathcal{O}_Q(\rho_s, \theta) := \mathcal{O}_Q((\rho_{s_0}, \rho_{s_1}, \rho_{s_2}), \theta)$ where ρ_{s_0} is used for the sampling of $\overline{n}$, ρ_{s_1} for the sampling of other names in Q, and ρ_{s_2} for the reminder.

Then $[\mathcal{M}]_{\overline{n}}^{\mathcal{O}}(\rho_r, 1^\eta)$ is the machine that:

- Splits ρ_r into two infinite and disjoints ρ_{sQ}, ρ_{rM} and initializes an extra tape θ to zero.
- Simulates $\mathcal{M}(\rho_{rM}, 1^\eta)$ but every time $\mathcal{M}$ calls $\mathcal{O}_Q$ with input u, the machine adds u to θ, and produces the output of $\mathcal{O}_Q((\overline{v}, \rho_{rQ}, 0), \theta)$.

Such a machine runs in deterministic polynomial time (w.r.t. η). For any machine $\mathcal{M}^{\mathcal{O},\mathcal{O}_Q,\mathcal{O}_P}$, we similarly define $[\mathcal{M}]_{\overline{n}}^{\mathcal{O},\mathcal{O}_P}$. Now, we have that, for any c, by letting, for any X and U, $\mathbb{P}_X^{c,\overline{v}}(U) := \mathbb{P}_X\{U = c \mid [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}\}$:

$$\begin{array}{ll} & \mathbb{P}^{c,\overline{v}}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}(\mathcal{D}^{\mathcal{O},\mathcal{O}_P(\rho_{s_0},\rho_{s_1},\rho_{s_2}),\mathcal{O}_Q(\rho_{s_0},\rho_{s_1},\rho_{s_2})}(\rho_r, 1^\eta)) \\ =^1 & \mathbb{P}^{c,\overline{v}}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}(\mathcal{D}^{\mathcal{O},\mathcal{O}_P(\rho_{s_0},\rho_{s_1},\rho_{s_2}),\mathcal{O}_Q(\rho_{s_0},\rho_{s_1},0)}(\rho_r, 1^\eta)) \\ =^2 & \mathbb{P}^{c,\overline{v}}_{\rho_{s_1},\rho_{s_2},\rho_r,\rho_{\mathcal{O}}}(\mathcal{D}^{\mathcal{O},\mathcal{O}_P(\rho_{s_0},0,\rho_{s_2}),\mathcal{O}_Q(\rho_{s_0},\rho_{s_1},0)}(\rho_r, 1^\eta)) \\ =^3 & \mathbb{P}^{c,\overline{v}}_{\rho_{s_1},\rho_{s_2},\rho_r,\rho_{\mathcal{O}}}(\mathcal{D}^{\mathcal{O},\mathcal{O}_P(\overline{v},0,\rho_{s_2}),\mathcal{O}_Q(\overline{v},\rho_{s_1},0)}(\rho_r, 1^\eta)) \\ =^4 & \mathbb{P}^{c,\overline{v}}_{\rho_{sQ},\rho_s,\rho_{rD},\rho_{\mathcal{O}}}(\mathcal{D}^{\mathcal{O},\mathcal{O}_P(\overline{v},0,\rho_s),\mathcal{O}_Q(\overline{v},\rho_{sQ},0)}(\rho_r, 1^\eta)) \\ =^5 & \mathbb{P}^{c,\overline{v}}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}([\mathcal{D}]_{\overline{n}}^{\mathcal{O},\mathcal{O}_P(\rho_s)}(\rho_r, 1^\eta)) \text{ (ii)} \end{array}$$

Since

1. $\mathcal{O}_Q$ does not access ρ_{s_2}

2. $\mathcal{O}_P$ does not access ρ_{s_1}
3. We are sampling under the assumption that $[\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}$, i.e., ρ_{s_0} is equal to $\overline{v}$.
4. Renaming of tapes
5. By construction

And we also have similarly that, for any c:

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_r,\rho_\mathcal{O}}\{\mathcal{D}[\mathcal{A}_P^\mathcal{O}]^{\mathcal{O},\mathcal{O}_Q}(\rho_r,1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}\\ =\ &\mathbb{P}_{\rho_s,\rho_r,\rho_\mathcal{O}}\{[\mathcal{D}[\mathcal{A}_P^\mathcal{O}]]^\mathcal{O}_{\overline{v}}(\rho_r,1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}\ \text{(iii)}\end{aligned}$$

By applying the left-handside of (1) to $[\mathcal{D}]^{\mathcal{O},\mathcal{O}_P(\rho_s)}_{\overline{n}}(\rho_r,1^\eta)$ and $[\mathcal{D}[\mathcal{A}_P^\mathcal{O}]_j]^\mathcal{O}_{\overline{v}}(\rho_r,1^\eta)$, and using (ii) and (iii), we can conclude by transitivity. We conclude the proof of the lemma by putting Part 1 and Part 2 together.

■

We now prove the converse direction.

Lemma 5.4. *Given an oracle $\mathcal{O}$ with support $\overline{n}$, a functional model $\mathcal{M}^f$, protocols P, Q such that $\mathcal{N}(P)\cap\mathcal{N}(Q) \subset \overline{n}$, if there is a PTOM $\mathcal{A}_P^\mathcal{O}$ such that, for every PTOM $\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}$, for every η, every $\overline{v} \in D^\eta_{\overline{n}}$ and every $c \in \{0,1\}^\star$,*

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_r,\rho_\mathcal{O}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}(\rho_r,1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}\\ &\quad= \mathbb{P}_{\rho_s,\rho_r,\rho_\mathcal{O}}\{\mathcal{D}[\mathcal{A}_P^\mathcal{O}]^{\mathcal{O},\mathcal{O}_Q}(\rho_r,1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}\end{aligned}$$

then $\nu\overline{n}.P$ is $\mathcal{O}$-simulatable.

Proof. Let $\mathcal{B}$ be a PTOM, η, an interpretation $\overline{v} \in D^\eta_{\overline{n}}$ and $m \in \mathcal{N}$, we must prove that the output distribution of B will be the same whether it interacts m-th time with $\mathcal{A}_P^\mathcal{O}$ or $\mathcal{O}_P$. We define $\mathcal{D}$ as follows. For $i := 0$ to $m-1$, $\mathcal{D}$ computes $w_i := \mathcal{B}(\alpha_1,\ldots,\alpha_i)$. Then $\mathcal{D}$ calls $\mathcal{O}_P$ with w_i and let α_{i+1} be the reply. $\mathcal{D}$ finally outputs α_m. We denote by w'_i and α'_i the corresponding values for $\mathcal{D}[\mathcal{A}_P^\mathcal{O}]^{\mathcal{O},\mathcal{O}_Q}$

Let us denote

$$\begin{aligned}\phi^2_{k+1} &= \phi^2_k, \mathcal{O}_P(\rho_s,\theta^2_k)\\ \phi^1_{k+1} &= \phi^1_k, \mathcal{A}^{\mathcal{O}(\rho_s,\rho_\mathcal{O})}(\mathcal{M}_f,\rho_{r_1},\theta^1_k,\eta)\\ \theta^i_{k+1} &= \theta^i_k, \mathcal{B}^{\mathcal{O}(\rho_s,\rho_\mathcal{O})}(\mathcal{M}_f,\rho_{r_2},\eta,\phi^i_k)\end{aligned}$$

for $0 \le k < m$ and $\phi_0 = \theta_0 = \emptyset$.

We have by construction of $\mathcal{D}$ for any c:

$$\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_\mathcal{O}}\{w_m = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} = \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_\mathcal{O}}\{\mathcal{O}_P(\rho_s,\theta^2_m) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}$$

and

$$\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_\mathcal{O}}\{w'_m = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} = \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_\mathcal{O}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_\mathcal{O})}(\mathcal{M}_f,\rho_{r_1},\theta^1_m,\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}$$

The hypothesis gives us that :

$$\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_\mathcal{O}}\{w_m = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} = \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_\mathcal{O}}\{w'_m = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}$$

So we conclude that:

$$\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\mathcal{M}_f,\rho_{r_1},\theta^1_m,\eta)=c \mid [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}\}$$
$$=\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{O}_P(\rho_s,\theta^2_m)=c \mid [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}\}$$

■

We can finally conclude, as Lemmas 5.3 and 5.4 directly yields that Definition 5.6 is equivalent to Definition 5.6 simply by taking Q as the empty protocol.

Example 5.3. We fix first $\mathcal{M}_f$ (in an arbitrary way). We consider the following handshake protocol, in which n, r, k, r' are names:

$$\begin{array}{rll} & A := & \textbf{in}\ (c_A, x_0).\textbf{out}(c_A, \mathtt{enc}(n,r,k)).\ \textbf{in}\ (c_A,x). \\ & & \quad \textbf{if}\ \mathtt{dec}(x,k)=\langle n,1\rangle\ \textbf{then out}(c_A,\mathtt{ok}) \\ \| & B := & \textbf{in}\ (c_B,y).\textbf{out}(c_B,\mathtt{enc}(\langle \mathtt{dec}(y,k),1\rangle, r',k)) \end{array}$$

We consider the oracle $\mathcal{O}_k^{\mathtt{enc},\mathtt{dec}}$ that, when receiving $\langle t, m\rangle$ as input, answers $enc(m, r_o, k)$ if $t =$ `"enc"`, and $dec(m, l)$ if $t =$ `"dec"` (the oracle actually also expects an handle for the secret key and a tag to specify where to sample r_o). We can easily prove that $\nu k.A$ is $\mathcal{O}_k^{\mathtt{enc},\mathtt{dec}}$-simulatable, as the attacker can sample an arbitrary n', use the oracle to compute $\mathtt{enc}(n', r_o, k)$ (which has the same distribution as $\mathtt{enc}(n', r, k)$ for any fixed value of k) with the request $\langle$`"enc"`$, n\rangle$, and $\mathtt{dec}(x, k)$ with the request $\langle$`"dec"`$, x\rangle$.

Intuitively, the shared secret k is only used in A in ways that are directly simulatable with the oracle, and A is thus $\mathcal{O}$-simulatable.

Thanks to the more intuitive Definition of simulatability (cf. Definition 5.6 for details), proving simulatability is in practice a syntactic verification. With $\mathcal{O}_k^{\mathtt{enc},\mathtt{dec}}$ from the previous example, $\nu k.P$ is $\mathcal{O}$-simulatable for any P where all occurrences of k occurs at key position, and all encryptions use fresh randoms.

Technical Details

Let us explain why the previous examples illustrate the need for prefixed models.

Example 5.4. We take a more formal view on Example 5.3.

Let $\mathcal{O}$ be the encryption-decryption oracle: it expects an input $\langle$`"dec"`$, m\rangle$ or $\langle$`"enc"`$, m\rangle$, a key $s = 1$ (only one encryption key is considered), an input tag t and a security parameter η and returns

- $\mathtt{enc}(m, r, k)$ if the query is prefixed by `"enc"`, k is the secret value extracted from ρ_s corresponding to the key 1, r is drawn from $\rho_{\mathcal{O}}$ and associated with the tag t (via e_1).
- $\mathtt{dec}(m, k)$ if the query is prefixed by `"dec"`, k is the secret value extracted from ρ_s corresponding to the key 1
- an error message otherwise (either the primitives fail or the query does not have the expected format).

The goal is to show that $\nu k.A$ is $\mathcal{O}$-simulatable. (So, here, B is useless, and we let P be A).

$\mathcal{O}_P$ is then defined as follows (according to the Section 2.3.2):

- On input w_1, with an empty history, it outputs $[\![\mathtt{enc}(n,r,k)]\!]^{\eta}_{\rho_s}$ and writes w_1 on the history tape.
- On input w_2 with a non empty history tape, it outputs $\mathtt{ok}$ if $[\![\mathtt{dec}(x,k)]\!]^{\eta,x\mapsto w_2}_{\rho_s} = [\![\langle n,1\rangle]\!]^{\eta}_{\rho_s}$ and an error otherwise.

The machine $\mathcal{A}^{\mathcal{O}}(\rho_{r_1},\theta,\eta)$ is then defined as follows:

- If $\theta = \{m_1\}$
  1. $\mathcal{A}$ draws α (for the value of n) from ρ_{r_1} and draws t from ρ_{r_1}
  2. calls $\mathcal{O}$ with $(\langle \mathtt{"enc"}, \alpha\rangle, 1, t)$ and gets back the bitstring $[\![\mathtt{enc}(n,r,z)]\!]^{\eta,z\mapsto [\![k]\!]^{\eta}_{\rho_s}}_{\rho_{r_1},\rho_{\mathcal{O}}}$. The interpretation of k is indeed fixed at once since it belongs to the "shared" names bounded by ν.
  3. outputs $[\![\mathtt{enc}(x,r,z)]\!]^{\eta,x\mapsto\alpha,z\mapsto[\![k]\!]^{\eta}_{\rho_s}}_{\rho_{r_1}}$
- If $\theta = (m_1, m_2)$,
  1. calls $\mathcal{O}$ with $(\langle \mathtt{"dec"}, m_2\rangle, 1, -)$ and gets back the bitstring $w = [\![\mathtt{dec}(y,z)]\!]^{y\mapsto m_2, z\mapsto[\![k]\!]^{\eta}_{\rho_s}}$ or an error message.
  2. checks whether $w = [\![\langle n,1\rangle]\!]^{\eta}_{\rho_{r_1}}$. If it is the case, then outputs $\mathtt{ok}$.

Now, consider an arbitrary PTOM $\mathcal{B}^{\mathcal{O}}$.

- $\phi^1_1 = [\![\mathtt{enc}(n,x,k)]\!]^{\eta,x\mapsto s_1}_{\rho_{r_1}}$ where s_1 is the randomness used by $\mathcal{O}$ when queried with $[\![t]\!]_{\rho_{r_1}}$ (note: we will see that it does matter to be very precise here; we cannot simply claim that the value of x is just a randomness drawn by $\mathcal{O}$).
- $\phi^2_1 = [\![\mathtt{enc}(n,r,k)]\!]^{\eta}_{\rho_s}$
- $\theta^1_i = w_i$, an arbitrary bitstring, computed by $\mathcal{B}^{\mathcal{O}}$ using the oracle $\mathcal{O}$, ϕ^1_i and the random tape ρ_{r_2}.
- $\phi^1_2 = \phi^1_1, \mathtt{ok}$ if $[\![\mathtt{dec}(y,z)]\!]^{y\mapsto w_1, z\mapsto[\![k]\!]^{\eta}_{\rho_s}} = [\![\langle n,1\rangle]\!]^{\eta}_{\rho_{r_1}}$ and an error otherwise
- $\phi^2_2 = \phi^2_1, \mathtt{ok}$ if $[\![\mathtt{dec}(x,k)]\!]^{\eta,x\mapsto w_2}_{\rho_s} = [\![\langle n,1\rangle]\!]^{\eta}_{\rho_s}$ and an error otherwise

$\mathcal{A}$ $\mathcal{O}$-simulates $\nu k.P$ iff, for every $v = [\![k]\!]_{\rho_s}$,

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r1},\rho_{r_2},\rho_{\mathcal{O}}}\{[\![\mathtt{dec}(y,z)]\!]^{y\mapsto w_1,z\mapsto v} = [\![\langle n,1\rangle]\!]^{\eta}_{\rho_{r_1}}\}\\ &\quad = \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r2},\rho_{\mathcal{O}}}\{[\![\mathtt{dec}(x,k)]\!]^{\eta,x\mapsto w_2}_{\rho_s} = [\![\langle n,1\rangle]\!]^{\eta}_{\rho_s}\}\end{aligned}$$

First, the distributions of ϕ^1_1 and ϕ^2_1 are identical. ϕ^1_1 depends on ρ_{r_1} and $\rho_{\mathcal{O}}$, while ϕ^2_1 depends on ρ_s only. The distributions of $\phi^1_1, [\![\langle n,1\rangle]\!]_{\rho_{r_1}}$ and $\phi^2_1, [\![\langle n,1\rangle]\!]_{\rho_s}$ are also identical.

Now the distributions $w_1 = \mathcal{B}^{\mathcal{O}}(\phi^1_1,\rho_{r_2}), [\![\langle n,1\rangle]\!]_{\rho_{r_1}}$ and $w_2 = \mathcal{B}^{\mathcal{O}}(\phi^1_2,\rho_{r_2}), [\![\langle n,1\rangle]\!]_{\rho_s}$ are equal *if the randomness used by $\mathcal{B}$ are disjoint from the random coins used in ϕ^1_1, ϕ^2_1*. This is why there is an assumption that ρ_{r_1} and ρ_{r_2} are disjoint and why *it should be the case* that the random coins used in the oracle queries of $\mathcal{B}$ are distinct from the ones used in the oracle queries of $\mathcal{A}$. This can be ensured by the disjointness of tags used by $\mathcal{A}$ and $\mathcal{B}$ respectively.

With these assumptions, we get the identity of the distributions of $\mathtt{dec}(w_1,v), [\![\langle n,1\rangle]\!]_{\rho_s}$ and $\mathtt{dec}(w_2,v), [\![\langle n,1\rangle]\!]_{\rho_s}$, hence the desired result.

Without these assumptions (for instance non-disjointness of tags used by $\mathcal{B}$, $\mathcal{A}$), $\mathcal{B}$ can query $\mathcal{O}$ with a random input and a random tag, say n', t'. As above, we let s_1 be the random value drawn by $\mathcal{O}$ corresponding to the tag t'. Then $\mathbb{P}\{[\![n]\!]_{\rho_s} = n' \wedge [\![r]\!]_{\rho_s} = s_1\} = \frac{1}{2^{2\eta}}$ while

$$\begin{aligned}\mathbb{P}\{[\![n]\!]_{\rho_{r_1}} = n' \wedge [\![r]\!]_{\rho_{r_1}} = s_1\} &= \tfrac{1}{2^{\eta}}\mathbb{P}\{[\![t]\!]_{\rho_{r_1}} = [\![t']\!]_{\rho_{r_2}} \vee ([\![t]\!]_{\rho_{r_1}} \neq [\![t']\!]_{\rho_{r_2}} \wedge [\![r]\!]_{\rho_{r_1}} = [\![r']\!]_{\rho_{\mathcal{O}}})\}\\ &= \tfrac{1}{2^{\eta}} \times (\tfrac{1}{2^{\eta}} + \tfrac{2^{\eta}-1}{2^{\eta}} \times \tfrac{1}{2^{\eta}})\\ &= \tfrac{1}{2^{2\eta}}(2 - \tfrac{1}{2^{\eta}})\end{aligned}$$

In other words, the collision is more likely to occur since it can result from either a collision in the tags or a collision in the randomness corresponding to different tags.

As demonstrated in the previous example, it is necessary to assume that oracle randomness used by the simulator queries and the attacker queries are disjoint. The simplest way of ensuring this is to force all tags of oracle calls to be prefixed. We show here that this assumption can be made without loss of generality.

Definition 5.7. Given a PTOM $\mathcal{A}^{\mathcal{O}}$ and a constant c. We define $\mathcal{A}^{\mathcal{O}}_{pref-c}$ as a copy of $\mathcal{A}$, except that all calls to the oracle of the form $\overline{w}, r, s$ are replaced with calls of the form $\overline{w}, c \cdot r, s$, where the $\cdot$ denotes the concatenation of bitstrings.

The following lemma shows that we can, w.l.o.g., consider models, in which the tags are prefixed.

Lemma 5.5. *For any non-empty constant c and any PTOM $\mathcal{A}^{\mathcal{O}}$, we haves*

$$\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\rho_r, 1^\eta) = 1\} = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}_{pref-c}(\rho_r, 1^\eta) = 1\}$$

Proof. We fix a constant c, for any oracle $\mathcal{O}$ (with functions n, e_1, e_2), we define $\mathcal{O}_{pref-c}$ (with mapping function n', e_1', e_2') the copy of $\mathcal{O}$ such that:

$$n'(w, s, r) = n(w, s, c|r)$$

n is injective by definition, so n' is injective too. For any $v \in \{0,1\}^\eta$, as all extractions of e_1 are unique for each value of n and their length only depends on η, we have for any w, r, s

$$\mathbb{P}_{\rho_{\mathcal{O}}}\{e_1(n(w, s, r), \eta, \rho_{\mathcal{O}}) = v\} = \mathbb{P}_{\rho_{\mathcal{O}}}\{e_1'(n'(w, s, r), \eta, \rho_{\mathcal{O}}) = v\}$$

This implies that for any input, $\mathcal{O}$ and $\mathcal{O}_{pref-c}$ will produce the same output distribution. So $\mathcal{A}^{\mathcal{O}}$ and $\mathcal{A}^{\mathcal{O}_{pref-c}}$ will produce the same distributions for any input. We conclude by remarking that $\mathcal{A}^{\mathcal{O}_{pref-c}}$ and $\mathcal{A}^{\mathcal{O}}_{pref-c}$ behaves the same by construction. ■

An immediate consequence of this Lemma is that for all indistinguishability results, we can, w.l.o.g., constrain attackers to only use prefixed oracle calls.

In particular it implies equivalence between indistinguishability in a computational model and indistinguishability for prefixed distinguishers in the prefixed computational model.

Thanks to the previous Definitions, simulatability is stable under composition operators. This is an important feature of the notion of simulatability, as it allows to reduce the simulation of large processes to the simulation of simpler processes.

Theorem 5.1. *Given an oracle $\mathcal{O}$, protocols P, Q, and $\overline{n} = \mathcal{N}(P) \cap \mathcal{N}(Q)$, if*

- *$\nu\overline{n}.P$ is $\mathcal{O}$-simulatable*
- *$\nu\overline{n}.Q$ is $\mathcal{O}$-simulatable*

Then $\nu\overline{n}.P \| Q$ and $\nu\overline{n}.P; Q$ are $\mathcal{O}$-simulatable.

Proof. Let $\mathcal{D}$ be an arbitrary PTOM. By Lemma 5.3, there is a machine $\mathcal{A}_P^{\mathcal{O}}$ s.t.

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}(\rho_r,1^\eta)=c \mid [\![\overline{n}]\!]^\eta_{\rho_s}=\overline{v}\}\\ &\quad=\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}]^{\mathcal{O},\mathcal{O}_Q}(\rho_r,1^\eta)=c \mid [\![\overline{n}]\!]^\eta_{\rho_s}=\overline{v}\}\end{aligned}$$

Applying once more the Lemma 5.3, there is a machine $\mathcal{A}_Q^{\mathcal{O}}$ s. t., for every $c \in \{0,1\}^\star$,

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}^{\mathcal{O},\mathcal{O}_P,\mathcal{O}_Q}(\rho_r,1^\eta)=c \mid [\![\overline{n}]\!]^\eta_{\rho_s}=\overline{v}\}\\ &\quad=\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}][\mathcal{A}_Q^{\mathcal{O}}]^{\mathcal{O}}(\rho_r,1^\eta)=c \mid [\![\overline{n}]\!]^\eta_{\rho_s}=\overline{v}\}\end{aligned}$$

We define $\mathcal{A}^{\mathcal{O}}_{P\|Q}(\mathcal{M}^f,\rho_{r_1},\theta,1^\eta,m)$ as the machine that behaves as $\mathcal{A}_P^{\mathcal{O}}(\mathcal{M}^f,\rho_{r_1,P},\theta_P,1^\eta,m)$ (resp. $\mathcal{A}_Q^{\mathcal{O}}(\mathcal{M}^f,\rho_{r_1,Q},\theta_Q,m)$) if m is a message supposed to be handled by P (resp. by Q) (use of action determinism) Then the result is appended to θ_P (resp. θ_Q). This assumes (this is an invariant) that θ can be split into θ_P and θ_Q.

We note that $\mathcal{D}[\mathcal{A}_P^{\mathcal{O}}][\mathcal{A}_Q^{\mathcal{O}}]^{\mathcal{O}} = \mathcal{D}[\mathcal{A}^{\mathcal{O}}_{P\|Q}]^{\mathcal{O}}$. Then we use Lemma 5.4 to conclude. ■

Q Technical Details

Alternative notions of simulatability We discuss here some variation on our notion of simulatability. First, let us note that our notion of simulatability assumes that models are prefixed. As demonstrated previously this is necessary in order to get an achievable notion of simulatability. We will therefore not consider models that are not prefixed. We may consider variants of simulatability, depending on the order of the quantifiers and sharing of randomness between simulator and distinguisher. We define simulatability as the existence of a simulator that works for all distinguishers. In other words our ordering of quantifier is:

$$\exists\mathcal{A}^{\mathcal{O}}(\rho_{r_1})\forall\mathcal{D}(\rho_{r_2})$$

In a prefixed model, we believe that switching the quantifiers lead to the same notion:

$$\exists\mathcal{A}^{\mathcal{O}}(\rho_{r_1})\forall\mathcal{D}(\rho_{r_2}) \Leftrightarrow \forall\mathcal{D}(\rho_{r_2})\exists\mathcal{A}^{\mathcal{O}}(\rho_{r_1})$$

We provide no proof, but the intuition is that there exists a "universal" distinguisher, namely the PTOM $\mathcal{D}$, which performs any possible queries with uniform probability. Now, considering any other distinguisher $\mathcal{D}'$, as the simulator $\mathcal{A}^{\mathcal{O}}$ for $\mathcal{D}$ has to provide the exact same distribution as the protocol for each query of $\mathcal{D}$, as $\mathcal{D}$ performs all possible queries (with very small probability), $\mathcal{A}^{\mathcal{O}}$ will also be a correct simulator for $\mathcal{D}'$.

Another alternative is to allow the simulator and the distinguisher to share the same randomness. Then, $\exists\mathcal{A}^{\mathcal{O}}(\rho_r)\forall\mathcal{D}(\rho_r)$ seems to provide an unachievable definition. Indeed, if the simulator is not allowed to use private randomness while the protocol is, the simulator cannot mimic the probabilistic behavior of the protocol.

The last possibility however seems to offer an alternative definition for simulatability:

$$\forall\mathcal{D}(\rho_r)\exists\mathcal{A}^{\mathcal{O}}(\rho_r)$$

This seems to be a weaker definition than ours as the choices of the simulator can depend on the ones of the distinguisher. It may simplify (slightly) the proofs for the main theorem, but it would create issues for the unbounded replication as it would break uniformity of reductions (since the runtime of the simulator may now depend on the environment it is running in).

5.3.2 Generic Oracles for Tagged Protocols

Section Summary

In order for our definition of simulatability to be useful, the design of oracles is a key point. They need to be:

1. generic/simple, yet powerful enough so that protocols can be easily shown to be simulatable,
2. restrictive enough so that proving protocols in the presence of oracles is doable.

We provide here with examples of such oracles, namely generic tagged oracles for signature, that will be parameterized by arbitrary functions, together with security properties that are still true in the presence of tagged oracles.

In practice, protocols that use some shared secrets use tags, for instance string prefixes, to ensure that messages meant for one of the protocol cannot be confused with messages meant for the other one. These tags can ensure what is called "domain separation" of the two protocols, ensuring that the messages obtained from one cannot interfere with the security of the second protocol. These tags can be explicit, for instance by adding a fixed constant to the messages, or implicit, where each message of a protocol depend on some fresh randomness that can be used to define some kind of session identifier.

We define generic oracles for decryption and signatures, parameterized by an abstract tagging function T and a secret key sk, that allow to perform a cryptographic operation with the key sk, on any message m satisfying $T(m)$. T can then simply check the presence of a prefix, or realize some implicit tagging, checking that the message depends on the randomness used by a specific session.

After defining those generic oracles, we define generic axioms, parameterized by T, that allow to perform proofs against attackers with access to the oracle. The generic axiom for signatures (or any other primitive) are implied by the classical cryptographic axioms.

We see tagging as a boolean function T computable in polynomial time over the interpretation of messages. For instance, if the messages of protocol P are all prefixed with the identifier id_P, T is expressed as $T(m) := \exists x.m = \langle id_P, x\rangle$. In a real life protocol, id_P could for instance contain the name and version of the protocol.

Intuitively tagged oracles produce the signature of any properly tagged message and allow to simulate P.

With these oracles, an immediate consequence of the composition Theorems found in Section 5.4 is the classical result that if two protocols tag their messages differently, they can be safely composed [ACD12]. Note that as our tag checking function is an arbitrary boolean function: tagging can be implicit, as illustrated in our applications in Section 5.6.

As an example, we provide two oracles, one for encryption and one for signing, that allow to simulate any protocol that only produces messages that are well tagged for T.

Definition 5.8. Given a name sk and a tagging function T, we define the generic signing

oracle $\mathcal{O}^{sign}_{T,sk}$ and the generic decryption oracle $\mathcal{O}^{dec}_{T,sk}$ as follows:

$$\begin{aligned}\mathcal{O}^{\mathtt{sign}}_{T,sk}(m) := \quad & \textbf{if } T(m) \textbf{ then} \\ & \quad \text{output}(\mathtt{sign}(m, sk)) \\ \mathcal{O}^{\mathtt{dec}}_{T,sk}(m) := \quad & \textbf{if } T(\mathtt{dec}(m, sk)) \textbf{ then} \\ & \quad \text{output}(\mathtt{dec}(m, sk))\end{aligned}$$

Any well-tagged protocol according to T, i.e., a protocol that only decrypts or signs well tagged messages, will be simulatable using the previous oracles. Hence we meet the goal 1 stated at the beginning of this section, as this can be checked syntactically on a protocol. We provide, as an example, the conditions for a tagged signature.

Example 5.5. Any protocol P whose signatures are all of the form **if** $T(t)$ **then** $\mathtt{sign}(t, sk)$ for some term t (that does not use sk) is immediately $\nu sk.P$ $\mathcal{O}^{\mathtt{sign}}_{T,sk}$-simulatable. Indeed, informally, all internal values of the protocol except sk can be picked by the simulator from its own randomness, while all terms using sk can be obtained by calls to the tagged signing oracle, as all signed terms in P are correctly tagged. Let us emphasize that the simulation holds for any specific value of sk, as the distribution of outputs is the same, whether it is the simulator that draws the internal names of P, except sk, or P itself.

As we need to perform cryptographic proofs in the presence of oracles, it is useful to define security properties that cannot be broken by attackers with access to these oracles (without having to consider the specific calls made to these oracles). The games defining these properties slightly differ from the classical security games. Consider the example of signatures and the usual EUF-CMA game. If the attacker is, in addition, equipped with an oracle $\mathcal{O}$ that signs tagged messages, they immediately win the EUF-CMA game, "forging" a signature by a simple call to $\mathcal{O}$. We thus define a tagged unforgeability game (EUF-CMA$_{T,sk}$), derived from the EUF-CMA game presented in Definition 2.13, where the adversary wins the game only if they are able to produce the signature of a message that is not tagged.

Definition 5.9. A signature scheme $(\mathsf{Sign}, \mathsf{Vrfy})$ is EUF-CMA$_{T,sk}$ secure for oracle $\mathcal{O}$ and interpretation of keys $\mathcal{A}_{sk}$ if, for any PTOM $\mathcal{A}$, the game described in Figure 5.1 returns true with probability (over $\rho_r, \rho_s, \rho_{\mathcal{O}}$) negligible in η.

Game EUF-CMA$^{\Sigma,\mathcal{A}}_{T,sk}(\eta, \rho_r, \rho_s, \rho_{\mathcal{O}})$:	**Oracle** $\mathsf{Sign}(\mathsf{m})$:
$\mathsf{List} \leftarrow []$	$\mathsf{List} \leftarrow (\mathsf{m} : \mathsf{List})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow (\llbracket pk \rrbracket_{\rho_s}, \llbracket sk \rrbracket_{\rho_s})$	$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$
$(\mathsf{m}, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}}),\mathsf{Sign}}(\mathsf{pk}, \eta, \rho_r)$	Return σ
Return $\neg T(\mathsf{m}) \wedge \mathsf{Vrfy}(\mathsf{pk}, \mathsf{m}, \sigma) \wedge \mathsf{m} \notin \mathsf{List}$	

Figure 5.1: Game for Tagged Unforgeability (EUF-CMA$_{T,sk}$)

The main goal of the previous definition is to allow us to prove protocols in the presence of oracles (hence composed with simulated ones), reaching the goal 2 stated at the beginning of the section.

More precisely, one can, for instance, simply design a classical game based proof, reducing the security of the protocol to the security of the EUF-CMA$_{T,sk}$ game rather than the classical EUF-CMA game. This reasoning is valid as EUF-CMA implies EUF-CMA$_{T,sk}$ even in the presence of the corresponding oracle.

Proposition 5.2. *If a signature scheme* $(\mathsf{Sign}, \mathsf{Vrfy})$ *is* EUF-CMA *secure for keys given by* $\mathcal{A}_{sk}$*, then* $(\mathsf{Sign}, \mathsf{Vrfy})$ *is* EUF-CMA$_{T,sk}$ *secure for the oracle* $\mathcal{O}^{sign}_{T,sk}$ *and the interpretation of keys* $\mathcal{A}_{sk}$*.*

Remark that the base assumptions made about the cryptographic primitives are classical ones, and thus the final proof of the composed protocol only depends on some classical cryptographic hypotheses.

5.4 Main Composition Theorems

Section Summary

We distinguish between two complementary cases. First, Theorem 5.2 covers protocols composed in a way where they do not share states besides the shared secrets (e.g., parallel composition of different protocols using the same master secret key). Second, Theorem 5.4 covers protocols passing states from one to the other (e.g., a key exchange passing an ephemeral key to a secure channel protocol). We finally extend these composition results to self-composition, i.e., proving the security of multiple sessions from the security of a single one or the security of a protocol lopping on itself, for instance a key renewal protocol.

5.4.1 Composition without State Passing

Essentially, if two protocols P, Q are indistinguishable, they are still indistinguishable when running in any simulatable context. The context must be simulatable for any fixed values of the shared names of P, Q and the context. The context can contain parallel or sequential composition as illustrated by the following example.

Example 5.6. Let P, Q, R, S be protocols and $\mathcal{O}$ an oracle. Let $\overline{n} = \mathcal{N}(P\|Q) \cap \mathcal{N}(R\|S)$. If $P \cong_{\mathcal{O}} Q$ and $\nu\overline{n}.R\|S$ is $\mathcal{O}$-simulatable, then some applications of Theorem 5.2 can yield

1. $P\|R \cong_{\mathcal{O}} Q\|R$
2. $R; P \cong_{\mathcal{O}} R; Q$
3. $(R; P)\|S \cong_{\mathcal{O}} (R; Q)\|S$

We generalize the previous example to any simulatable context and to n protocols. For any integer n, we denote by $C[__1, \ldots, __n]$ a *context*, i.e., a protocol built using the syntax of Figure 2.3 and distinct symbols $__i$, viewed as elementary processes. $C[P_1, \ldots, P_n]$ is the protocol in which each hole $_i$ is replaced with P_i.

Example 5.7. In the three examples of Example 5.6, in order to apply the next theorem, we respectively use as contexts

- $C[__1] := __1\|R$
- $C[__1] := R; __1$
- $C[__1] := (R; __1)\|S$.

In this first Theorem, no values (e.g., ephemeral keys) are passed from the context to the protocols. In particular, the protocols do not have free variables which may be bound by the context.

Theorem 5.2. *Given a functional model $\mathcal{M}_f$ and an oracle $\mathcal{O}$, let $P_1, \ldots, P_n$, $Q_1, \ldots, Q_n$ be protocols and $C[__1, \ldots, __n]$ be a context such that all their channels are disjoint, 0 some constant, $\overline{n}$ a sequence of names and $c_1, \ldots, c_n$ fresh channel names. If*

1. $\mathcal{N}(C) \cap \mathcal{N}(P_1, \ldots, P_n, Q_1, \ldots, Q_n) \subseteq \overline{n}$
2. $\nu\overline{n}.C[\mathbf{out}(c_1, 0), \ldots, \mathbf{out}(c_n, 0)]$ *is $\mathcal{O}$-simulatable*
3. $P_1\| \ldots \|P_n \cong_{\mathcal{O}} Q_1\| \ldots \|Q_n$

Then

$$C[P_1, \ldots, P_n] \cong_{\mathcal{O}} C[Q_1, \ldots, Q_n]$$

Specifically[a]*, there exists a polynomial* p_S *(independent of* C*) such that, if* p_C *is the polynomial bound on the runtime of the simulator for* C*, we have,*

$$\mathrm{Adv}^{C[P_1,\ldots,P_n] \cong_{\mathcal{O}} C[Q_1,\ldots,Q_n]}(t)$$

$$\leq \mathrm{Adv}^{P_1 \| \ldots \| P_n \cong_{\mathcal{O}} Q_1 \| \ldots \| Q_n}\left(p_S\left(t, n, |C|, p_C(t)\right)\right)$$

[a] We provide, in this Theorem and the following ones, explicit advantages, as our constructions do not directly allow for unbounded replication. This will later be used to ensure that the advantage of the adversary only grows polynomially with respect to the number of sessions.

Note that the bound we obtain for the reduction is polynomial in the running time of the context. We denote by $\overline{C}$ the protocol C in which each $_i$ is replaced with $\mathbf{out}(c_i, 0).\mathbf{0}$, where c_i is a channel name and 0 is a public value. Intuitively, $\overline{C}$ abstracts out the components P_i, only revealing which P_i is running at any time. The intuition behind the proof of the Theorem is then as follows. First, we show that $\overline{C}\|P_1\| \ldots \|P_n \cong_{\mathcal{O}} \overline{C}\|Q_1\| \ldots \|Q_n$ implies $C[P_1, \ldots, P_n] \cong_{\mathcal{O}} C[Q_1, \ldots, Q_n]$. This is done by a reduction, where we mainly have to handle the scheduling, which is possible thanks to the information leaked by $\overline{C}$, and the action determinism of the protocols. In a sense, this means that indistinguishability for protocols in parallel implies indistinguishability for any scheduling of those protocols. Secondly, by simulating $\overline{C}$ thanks to Proposition 5.1, the two hypothesis of the Theorem imply $\overline{C}\|P_1\| \ldots \|P_n \cong_{\mathcal{O}} \overline{C}\|Q_1\| \ldots \|Q_n$. The second part is where our notion of simulatability comes into play, and where it is essential to deal carefully with the shared secrets.

For our latter results, we must actually generalize slightly this Theorem. A use case is for instance when we want to prove that $P\|Q \cong P\|P$ implies that $\mathbf{if}\, b\, \mathbf{then}\, P\, \mathbf{else}\, Q \cong P$ for some boolean condition b. In this case, we actually need to rename the channels used by P and Q in the second protocol, so that both P and Q uses the same channels. We thus introduce a renaming on channels σ that allows us to compose components in an arbitrary way.

Technical Details

The generalized version of the Theorem is as follows.

Theorem 5.3. *Let* $C[__1, \ldots, __n]$ *be a context. Let* $P_1, \ldots, P_n$, $Q_1, \ldots, Q_n$ *be protocols, and let* $\sigma : \mathcal{C}(P_1, \ldots, P_n) \mapsto \mathcal{C}$ *such that* $\overline{C}\|P_1\| \ldots \|P_n$, $\overline{C}\|Q_1\| \ldots \|Q_n$, $C[P_1\sigma, \ldots, P_n\sigma]$, $C[Q_1\sigma, \ldots, Q_n\sigma]$ *are protocols. Given a functional model* $\mathcal{M}_f$*, an oracle* $\mathcal{O}$*, if*

1. $\overline{n} \supseteq \mathcal{N}(C) \cap \mathcal{N}(P_1, \ldots, P_n, Q_1, \ldots, Q_n)$
2. $\nu\overline{n}.\overline{C}$ *is* $\mathcal{O}$*-simulatable*
3. $P_1\| \ldots \|P_n \cong_{\mathcal{O}} Q_1\| \ldots \|Q_n$

Then

$$C[P_1\sigma, \ldots, P_n\sigma] \cong_{\mathcal{O}} C[Q_1\sigma, \ldots, Q_n\sigma]$$

Specifically, there exists a polynomial p_S *(independent of* C*) such that, if* p_C *is the polynomial bound on the runtime of the simulator for* $\overline{C}$*, we have,*

$$\mathrm{Adv}^{C[P_1\sigma,\ldots,P_n\sigma] \cong_{\mathcal{O}} C[Q_1\sigma,\ldots,Q_n\sigma]}(t) \leq \mathrm{Adv}^{P_1 \| \ldots \| P_n \cong_{\mathcal{O}} Q_1 \| \ldots \| Q_n}\left(p_S\left(t, n, |C|, |\sigma|, p_C(t)\right)\right)$$

Proof. Let $\mathcal{A}$ be an attacker against

$$C[P_1\sigma, \ldots, P_n\sigma] \cong_{\mathcal{O}} C[Q_1\sigma, \ldots, Q_n\sigma].$$

In the scheduling part, we first build an attacker against

$$\overline{C}\|P_1\|\ldots\|P_n \cong_{\mathcal{O}} \overline{C}\|Q_1\|\ldots\|Q_n.$$

We then remove the context $\overline{C}$ through the $\mathcal{O}$-simulatability.

Scheduling part Let us construct $\mathcal{B}^{\mathcal{O},\mathcal{O}_{\overline{C}},\mathcal{O}_{R_1},\ldots,\mathcal{O}_{R_n}}$ with either for every i, $R_i = P_i$, or, for every i, $R_i = Q_i$. $\mathcal{B}^{\mathcal{O},\mathcal{O}_{\overline{C}},\mathcal{O}_{R_1},\ldots,\mathcal{O}_{R_n}}$ initially sets variables $c_1,\ldots,c_n$ to 0 (intuitively, c_i records which processes have been triggered) and sets $\overline{x}$ to the empty list. It then simulates $\mathcal{A}^{\mathcal{O},\mathcal{O}_{C[R_1\sigma,\ldots,R_n\sigma]}}$ but, each interaction with $\mathcal{O}_{C[R_1\sigma,\ldots,R_n\sigma]}$ and the corresponding request (c,m) is replaced with:

- if there exist i such that $c_i = 1$ and $c \in \mathcal{C}(R_i\sigma)$ then
  - query $\mathcal{O}_{R_i}$ with $(c\sigma^{-1}, m)$
  - if $\mathcal{O}_{R_i}$ returns $\bot$, then, if contexts C_1 and C_2 are such that $C[__1,\ldots,__n] = C_1[__i; C_2]$, it adds to $\overline{x}$ the channels $\mathcal{C}(C_2)$. (This corresponds to the semantics of sequential composition: an error message disables the continuation).
  - else the answer (c', m') is changed $(c'\sigma, m')$ (and the simulation goes on)
- else if $c \in \mathcal{C}(\overline{C})$ and $c \notin \overline{x}$ then
  - query $\mathcal{O}_{\overline{C}}$ with (c, m)
  - if $\mathcal{O}_{\overline{C}}$ answers $\top$ on channel γ_i, set $c_i = 1$
  - else continue with the reply of $\mathcal{O}_{\overline{C}}$

This new attacker is basically simply handling the scheduling of the protocols, using the signals raised in the context to synchronize everything. The condition that there exists i such that $c_i = 1$ and $c \in \mathcal{C}(R_i)$ is always satisfied by a unique i, otherwise $C[P_1\sigma,\ldots,P_n\sigma]$ or $C[Q_1\sigma,\ldots,Q_n\sigma]$ would not be well formed.

The execution time of $\mathcal{B}$ then only depends on the number of channels in C, the size of the channel substitution σ, the number of protocols n in addition to the cost of simulating $\mathcal{A}$. Hence if t is the runtime of $\mathcal{A}$, there exists p_{S_1} such that the runtime of $\mathcal{B}$ is bounded (uniformly in C, $P_1,\ldots,P_n,Q_1,\ldots Q_n$) by $p_{S_1}(n,t,|C|,|\sigma|)$:

$$\mathrm{Adv}^{C[P_1,\ldots,P_n]\cong C[Q_1,\ldots,Q_n]}_{\mathcal{A}^{\mathcal{O}}}(t) \le \mathrm{Adv}^{P_1\|\ldots\|P_n\cong Q_1\|\ldots\|Q_n}_{\mathcal{B}^{\mathcal{O},\mathcal{O}_{\overline{C}}}}(p_{S_1}(t,n,|C|,|\sigma|))$$

Simulatability Now, with the fact that $\nu\overline{n}.\overline{C}$ is $\mathcal{O}$-simulatable, we have a simulator $\mathcal{A}^{\mathcal{O}}_{\overline{C}}$ such that, thanks to Lemma 5.3, $\mathcal{B}[\mathcal{A}^{\mathcal{O}}_{\overline{C}}]^{\mathcal{O},\mathcal{O}_R}$ behaves exactly as $\mathcal{B}^{\mathcal{O},\mathcal{O}_{\overline{C}},\mathcal{O}_R}$. We have, for p_C the polynomial bound on the runtime of $\mathcal{A}_{\overline{C}}$, by Definition 5.4,

$$\mathrm{Adv}^{P_1\|\ldots\|P_n\cong Q_1\|\ldots\|Q_n}_{\mathcal{B}^{\mathcal{O},\mathcal{O}_{\overline{C}}}}(t) \le \mathrm{Adv}^{P_1\|\ldots\|P_n\cong Q_1\|\ldots\|Q_n}_{\mathcal{B}[\mathcal{A}^{\mathcal{O}}_{\overline{C}}]^{\mathcal{O}}}(q(p_C(t)+t))$$

and finally,

$$\begin{aligned}&\mathrm{Adv}^{C[P_1\sigma,\ldots,P_n\sigma]\cong C[Q_1\sigma,\ldots,Q_n\sigma]}_{\mathcal{A}^{\mathcal{O}}}(t)\\ &\le \mathrm{Adv}^{P_1\|\ldots\|P_n\cong Q_1\|\ldots\|Q_n}_{\mathcal{B}[\mathcal{A}^{\mathcal{O}}_{\overline{C}}]^{\mathcal{O}}}(q(p_C\circ p_{S_1}(n,t,|C|,|\sigma|)+p_{S_1}(n,t,|C|,|\sigma|)))\end{aligned}$$

■

Given a protocol P and a context C, for Theorem 5.2 to be used, we need an oracle such that:

1. the context C is simulatable with the oracle $\mathcal{O}$,

2. the protocol P is secure even for an attacker with access to $\mathcal{O}$ ($P \cong_{\mathcal{O}} Q$).

Our goal is to find an oracle that is generic enough to allow for a simple proof of indistinguishability of P and Q under the oracle, but still allows to simulate $\overline{C}$. Notably, if we take as oracle the protocol oracle corresponding to the context itself, we can trivially apply Theorem 5.2 but proving $P \cong_{\mathcal{O}} Q$ amounts to proving $C[P] \cong C[Q]$.

Application to tagged protocols We consider two versions of SSH, calling them SSH_2 and SSH_1, assuming that all messages are prefixed respectively with the strings "SSHv2.0" and "SSHv1.0". Both versions are using the same long term secret key sk for signatures. We assume that both versions check the string prefix.

To prove the security of SSH_2 running in the context of SSH_1, we can use Theorem 5.2. If we denote by I the idealized version of SSH_2, the desired conclusion is $SSH_2 \| SSH_1 \cong I \| SSH_1$. Letting $C[__1] = __1 \| SSH_1$, it is then sufficient to find an oracle $\mathcal{O}$ such that:

1. $\nu sk.SSH_1$ is $\mathcal{O}$-simulatable (the simulatability of C directly follows),
2. $SSH_2 \cong_{\mathcal{O}} I$

If we define the tagging function T_{SSH_1} that checks the prefix, SSH_1 is trivially $\mathcal{O}^{\mathtt{sign}}_{T_{SSH_1},sk}$-simulatable (see Definition 5.8) as SSH_1 does enforce the tagging checks. We thus let $\mathcal{O}$ be $\mathcal{O}^{\mathtt{sign}}_{T_{SSH_1},sk}$.

Assuming that `sign` verifies the classical EUF-CMA axiom, by Proposition 5.2, it also verifies the tagged version EUF-CMA$_{T_{SSH_1},sk}$. To conclude, it is then sufficient to prove that $SSH_2 \cong_{\mathcal{O}} I$ with a reduction to EUF-CMA$_{T_{SSH_1},sk}$.

Application to encrypt and sign For performances considerations, keys are sometimes used both for signing and encryption, for instance in the EMV protocol. In [PSS+11], an encryption scheme is proven to be secure even in the presence of a signing oracle using the same key. Our Theorem formalizes the underlying intuition, i.e. if a protocol can be proven secure while using this encryption scheme, it will be secure in any context where signatures with the same key are also performed.

5.4.2 Composition with State Passing

In some cases, a context passes a sequence of terms to another protocol. If the sequence of terms is indistinguishable from another one, we would like the two experiments, with either sequences of terms, to be indistinguishable.

Example 5.8. Let us consider once again the protocol $P(x_1, x_2) := \mathbf{in}(c, x).\mathbf{out}(c, \mathtt{enc}(x, x_1, x_2))$ of Example 2.3. We assume that we have a function kdf, which, given a random input, generates a suitable key for the encryption scheme. Let a random name $seed$ and let $C[__1] := \mathbf{let}\ sk = \mathsf{kdf}(seed)\ \mathbf{in}\ __1$. $C[\|^i P(r_i, sk)]$ provides an access to an encryption oracle for the key generated in C:

$$C[\|^i P(r_i, sk)] := \begin{array}{l} \mathbf{let}\ sk = \mathsf{kdf}(seed)\ \mathbf{in} \\ \qquad \|^i(\mathbf{in}(c, x).\mathbf{out}(c, \mathtt{enc}(x, r_i, sk))) \end{array}$$

A classical example is a key exchange, used to establish a secure channel. The situation is dual with respect to the previous theorem: contexts must be indistinguishable and the continuation must be simulatable.

Theorem 5.4. *Let C, C' be n-ary contexts such that each hole is terminal. Let $P_1(\overline{x}), \ldots, P_n(\overline{x})$ be parameterized protocols, such that channel sets are pairwise disjoint. Given a functional model $\mathcal{M}_f$, an oracle $\mathcal{O}$, $\overline{n} \supseteq \mathcal{N}(C) \cap \mathcal{N}(P_1, \ldots, P_n)$, $\overline{t_1}, \ldots, \overline{t_n}, \overline{t'_1}, \ldots, \overline{t'_n}$ sequences of terms, $\widetilde{C} := C[\boldsymbol{out}(c_1, \overline{t_1}), \ldots, \boldsymbol{out}(c_n, \overline{t_n})]$ and $\widetilde{C}' := C'[\boldsymbol{out}(c_1, \overline{t'_1}), \ldots, \boldsymbol{out}(c_n, \overline{t'_n})]$. If $\widetilde{C} \| \boldsymbol{in}(c_1, \overline{x}).P_1(\overline{x}) \| \ldots \| \boldsymbol{in}(c_n, \overline{x}).P_n(\overline{x})$ is a protocol and:*

1. *$\widetilde{C} \cong_{\mathcal{O}} \widetilde{C}'$*
2. *$\nu \overline{n}.\boldsymbol{in}(c_1, \overline{x}).P_1(\overline{x}) \| \ldots \| \boldsymbol{in}(c_n, \overline{x}).P_n(\overline{x})$ is $\mathcal{O}$-simulatable*

then $C[P_1(\overline{t_1}), \ldots, P_n(\overline{t_n})] \cong_{\mathcal{O}} C'[P_1(\overline{t'_1}), \ldots, P_n(\overline{t'_n})]$

Specifically, there exists a polynomial p_S (independent of $P_1, \ldots, P_n$) such that if p_O is the polynomial bound on the runtime of the simulator for $P := \boldsymbol{in}(c_1, \overline{x}).P_1(\overline{x}) \| \ldots \| \boldsymbol{in}(c_n, \overline{x}).P_n(\overline{x})$, we have,

$$\mathrm{Adv}^{C[P_1(\overline{t_1}),\ldots,P_n(\overline{t_n})] \cong_{\mathcal{O}} C[P_1(\overline{t'_1}),\ldots,P_n(\overline{t'_n})]}(t) \leq \mathrm{Adv}^{\widetilde{C} \cong_{\mathcal{O}} \widetilde{C}'}\Big(p_S\big(t, n, |P|, p_P(t)\big)\Big)$$

$\widetilde{C}$ is the context, in which all the bound values (for instance the key derived by a key exchange) are outputted on distinct channels. $\widetilde{C'}$ corresponds to the idealized version. We can pass those bound values to another protocol P, if this protocol P can be simulated for any possible value of the bound values.

Proof. The proof is very similar to Theorem 5.2.

Let us assume that we have an attacker such that

$$\mathrm{Adv}\left(\mathcal{A}^{\mathcal{O}, \mathcal{O}_{C[P_1(\overline{t_1}),\ldots,P_n(\overline{t_n})]?C[P_1(\overline{t'_1}),\ldots,P_n(\overline{t'_n})]}}\right) = \epsilon_0$$

We denote $C_1 = C[\mathbf{out}(c_1, \overline{t_1}), \ldots, \mathbf{out}(c_n, \overline{t_n})]$, $C_2 = C[\mathbf{out}(c_1, \overline{t'_1}), \ldots, \mathbf{out}(c_n, \overline{t'_n})]$, $P'_1 = in(1, \overline{x}).P_1(\overline{x})$, $\ldots$, $P'_n = in(n, \overline{x}).P_n(\overline{x})$. We first construct an attacker against:

$$C_1 \| P'_1 \| \ldots \| P'_n \cong C_2 \| P'_1 \| \ldots \| P'_n$$

Let us consider $\mathcal{B}^{\mathcal{O}, \mathcal{O}_D, \mathcal{O}_{P'_1}, \ldots, \mathcal{O}_{P'_n}}$ which simulates $\mathcal{A}^{\mathcal{O}, \mathcal{O}_{C[P_1(\overline{t_1}),\ldots,P_n(\overline{t_n})]?C[P_1(\overline{t'_1}),\ldots,P_n(\overline{t'_n})]}}$ but, after setting some variables $d_1, \ldots, d_n$ to 0 and some list $\overline{x}$ to the empty list, for every call to $\mathcal{O}_{C[P_1(\overline{t_1}),\ldots,P_n(\overline{t_n})]?C[P_1(\overline{t'_1}),\ldots,P_n(\overline{t'_n})]}$ of the form (c, m):

- if there exist i such that $d_i = 1$ and $c \in \mathcal{C}(P'_i)$ then
  - query $\mathcal{O}_{P'_i}$ with $(c\sigma^{-1}, m)$
  - if $\mathcal{O}_{P'_i}$ terminates set $c_i = 0$ and if it returns $\bot$, then, with $\overline{C}$ and C'' such that $C[__1, \ldots, __n] = \overline{C}[__i; C'']$ it adds to $\overline{x}$ the channels $\mathcal{C}(C'')$
  - else it forwards the answer (c', m') as $(c'\sigma, m')$
- else if $c \in \mathcal{C}(C_1)$ and $c \notin \overline{x}$ then
  - queries $\mathcal{O}_D$ with (c, m)
  - if $\mathcal{O}_D$ answers with some $\overline{t_i}$ on channel i
    - set $d_i = 1$
    - sends $(i, \overline{t_i})$ to $\mathcal{O}_{P'_i}$ and forwards the answer
  - else forwards the answer of $\mathcal{O}_D$

With this construction, we do have

$$\mathrm{Adv}\left(\mathcal{B}^{\mathcal{O},\mathcal{O}_{C_1?C_2},\mathcal{O}_{P'_1},\ldots,\mathcal{O}_{P'_n}}\right) = \epsilon_0$$

Using Lemma 5.1, we get a distinguisher $\mathcal{B}'$ such that:

$$\mathrm{Adv}\left(\mathcal{B}'^{\mathcal{O},\mathcal{O}_{C_1?C_2},\mathcal{O}_{P'_1\|\ldots\|P'_n}}\right) = \epsilon_0$$

Now, with the fact that $\nu\overline{n}.P'_1\|\ldots\|P'_n$ is $\mathcal{O}$ simulatable, we have a simulator $\mathcal{A}^{\mathcal{O}}_{P'_1\|\ldots\|P'_n}$ such that thanks to Proposition 5.1, $\mathcal{B}'[\mathcal{A}^{\mathcal{O}}_{P'_1\|\ldots\|P'_n}]^{\mathcal{O},\mathcal{O}_D}$ behaves exactly as $\mathcal{B}^{\mathcal{O},\mathcal{O}_{P'_1\|\ldots\|P'_n},\mathcal{O}_D}$.

We finally have $\mathrm{Adv}\left(\mathcal{B}'[\mathcal{A}^{\mathcal{O}}_{P'_1\|\ldots\|P'_n}]^{\mathcal{O},\mathcal{O}_{C_1?C_2}}\right) = \epsilon_0$.

The bound on the advantage is derived similarly to Theorem 5.2.

■

When we do so, we only assume that they are all distinct. The following example shows how Theorems 5.2 and 5.4 can be used to derive the security of one session of a key exchange composed with a protocol.

Example 5.9. Let us consider a key exchange $I\|R$ where x^I (resp. x^R) is the key derived by the initiator I (resp. the responder R) in case of success. We denote by $KE[__1,__2] := I;__1\|R;__2$ the composition of the key exchange with two continuations; the binding of x^I (resp. x^R) is passed to the protocol in sequence. Consider possible continuations $P^I(x^I), P^R(x^R)$ that use the derived keys and ideal continuations (whatever "ideal" is) $Q^I(x^I), Q^R(x^R)$. We sketch here how to prove $KE[P^I(x^I), P^R(x^R)] \cong KE[Q^I(x^I), Q^R(x^R)]$ (i.e., the security of the channel established by the key exchange). This will be generalized to multi-sessions in Section 5.6. We use both Theorems 5.2 and 5.4.

Assume, with a fresh name k, that:

1. $\mathcal{O}_{ke}$ is an oracle allowing to simulate the key exchange
2. $\mathcal{O}_{P,Q}$ allows to simulate $\mathbf{in}(c_I,x).P^I(x)\|\mathbf{in}(c_R,x).P^R(x)$ and $\mathbf{in}(c_I,x).Q^I(x)\|\mathbf{in}(c_R,x).Q^R(x)$
3. $P^I(k)\|P^R(k) \cong_{\mathcal{O}_{ke}} Q^I(k)\|Q^R(k)$
4. $KE[\mathbf{out}(c_I,x^I),\mathbf{out}(c_R,x^R)] \cong_{\mathcal{O}_{P,Q}} KE[\mathbf{out}(c_I,k),\mathbf{out}(c_R,k)]$

Hypothesis 3 captures the security of the channel when executed with an ideal key, and Hypothesis 4 captures the security of the key exchange. Both indistinguishability are for an attacker that can simulate the other part of the protocol.

Using Theorem 5.2 with Hypothesis 1 and 3 yields

$$KE[P^I(k), P^R(k)] \cong KE[Q^I(k), Q^R(k)]$$

Hypothesis 2 and 4 yield, with two applications of Theorem 5.4, one for P and one for Q, that $KE[P^I(x^I), P^R(x^R)] \cong KE[P^I(k), P^R(k)]$ and $KE[Q^I(x^I), Q^R(x^R)] \cong KE[Q^I(k), Q^R(k)]$. Transitivity allows us to conclude that the key exchange followed by the channel using the produced key is indistinguishable from the key exchange followed by the ideal secure channel:

$$KE[P^I(x^I), P^R(x^R)] \cong KE[Q^I(x^I), Q^R(x^R)]$$

In Theorem 5.4, the simulatability of

$$\nu\overline{n}.\mathbf{in}(c_P, k); P(k) \| \mathbf{in}(c_Q, k); Q(k)$$

may be a requirement too strong in some applications. This issue will be raised when we consider the forwarding agent of the SSH protocol, as detailed in Section 5.9.3, but we can avoid it in this specific case. For more complex applications, it might be interesting in the future to consider a weaker version of function applications where the produced key k always satisfies a condition $H(k)$. We could then design an oracle $\mathcal{O}$ so that for all names satisfying condition $H(k)$ we would have that $P(k)\|Q(k)$ is $\mathcal{O}$-simulatable.

5.4.3 Unbounded Replication

An important feature of a compositional framework is the ability to derive the security of a multi session protocol from the analysis of a single session. To refer to multiple sessions of a protocol, we consider that each session uses some fresh randomness that we see as a local session identifier.

The main idea behind the Theorem is that the oracle will depend on a sequence of names of arbitrary length. This sequence of names represents the list of honest randomness sampled by each party of the protocol, and the oracle enables simulatability of those parties.

We provide bellow the Proposition that allows to put in parallel any number of replications of simulatable protocols.

Proposition 5.3. *Let $\mathcal{O}_r$ be an oracle parameterized by a sequence of names $\overline{s}$, and $\mathcal{O}$ an oracle. Let $\overline{p}$ be a sequence of names, $P(\overline{x})$, $R_i^1(\overline{x},\overline{y}),\ldots,R_i^k(\overline{x},\overline{y})$ and $Q(\overline{x})$ be protocols, such that $\mathcal{N}_l(R_i^1,\ldots,R_i^k)$ is disjoint of the oracle support. If we have, for sequences of names $\overline{lsid}^1,\ldots,\overline{lsid}^k$, with $\overline{s} = \{\overline{lsid}_i^j\}_{1\leq j\leq k, i\in\mathbb{N}}$:*

1. *$\forall i,j \in \mathbb{N}, \nu\overline{p}, \overline{lsid}_i^j.R_i^j(\overline{p},\overline{lsid}_i^j)$ is $\mathcal{O}_r$-simulatable.*
2. *$P(\overline{p}) \cong_{\mathcal{O}_r} Q(\overline{p})$*
3. *$\overline{s}$ is disjoint of the support of $\mathcal{O}$.*

Then, for any integers $N_1,\ldots,N_k$:

$$\begin{aligned} &P(\overline{p})\|^{i\leq N_1}(R_i^1(\overline{p},\overline{lsid}_i^1))\|\ldots\|^{i\leq N_k}R_i^k(\overline{p},\overline{lsid}_i^k) \\ &\quad\cong_{\mathcal{O},\mathcal{O}_r} Q(\overline{p})\|^{i\leq N_1}R_i^1(\overline{p},\overline{lsid}_i^1)\|\ldots\|^{i\leq N_k}R_i^k(\overline{p},\overline{lsid}_i^k)\end{aligned}$$

Specifically, there exists a polynomial p_S (independent of all R^j) such that if p_{R^j} is the polynomial bound on the runtime of the simulator for R^j, we have,

$$\begin{aligned}&\mathrm{Adv}^{P(\overline{p})\|^{i\leq N_1}(R_i^1(\overline{p},\overline{lsid}_i^1))\|\ldots\|^{i\leq N_k}R_i^k(\overline{p},\overline{lsid}_i^k)\cong_{\mathcal{O}}Q(\overline{p})\|^{i\leq N_1}R_i^1(\overline{p},\overline{lsid}_i^1)\|\ldots\|^{i\leq N_k}R_i^k(\overline{p},\overline{lsid}_i^k)}(t) \\ &\quad\leq \mathrm{Adv}^{P(\overline{p})\cong_{\mathcal{O},\mathcal{O}_r}Q(\overline{p})}\Big(p_S\big(t,N_1,|R^1|,\ldots,N_k,|R^k|,p_{R^1}(t),\ldots,p_{R^k}(t)\big)\Big)\end{aligned}$$

Technical Details

In the previous proposition and following applications, we talk about sequences of names of the form $\overline{s} = \{\overline{lsid}_i^j\}_{1\leq j\leq k, i\in\mathbb{N}}$. This does not have any practical meaning and is only a shortcut. In practice, we must have that the previous hypotheses hold for any polynomial p and any sequence $\overline{s} = \{\overline{lsid}_i^j\}_{1\leq j\leq k, 1\leq i\leq p(\eta)}$. We will precisely define this in Section 6.4.

Applying the previous Proposition with P and Q as R^1 and R^2, we can obtain the Theorem for the unbounded replication of a protocol, where the number of sessions depends on the security parameter.

Theorem 5.5. *Let $\mathcal{O}_r$, $\mathcal{O}$ be oracles both parameterized by a sequence of names $\overline{s}$. Let $\overline{p}$ be a sequence of names, $P_i(\overline{x},\overline{y})$ and $Q_i(\overline{x},\overline{y})$ be parameterized protocols, such that $\mathcal{N}_l(P,Q)$ is disjoint of the oracles support. If we have, for sequences of names $\overline{lsid}^P, \overline{lsid}^Q$, with $\overline{s} = \{\overline{lsid_i}^P, \overline{lsid_i}^Q\}_{i\in\mathbb{N}}$:*

1. $\forall\ i \geq 1, \nu\overline{p}, \overline{lsid_i}^P.P_i(\overline{p}, \overline{lsid_i}^P)$ *is $\mathcal{O}_r$-simulatable.*
2. $\forall\ i \geq 1, \nu\overline{p}, \overline{lsid_i}^Q.Q_i(\overline{p}, \overline{lsid_i}^Q)$ *is $\mathcal{O}_r$-simulatable.*
3. *$\overline{s}$ is disjoint of the support of $\mathcal{O}$.*
4. $P_0(\overline{p}, \overline{lsid_0}^P) \cong_{\mathcal{O}_r,\mathcal{O}} Q_0(\overline{p}, \overline{lsid_0}^Q)$

then,

$$||^i P_i(\overline{p}, \overline{lsid_i}^P) \cong_{\mathcal{O}} ||^i Q_i(\overline{p}, \overline{lsid_i}^Q)$$

To prove this result, we use the explicit advantages that can be derived from our composition Theorems, which increases polynomially with respect to the number of sessions, and apply a classical hybrid argument to conclude.

In our applications (Section 5.6), the main idea is to first use Theorem 5.5 to reduce the multi-session security of a key exchange or a communication channel to a single session, and then use Theorems 5.2 and 5.4 to combine the multiple key exchanges and the multiple channels.

Remark, that in practice, to express the security properties of the protocols, we need to allow the protocols to use a predicate $T(x)$ whose interpretation may depend on the list of honest randomness sampled by each party of the protocol. For instance, this predicate may be used to check whether a value received by a party corresponds to a randomness sent by another party, and we would have $T(x) := x \in \overline{s}$. The two previous Theorems are in fact also valid in such cases, and we will use such notations in the application to key exchanges, but we delay to Chapter 6 the formalization of such predicates.

5.5 Unbounded Sequential Replication

We replicate a sequential composition where at each occurrence, a value produced by the protocol is transmitted to the next occurrence. This corresponds to the security of a protocol looping on itself, as it is the case for some key renewal protocols.

Such protocols depend on an original key, and are thus parameterized process of the form $P(x)$. As they renew the key stored in the variable x, they rebind x to some new value and thus contain a construct of the form **let** $x =$ _ **in** .

Proposition 5.4. *Let $\mathcal{O}$ be an oracle, two parameterized processes $P(x), Q(x)$, a set of names $\overline{n} = \mathcal{N}_g(P,Q)$ and fresh names k_0, l. We assume that $\mathcal{N}_l(P,Q)$ is disjoint of the support of $\mathcal{O}$. If:*

- $\nu\overline{n}.\mathbf{in}(c_P,x);P(x)\|\mathbf{in}(c_Q,x);Q(x)$ *is $\mathcal{O}$-simulatable, and*
- $P(k_0);\mathbf{out}(c_P,x)\|Q(k_0);\mathbf{out}(c_Q,x) \cong_{\mathcal{O}} P(k_0);\mathbf{out}(c_P,l)\|Q(k_0);\mathbf{out}(c_Q,l)$

then, for any N,

$$P(k_0); P(x)^{;N}; \mathbf{out}(c_P, x) \| Q(k_0); Q(x)^{;N}; \mathbf{out}(c_Q, x) \\ \cong_{\mathcal{O}} P(k_0); P(x)^{;N}; \mathbf{out}(c_P, l) \| Q(k_0); Q(x)^{;N}; \mathbf{out}(c_Q, l)$$

The main idea behind the proof is to perform as many function applications (Theorem 5.4) as needed, one for each replication of the protocol. Remark that compared to the previous replication, where we considered multiple sessions of the protocol and thus a notion of local session identifier was required, here we consider a single session looping on itself, and we do not need those identifiers.

5.6 Application to Key Exchanges

Section Summary

Although our framework is not specifically tailored to key exchanges or any specific property, we choose to focus here on this application. We outline how our theorems may be used to prove the security of a protocol using a key derived by a key exchange in a compositional way. (Let us recall that the key exchange and the protocol using the derived key may share long term secrets).

5.6.1 Our Model of Key Exchange

In order to obtain injective agreement, key exchanges usually use fresh randomness for each session as local session identifiers. For instance in the case of a Diffie-Hellman key exchange, the group shares may be seen as local session identifiers.

As in Example 5.9, KE is a key exchange with possible continuations. In addition, we consider multiple copies of KE, indexed by i, and local session identifiers $lsid$ for each copy:

$$KE_i[__1, __2] := I(lsid_i^I, id^I); __1 \| R(lsid_i^R, id^R); __2$$

Here, id captures the identities of the parties and $lsid$ captures the randomness that will be used by I and R to derive their respective local session identifiers. In the key exchange, I binds x^I to the key that it computes, x^I_{lsid} to the value of $lsid$ received from the other party and x^I_{id} to the received identity. Symmetrically, R binds the variables x^R, x^R_{lsid} and x^R_{id}.

If we denote by $P_i^I(x^I) \| P_i^R(x^R)$ the continuation (e.g., a record protocol based on the derived secret key), $KE_i[P_i^I(x^I), P_i^R(x^R)]$ is the composition of a session of the key exchange with the protocol where the values of x^I, x^R (computed keys) are passed respectively to $P_i^I(x^I)$ or $P_i^R(x^R)$. With Q an idealized version of P (however it is defined), the security of the composed protocol is expressed as follows:

$$\|^i KE_i[P_i^I(x^I), P_i^R(x^R)] \cong \|^i KE_i[Q_i^I(x^I), Q_i^R(x^R)]$$

Intuitively, from the adversary point of view, P is equivalent to its idealized version, even if the key is derived from the key exchange as opposed to magically shared.

Equivalently, the security of the composed protocol can be proved if we have that the advantage against the following indistinguishability is polynomial in N (and of course negligible).

$$\|^{i \leq N} KE_i[P_i^I(x^I), P_i^R(x^R)] \cong \|^{i \leq N} KE_i[Q_i^I(x^I), Q_i^R(x^R)]$$

A Corollary formalizing the following discussion can be found in Appendix B.1.

5.6.2 Proofs of Composed Key Exchange Security

Following the same applications of Theorems 5.2 and 5.4 as in Example 5.9, we decompose the proof of the previous indistinguishability goals into the following goals:

1. find an oracle $\mathcal{O}_{P,Q}$ to simulate multiple sessions of P or Q,
2. design an oracle $\mathcal{O}_{ke}$ to simulate multiple sessions of KE
3. complete a security proof under $\mathcal{O}_{ke}$ for multiple sessions of the protocol using fresh keys,
4. complete a security proof under $\mathcal{O}_{P,Q}$ for multiple sessions of the key exchange.

We further reduce the security of the protocol to smaller proofs of single sessions of the various components of the protocols under well chosen oracles. The following paragraphs successively investigate how to simplify the goals (1),(2),(3),(4) above. For simplicity, we only consider here the case of two fixed honest identities.

In the following, we provide the conditions S-1,S-2,P-1,P-2,P-3,P-4,K-1,K-2,K-3 that must be satisfied, so that we can prove

$$\|^i KE_i[P_i^I(x^I), P_i^R(x^R)] \cong \|^i KE_i[Q_i^I(x^I), Q_i^R(x^R)]$$

using our framework and the decomposition of Example 5.9. Corollary B.2, that formalizes the following discussion and generalizes it to non fixed identities, can be found in Appendix B.1.

We denote $\overline{p} = \{id^I, id^R\}$ and assume that they are the only shared names between KE, P and Q and are the only names shared by two distinct copies P_i, P_j (resp. Q_i, Q_j). We also denote by $\overline{s} = \{lsid_i^I, lsid_i^R\}_{i\in\mathbb{N}}$ the set of all copies of the local session identifiers.

Protocol simulatability For the simulation of the protocol, there must exists an oracle $\mathcal{O}_{P,Q}$ such that

S-1 $\nu\overline{p}.\mathbf{in}(c_I, x^I).P_i^I(x^I)\|\mathbf{in}(c_R, x^R).P_i^R(x^R)$ is $\mathcal{O}_{P,Q}$-simulatable

Indeed, if this condition is fulfilled (and a similar one replacing P with Q), then, thanks to Theorem 5.1, $\nu\overline{p}.\|^i(\mathbf{in}(c_I, x^I).P_i^I(x^I)\|\mathbf{in}(c_R, x^R).P_i^R(x^R))$ is $\mathcal{O}_{P,Q}$-simulatable (and similarly for Q). This meets the condition (2) of Theorem 5.4.

Key exchange simulatability For the simulation of the key exchange context, we need N (with N polynomial in the security parameter) copies of KE and, in each of them, the initiator (resp. the responder) may communicate with N possible responders (resp. initiators). We therefore use Theorem 5.2 with a context C with $2N^2$ holes. C is the parallel composition of N contexts and, as above, we use Theorem 5.1 to get the condition (1) of Theorem 5.2. Let KE_i' be[1]

$$KE_i[\ \underset{1\le j\le N}{\mathsf{if}}\ x_{lsid}^I = lsid_j^R\ \mathbf{then}\ \mathbf{out}(c_I, \langle i, j\rangle)\ \mathbf{else}\ \bot,$$
$$\underset{1\le j\le N}{\mathsf{if}}\ x_{lsid}^R = lsid_j^I\ \mathbf{then}\ \mathbf{out}(c_R, \langle i, j\rangle)\ \mathbf{else}\ \bot]$$

$\overline{C}$ is then $\|^{i\le N} KE_i'$ and C can be inferred by replacing each $\mathbf{out}(\langle i, j\rangle)$ with a hole. We output $\langle i, j\rangle$ so that we know that the full scheduling is simulatable. Then, the condition to be met by the key exchange is that

S-2 $\nu\overline{p}.KE_i'$ is $\mathcal{O}_{ke}$-simulatable

We then get, thanks to Theorem 5.1 the condition (1) of Theorem 5.2.

[1] we denote $\underset{1\le j\le N}{\mathsf{if}}\ c_i\ \mathbf{then}\ a_i\ \mathbf{else}\ a' := \mathbf{if}\ c_1\ \mathbf{then}\ a_1\ \mathbf{else\ if}\ c_2 \cdots\ \mathbf{then}\ a_n\ \mathbf{else}\ a'$

Security of the protocol Our goal is $\|^i P_i(k_i) \cong_{\mathcal{O}_{ke}} \|^i Q_i(k_i)$. Based on Theorem 5.5, we only need an oracle $\mathcal{O}_r$ so that:

P-1) $\forall\ i \geq 1, \nu\overline{p}, k_i.P_0(k_i)$ is $\mathcal{O}_r$-simulatable,
P-2) $\forall\ i \geq 1, \nu\overline{p}, k_i.Q_0(k_i)$ is $\mathcal{O}_r$-simulatable,
P-3) $\overline{s}$ is disjoint of the support of $\mathcal{O}_{ke}$,
P-4) $P_0(k_0) \cong_{\mathcal{O}_r, \mathcal{O}_{ke}} Q_0(k_0)$.

We use the fresh names k_i to model fresh magically shared keys, and use them as local sids for Theorem 5.5. The intuition is similar to the notion of Single session game of [BFW+11], where the considered protocols are such that we can derive the security of multiple sessions from one session. For instance, if the key is used to establish a secure channel, revealing the other keys does not break the security of one session, but allows to simulate the other sessions.

Security of the key exchange The security of the key exchange is more complicated to define, in the sense that it cannot simply be written with a classical replication. The partnering of sessions is not performed beforehand, so we must consider all possibilities. We may express the security of a key exchange by testing the real-or-random for each possible session key. We denote $k_{i,j}$ the fresh name corresponding to the ideal key that will be produced by the i-th copy of the initiator believing to be partnered with the j-th copy of the responder. The security of the key exchange is captured through the following indistinguishability:

$$\|^{i\leq N} KE_i[\mathbf{out}(x^I), \mathbf{out}(x^R)] \cong_{\mathcal{O}_{P,Q}} \|^{i\leq N}\ KE_i[\ \mathsf{if}\ \underset{1\leq j\leq N}{} x^I_{lsid} = lsid^R_j\ \mathbf{then}\ \mathbf{out}(k_{i,j})\ \mathbf{else}\ \bot,\ \mathsf{if}\ \underset{1\leq j\leq N}{} (x^R_{lsid} = lsid^I_j)\ \mathbf{then}\ \mathbf{out}(k_{j,i})\ \mathbf{else}\ \bot]$$

where the advantage of the attacker is polynomial in N. Remark that we sometimes omit channels, when they only need to be distinct.

Using a classical cryptographic hybrid argument (detailed in Proposition B.2), we reduce the security of multiple sessions to the security of one session in parallel of multiple corrupted sessions; the security of each step of the hybrid game is derived from Equation (5.1) using Theorem 5.4. It is expressed, with $state^X_i = \langle x^X, lsid^X_i, x^X_{lsid}\rangle$, as

$$\begin{aligned}\|^{i\leq N} KE_i[\mathbf{out}(\langle state^I_i\rangle), \mathbf{out}(\langle state^R_i\rangle)] \cong_{\mathcal{O}_{P,Q}}\ & \|^{i\leq N-1} KE_i[\mathbf{out}(\langle state^I_i\rangle), \mathbf{out}(\langle state^R_i\rangle)]\\ \|\ & KE_N[\ \mathbf{if}\ x^I_{lsid} = lsid^R_N\ \mathbf{then}\ \mathbf{out}(\langle k, lsid^I_N, x^I_{lsid}\rangle)\\ & \quad \mathbf{else\ if}\ x^I_{lsid} \notin \{lsid^R_i\}_{1\leq i\leq N-1}\ \mathbf{then}\ \bot,\\ & \quad \mathbf{else}\ \mathbf{out}(\langle state^I_i\rangle),\\ & \quad \mathbf{if}\ x^R_{lsid} = lsid^I_N\ \mathbf{then}\ \mathbf{out}(\langle k, lsid^R_N, x^R_{lsid}\rangle)\\ & \quad \mathbf{else\ if}\ x^R_{lsid} \notin \{lsid^I_i\}_{1\leq i\leq N-1}\ \mathbf{then}\ \bot,\\ & \quad \mathbf{else}\ \mathbf{out}(\langle state^R_i\rangle)]\end{aligned} \tag{5.1}$$

The previous equivalence expresses that when we look at N sessions that all output their full state upon completion, the particular matching of the parties in KE_N has a key that is real or random if they are indeed partnered together, and if they are not partnered together, they must be talking to another agent from the other KE_i. We may see the other sessions as corrupted sessions, as they leak their states upon completion.

We further reduce the problem to proving the security of a single session even when there is an oracle simulating corrupted sessions. To this end, we need to reveal the dishonest local session's identifiers to the attacker, but also to allow him to perform the required cryptographic operations, e.g. signatures using the identities.

We define, for $X \in \{I, R\}$, $\overline{s}^X$ as the set of copies of the local session identifiers of I or R, except a distinguished one (indexed 0 below) and $\overline{s} = \overline{s}^I \cup \overline{s}^R$. To obtain the security of multiple sessions of

the key exchange, we use Proposition 5.3.[2]. To this end, we would need to design an oracle $\mathcal{O}_r$, such that the following assumptions are satisfied, where $\mathcal{O}_{P,Q}$ corresponds to $\mathcal{O}$ of Proposition 5.3:

K-1) $\forall 1 \leq i \leq N, \nu lsid_i^I, id^I, lsid_i^R, id^R.$
$KE_i[\mathbf{out}(x^I), \mathbf{out}(x^R)] \| \mathbf{out}(\langle lsid_i^R, lsid_i^I \rangle)$ is $\mathcal{O}_r$ simulatable.

K-2) $KE_0[\ \mathbf{out}(\langle x^I, lsid_0^I, x_{lsid}^I \rangle),\ \mathbf{out}(\langle x^R, lsid_0^R, x_{lsid}^R \rangle)]$ $\cong_{\mathcal{O}_r, \mathcal{O}_{P,Q}}$ $KE_0[$ $\mathbf{if}\ x_{lsid}^I = lsid_0^R\ \mathbf{then}\ \mathbf{out}(\langle k, lsid_0^I, x_{lsid}^I \rangle)$ $\mathbf{else\ if}\ x_{lsid}^I \notin \overline{s}^R\ \mathbf{then}\ \bot$ $\mathbf{else}\ \mathbf{out}(\langle x^I, lsid_0^I, x_{lsid}^I \rangle),$ $\mathbf{if}\ x_{lsid}^R = lsid_0^I\ \mathbf{then}\ \mathbf{out}(\langle k, lsid_0^R, x_{lsid}^R \rangle)$ $\mathbf{else\ if}\ x_{lsid}^R \notin \overline{s}^I\ \mathbf{then}\ \bot$ $\mathbf{else}\ \mathbf{out}(\langle x^R, lsid_0^R, x_{lsid}^R \rangle)]$

K-3) $\overline{s}$ is disjoint of the support of $\mathcal{O}_{P,Q}$.

Intuitively, if the initiator believes to be talking to the honest responder, then it outputs the ideal key, and if it is not talking to any simulated corrupted party, it raises a bad event.

Note that while the structure of the proof does not fundamentally change from other proofs of key exchanges, e.g. [BFW+11], each step of the proof becomes straightforward thanks to our composition results. Our proofs are also more flexible, as shown by the extension to key exchanges with key confirmation in Section 5.8.

5.7 Basic Diffie-Hellman Key Exchange

Section Summary

We outline here the application of our framework to the ISO 9798-3 protocol, a variant of the Diffie-Hellman key exchange. It is proven UC composable in [KR17]. We use our result to extend the security proof to a context with shared long term secrets (which was not the case in the UC proof). We present the protocol in Figure 5.2, and show how to instantiate the required values and oracles to perform the proof presented in Section 5.6.2. The formal proofs (using the BC model [BC14a] and our tool) are provided in Part IV.

Our decomposition and subsequent proofs show that the DDH key exchange can be used to securely derive a secret key for any protocol that does not rely on the long term secret used in the key exchange. Our proof is also modular, in the sense that it could be adapted to provide also the security when the continuation protocol uses the long term shared secret as well.

A high level view of the protocol is given in Figure 5.2, and it is formally expressed in our algebra in Figure 5.3, where $_I$ and $_R$ denote the possible continuations at the end of each party. We use pattern matching in the inputs to simplify the notations, where for instance $\mathbf{in}(c, \langle m, x \rangle)$ with m some constant only accepts inputs whose first projection is m, and then bind the variable x to the second projection. If the inputs are not of the given form, the protocols goes to an error branch.

Our goal is to apply the decomposition of Section 5.6.2, for some abstract continuations P and Q that are supposed to used the derived key. We need to find suitable identities and local session identifiers so that the Conditions from the decomposition of Section 5.6.2 are fulfilled. As we do not specify P and Q, we only discuss the conditions relative to the security of the key exchange, e.g., K-1,K-2 and K-3. Remark that those conditions are sufficient to derive a notion similar to the classical security of a key exchange, as for any P and Q that do not share long term shared secrets

[2] We also use Theorem 5.1 to get the simulatability of N sessions in parallel from the simulatability of each session.

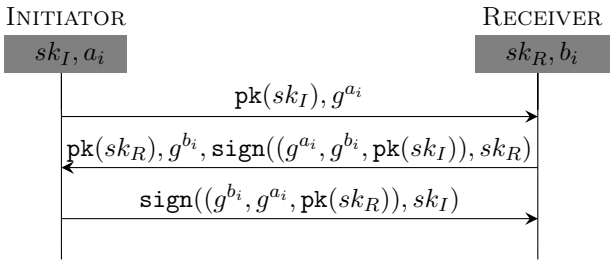


Figure 5.2: ISO 9798-3 Diffie Hellman Key Exchange

$$
\begin{array}{l}
\|^i \quad (\\
\quad I_i := \\
\qquad\qquad \mathbf{out}(\langle \mathtt{pk}(sk_I), g^{a_i} \rangle) \\
\qquad\qquad \mathbf{in}(\langle x_{pk}, x_B, x_m \rangle). \\
\qquad\qquad \mathbf{if}\ \mathtt{verify}(x_m, x_{pk}) = \langle g^{a_i}, x_B, \mathtt{pk}(sk_I) \rangle\ \mathbf{then} \\
\qquad\qquad\quad \mathbf{out}(\mathtt{sign}(\langle x_B, g^{a_i}, x_{pk} \rangle, sk_I)) \\
\qquad\qquad\quad \mathbf{let}\ k_I = x_B^{a_i}\ \mathbf{in} \\
\qquad\qquad\quad _I \\
\quad \| \\
\quad R_i := \\
\qquad\qquad \mathbf{in}(\langle x_{pk}, x_A \rangle). \\
\qquad\qquad \mathbf{out}(\langle \mathtt{pk}(sk_R), g^{b_i}, \mathtt{sign}(\langle x_A, g^{b_i}, x_{pk} \rangle, sk_R) \rangle) \\
\qquad\qquad \mathbf{in}(x_m). \\
\qquad\qquad \mathbf{if}\ \mathtt{verify}(x_m, x_{pk}) = \langle g^{b_i}, x_A, \mathtt{pk}(sk_R) \rangle\ \mathbf{then} \\
\qquad\qquad\quad \mathbf{let}\ k_R = x_A^{b_i}\ \mathbf{in} \\
\qquad\qquad\quad _R \\
\quad)
\end{array}
$$

Figure 5.3: ISO 9798-3 Diffie Hellman Key Exchange in the Pi Calculus (omitted channels)

with the key exchange. The other conditions are trivial to derive or only rely on the security of the continuation when using an ideal key.

The identity of each party is its long term secret key, and thus, we use sk_I and sk_R as id_I and id_R. Each session of the key exchange instantiates a fresh Diffie-Hellman share, that can be seen as a local session identifier. We thus use g^{a_i} and g^{b_i} as $lsid_i^I$ and $lsid_i^R$. These values can also be used as implicit tagging since any signed message either depends on a_i or b_i.

With those choices, we need to find a tagging function T that will provide a tagged oracle $\mathcal{O}_T$ such that the Conditions K of Section 5.6.2 are satisfied. Those Conditions, reformulated with the current notations and with $\mathcal{O}_T$ standing for $\mathcal{O}_r$, are expressed as follow:

K-1) $\forall 1 \leq i \leq N, \nu a_i, sk_I, b_i, sk_R.$
$I_i[\mathbf{out}(k_I)] \| R_i[\mathbf{out}(k_R)] \| \mathbf{out}(\langle g^{a_i}, g^{b_i} \rangle)$ is $\mathcal{O}_T$-simulatable.

K-2)
$$
\begin{array}{rl}
I_0\ [\mathbf{out}(\langle k_I, g^{a_0}, x_B \rangle)] & \\
\| R_0\ [\mathbf{out}(\langle k_R, g^{b_0}, x_A \rangle)] & \cong_{\mathcal{O}_T, \mathcal{O}_{P,Q}}
\end{array}
\begin{array}{l}
I_0 \left[\begin{array}{l}
\mathbf{if}\ x_B = g^{b_0}\ \mathbf{then\ out}(\langle x_B^{a_0}, g^{a_0}, x_B \rangle) \\
\mathbf{else\ if}\ x_B \notin \{g^{b_i}\}_{i \geq 1}\ \mathbf{then}\ \bot \\
\mathbf{else\ out}(\langle k_I, g^{a_0}, x_B \rangle)
\end{array} \right] \\
\| R_0 \left[\begin{array}{l}
\mathbf{if}\ x_A = g^{a_0}\ \mathbf{then\ out}(\langle x_A^{b_0}, g^{b_0}, x_A \rangle) \\
\mathbf{else\ if}\ x_A \notin \{g^{a_i}\}_{i \geq 1}\ \mathbf{then}\ \bot \\
\mathbf{else\ out}(\langle k_R, g^{a_0}, x_B \rangle)
\end{array} \right]
\end{array}
$$

K-3) $\{g^{a_i}, g^{b_i}\}_{i \geq 1}$ is disjoint of the support of $\mathcal{O}_{P,Q}$.

K-2 either corresponds to a matching conversation (i.e., all messages received by one were sent by the other) between the sessions with sids g^{a_0}, g^{b_0}, in which case the output is (twice) an ideal key k, or else it is a matching conversation with a simulated session, in which case it outputs the computed keys. It is neither of those cases, it should not happen, and we raise a bad event (denoted $\perp$). The proof of the K-2 is thus a real-or-random proof of a honestly produced key. We do not provide the proof of K-2 in this part, as it will be performed in the mechanized prover of Part IV.

We must define an implicit tagging that allows to both have the simulatability and the indistinguishability. Remark that first, we extend the tagging function T of Definition 5.8 so that it may depend on a second argument of arbitrary length, yielding $T(m, \overline{s})$, the corresponding signing oracle being denoted $\mathcal{O}^{\mathtt{sign}}_{T,sk,\overline{s}}$. This is required so that the implicit tagging may depend on all the possible local session identifiers. The exact definition of this extension is given in Section 6.4.

We define the implicit tagging functions T^I and T^R as

$$T^I(m, \{g^{a_i}, g^{b_i}\}_{i\geq 1}) := \exists s \in \{a_i\}_{i\geq 1}, \exists m_1, m_2.m = (m_1, g^s, m_2)$$
$$T^R(m, \{g^{a_i}, g^{b_i}\}_{i\geq 1}) := \exists s \in \{b_i\}_{i\geq 1}, \exists m_1, m_2.m = (m_1, g^s, m_2)$$

This tagging function will suit our needs, as all messages signed by the two parties follow this pattern. Moreover, in the protocol, the value sent in the first message should match g^{a_i} in the last message. Therefore, when the protocol of Figure 5.2 is successfully completed, we can prove that if $x_B \neq g^{b_0}$, then $x_B \in \{g^{b_i} | i \geq 1\}$, i.e., $T^R(x_B, \{g^{a_i}, g^{b_i}\}_{i\geq 1})$ is true (and similarly for R).

Let $\overline{s} = \{g^{a_i}, g^{b_i}\}_{i\geq 1}$, we finally define $\mathcal{O}_T = \mathcal{O}^{\mathtt{sign}}_{T^I,sk_I,\overline{s}}, \mathcal{O}^{\mathtt{sign}}_{T^R,sk_R,\overline{s}}, \mathcal{O}_{\overline{s}}$, where $\mathcal{O}_{\overline{s}}$ simply reveals the elements in $\overline{s}$, we do obtain the simulatability of multiple sessions of the key exchange (Hypothesis 1).

To adapt this proof to a concrete example, the security proof of K-2 would be performed under an oracle $\mathcal{O}_{P,Q}$ that allows to simulate the continuation (Condition P-1 of Section 5.6.2). The continuation should then be proven secure when using an ideal key (Conditions P of Section 5.6.2). In some cases, this step is trivial. Indeed, let us consider a record protocol $L := L^I(x^I) \| L^R(x^R)$, that exchanges encrypted messages using the exchanged key, and does not share any long term secret, i.e., does not use the signing keys of the key exchange. Without any shared secret, we do not need any oracle to simulate $\mathbf{in}(k); L^I(k) \| \mathbf{in}(k); L^R(k)$, so we can choose a trivial $\mathcal{O}_{P,Q}$ that does nothing.

5.8 Extension to Key Confirmations

Section Summary

We present how our compositional framework can be used to prove the security of a key exchange, in which the key is derived in a first part of the protocol and then used (key confirmation) in the second part. Compared to [BFS+13], our method allows in addition sharing of long term secrets.

Consider a key exchange $I(lsid^I_i, id^I) \| R(lsid^R_i, id^R)$. We further split I and R into $I_i := I^0_i(lsid^I_i, id^I); I^1_i(x^I)$ and $R_i := R^0_i(lsid^R_i, id^r); R^1_i(x^R)$, where I^0_i and R^0_i correspond to the key exchange up to, but not including, the first use of the secret key (x^I or x^R), and I^1_i and R^1_i are the remaining parts of the protocol. The intuition behind the proof of security is that at the end of I^0_i and R^0_i, i.e. just before the key confirmation, either the sessions are partnered together and the

derived key satisfies the real-or-random, or they are not, which means that the key confirmation performed by I_i^1 and R_i^1 will fail. We denote

$$KE_i[__1, __2] := I_i^0(lsid_i^I, id^I); I_i^1(x^I); __1 \| R_i^0(lsid_i^R, id^R); R_i^1(x^R); __2$$

and

$$KE_i^0[__1, __2] := I_i^0(lsid_i^I, id^I); __1 | R_i^0(lsid_j^R, id^R); __2$$

We proceed as in Section 5.6, outlining how we may split the security proof into smaller proofs using our framework, using the same composition Theorems at each step. We thus provide the necessary Conditions S-1,S-2,P-1,K-1,K-2,K-3 so that, for some continuation $P_i^I(x^I) \| P_i^R(x^R)$ and its idealized version Q,

$$\|^i KE_i[P_i^I(x^I), P_i^R(x^R)] \cong \|^i KE_i[Q_i^I(x^I), Q_i^R(x^R)]$$

A formal Corollary can be found in Appendix B.2.

5.8.1 Proofs with Key Confirmations

Key exchange and protocol simulatability We modify slightly the conditions S-1 and S-2 of Section 5.6.2 to reflect the fact that we now consider the key confirmation to be part of the continuation:

S-1) $\nu\overline{p}.\mathbf{in}(x).I^1(x); P^I(x)$, $\mathbf{in}(x).R^1(x); P^R(x)$, $\mathbf{in}(x).I^1(x); Q^I(x)$, $\mathbf{in}(x).R^1(x); Q^R(x)$ are $\mathcal{O}_{P,Q}$ simulatable.

S-2) $\nu\overline{p}.\ \|^{i\leq N}\ I_i^0(lsid_i^I, id^I);$ $\mathsf{if}\underset{1\leq i\leq N}{}\ x_{lsid}^I = lsid_j^R\ \mathbf{then}\ \mathbf{out}(\langle i,j\rangle)\ \mathbf{else}\ I_i^1(x^I); \bot$
$\|^{i\leq N}\ R_i^0(lsid_i^R, id^R);$ $\mathsf{if}\underset{1\leq i\leq N}{}\ x_{lsid}^R = lsid_j^I\ \mathbf{then}\ \mathbf{out}(\langle i,j\rangle)\ \mathbf{else}\ R_i^1(x^R); \bot$
is $\mathcal{O}_{ke}$-simulatable.

Security of the protocol Compared to Section 5.6.2, the continuation must be secure even in the presence of the messages produced during the key confirmation:

P-1) $\|^{i\leq N} I_i^1(x^I); P_i^I(x^I) \| R_i^1(x^R); P_i^R(x^R) \cong_{\mathcal{O}_r, \mathcal{O}_k} \|^{i\leq N} I_i^1(x^I); Q_i^I(x^I) \| R_i^1(x^R); Q_i^R(x^R)$

We could once again split this goal into a single session proof using Theorem 5.5. We remark that to prove the security of the single session, we can further reduce the proof by using an oracle that may simulate I^1 and R^1, as the security of P should not depend on the messages of the key confirmation.

Security of the key exchange We proceed in a similar way as in Section 5.6.2 and we use the same notations. The following Conditions are then suitable:

K-1) $\forall i \leq N, \nu lsid_i^I, id^I, lsid_i^R, id^R.$
$KE_i^0[\mathbf{out}(x^I), \mathbf{out}(x^R)] \| \mathbf{out}(\langle lsid_i^R, lsid_i^I\rangle)$ is $\mathcal{O}_T$-simulatable

K-2) $\overline{s}$ is disjoint of the support of $\mathcal{O}_{P,Q}$.

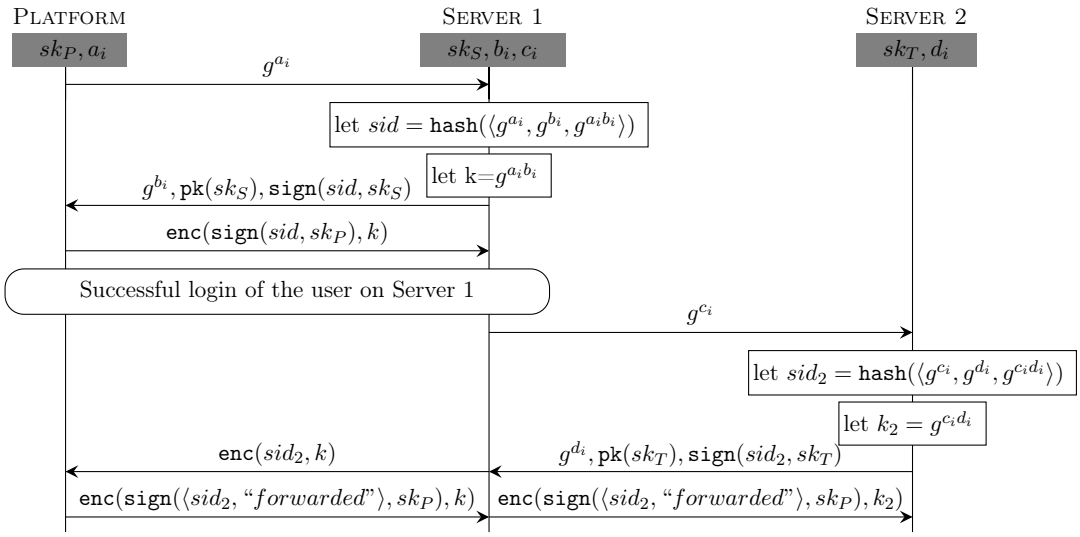


Figure 5.4: SSH with Forwarding Agent

K-3) $KE_0^0[\mathbf{if}\ x^I_{lsid} \notin \overline{s}^R\ \mathbf{then}\ I_0^1(x^I)\ \mathbf{else\ out}(\langle x^I, lsid_0^I, x^I_{lsid}\rangle),\ \mathbf{if}\ x^R_{lsid} \notin \overline{s}^I\ \mathbf{then}\ R_0^1(x^R)\ \mathbf{else\ out}(\langle x^R, lsid_0^R, x^R_{lsid}\rangle)]$
$\cong_{\mathcal{O}_{KE},\mathcal{O}_{P,Q}}$
$KE_0^0[\mathbf{if}\ x^I_{lsid} = lsid_0^R\ \mathbf{then\ out}(\langle k, lsid_0^I, x^I_{lsid}\rangle)\ \mathbf{else\ if}\ x^I_{lsid} \notin \overline{s}^R\ \mathbf{then}\ I_0^1(x^I); \mathbf{out}(\bot)\ \mathbf{else\ out}(\langle x^I, lsid_0^I, x^I_{lsid}\rangle),\ \mathbf{if}\ x^R_{lsid} = lsid_0^I\ \mathbf{then\ out}(\langle k, lsid_0^R, x^R_{lsid}\rangle)\ \mathbf{else\ if}\ x^R_{lsid} \notin \overline{s}^I\ \mathbf{then}\ R_0^1(x^R); \mathbf{out}(\bot)\ \mathbf{else\ out}(\langle x^R, lsid_0^R, x^R_{lsid}\rangle)]$

The indistinguishability expresses that, if the two singled out parties are partnered, i.e., $x^I_{lsid} = lsid_0^R$ or $x^R_{lsid} = lsid_0^I$, then we test the real-or-random of the key. Else, it specifies that a party must always be partnered with some honest session, i.e., that $x^X_{lsid} \notin \overline{s}^Y$ will never occur. To this end, on one side, when $x^X_{lsid} \notin \overline{s}^Y$ we run the key confirmation, and on the other side we run the key confirmation followed in case of success by a bad event. Finally, when two honest parties are partnered, but are not the singled out parties, they leak their states.

5.9 Application to SSH

Section Summary

SSH [YL] is a protocol that allows users to login onto a server from a remote platform. It is widely used in the version where signatures are used for authentication. An interesting feature is forwarding agent: once a user u is logged on a server S, they may, from S, perform another login on another server T. As S does not have access to the signing key of u, it forwards a signature request to u's platform using the secure SSH channel between u and S. This represents a challenge for compositional proofs: we compose a first key exchange with another one, the second one using a signature key already used in the first.

We provide the decomposition of the security proof of SSH composed with one (modified) forwarding agent. We use multiple times in sequence our composition Theorems, that allow us to further simplify the required indistinguishability proofs. The corresponding indistinguishability proofs are performed in Part IV.

There is a known weakness in this protocol: any privileged user on S can use the agents of any other user as a signing oracle. Thus, in order to be able to prove the security of the protocol,

$P_i :=$
out(g^{a_i});
in$(\langle x_B, \mathtt{pk}(sk_S), sign\rangle)$
let $k = x_B^{a_i}$ **in**
let $sid = \mathtt{hash}(\langle g^{a_i}, x_B, k\rangle)$ **in**
if $\mathtt{verify}(sign, \mathtt{pk}(sk_S)) = sid$ **then**
out$(\mathtt{enc}(\mathtt{sign}(sid, skP), k))$;
$_P$.

$S_i :=$
in(x_A);
let $k = x_A^{b_i}$ **in**
let $sid = \mathtt{hash}(\langle x_A, g^{b_i}, k\rangle)$ **in**
out$(\langle g^{b_i}, \mathtt{pk}(sk_S), \mathtt{sign}(sid, sk_S)\rangle)$
in$(\mathtt{enc}(x_{sign}, k))$
if $\mathtt{verify}(x_{sign}, \mathtt{pk}(skP)) = sid$ **then**
$_S$.

$$SSH := \|^i(P_i[0] \| S_i[0])$$

Figure 5.5: Basic SSH Key Exchange

we only consider the case where there is no such privileged user. Figure 5.4 presents an example of a login followed by a login using the forwarding agent. For simplicity, we abstract away some messages that are not relevant to the security of the protocol.

In the current specification of the forwarding agent, it is impossible for a server to know if the received signature was completed locally by the user's platform, or remotely through the forwarding agent. As the two behaviors are different in term of trust assumptions, we claim that they should be distinguishable by a server. For instance, a server should be able to reject signatures performed by a forwarded agent, because intermediate servers are not trusted. To this end, we assume that the signatures performed by the agent are (possibly implicitly) tagged in a way that distinguishes between their use in different parts of the protocol. This assumption also allows for domain separation between the two key exchanges, and thus simplifies the proof.

We consider a scenario in which there is an unbounded number of sessions of SSH, each with one (modified) forwarding agent, used to provide a secure channel for a protocol P. Thanks to multiple applications of Theorems 5.2 and 5.4, we are able to break the proof of this SSH scenario into small ones, that are very close to the proof of a simple Diffie-Hellman key exchange. This assumes the *decisional Diffie-Hellman* (DDH) hypothesis for the group, EUF-CMA for the signature scheme and that the encryption must ensure integrity of the cyphertexts (this last assumption is only required for the forwarded key exchange, where a signature is performed over an encrypted channel). P also has to satisfy the conditions of Section 5.8.1. In particular, it must be secure w.r.t. an attacker that has access to a hash that includes the exchanged secret key, since SSH produces such a hash. Note that the scenario includes multiple sessions, but only one forwarding. The extension would require an induction to prove in our framework the security for any number of chained forwardings.

5.9.1 The SSH Protocol

The basic SSH key exchange is presented in Figure 5.5, with possible continuations at the end denoted by $_P$ and $_S$. In this Section, we use a strong notion of pattern matching, where for instance **in**$(\mathtt{enc}(x_{sign}, k))$ is a syntactic sugar for **in**(x); **let** $x_{sign} = \mathtt{dec}(x, k)$ **in** $_$.

As it is always the case for key exchanges that contain a key confirmation, the indistinguishability of the derived key is not preserved through the protocol. The difficulty of SSH is moreover that once a user has established a secure connection to a server, they can from this server establish a secure connection to another server, while using the secure channel previously established to obtain the user credentials. We provide in Figure 5.6 a model of the SSH with forwarding of agent

$PDistant_i(oldk) :=$
  out(g^{a_i});
  in$(\langle x_B, \mathtt{pk}(sk_S), sign\rangle)$
  let $k = x_B^{a_i}$ **in**
  let $sid = \mathtt{hash}(\langle g^{a_i}, x_B, kP\rangle)$ **in**
  if $\mathtt{verify}(sign, \mathtt{pk}(sk_S)) = sid$ **then**
    out$(\mathtt{enc}(sid, oldk))$
    in$(\mathtt{enc}(sign, oldk))$
    out$(\mathtt{enc}(sign, k))$
    $_PD\cdot$

$ForwardAgent(k) :=$
  in$(\mathtt{enc}(sid, k))$
  out$(\mathtt{enc}(\mathtt{sign}(\langle sid, \text{“}fwd\text{”}\rangle, skP), k))$

$SForward_i :=$
  in(x_A);
  let $k = x_A^{b_i}$ **in**
  let $sid = \mathtt{hash}(\langle x_A, g^{b_i}, k\rangle)$ **in**
  out$(\langle g^{b_i}, \mathtt{pk}(sk_S), \mathtt{sign}(sid, sk_S)\rangle)$
  in$(\mathtt{enc}(sign, k))$
  if $\mathtt{verify}(sign, \mathtt{pk}(skP)) = \langle sid, \text{“}fwd\text{”}\rangle$ **then**
    $_SF$

$$SSH_{Forward} := \|^i(P_i[ForwardAgent(k)]\|SForward_i\|S_i[PDistant_i(k)])$$

Figure 5.6: SSH Key Exchange with Forwarding Agent

(reusing the definitions of P and S from Figure 5.5). After a session of P terminates successfully, a $ForwardAgent$ is started on the computer. It can receive on the secret channel a signing request and perform the signature of it. In parallel, after the completion of a session of S, a distant session of P that runs on the same machine as S can be initiated by $PDistant$. It will request on the previously established secret channel the signature of the corresponding sid. Finally, as the forwarding can be chained multiple time, at the end of a successful $PDistant$, a $ForwardServer$ is set up. It accepts to receive a signing request on the new secret channel of $PDistant$, forwards the request on the old secret channel, gets the signature and finally forwards it.

The forwarding agent implies a difficult composition problem: we sequentially compose a basic SSH exchange with a second one that uses the derived key and the same long term secret keys. Thus, to be able to prove the security of SSH with forwarding agent, we must be able to handle key confirmations and composition with shared long term secrets.

5.9.2 Security of SSH

We show how to prove the Conditions of Section 5.8 to the basic SSH protocol (without forwarding agent). We provide in Figure 5.7 the decomposition for key exchanges with key confirmation corresponding to the SSH protocol. We directly specify that P and S may only relate to each other by hard-coding the expected public keys in them. This is the classical behaviour of SSH where a user wants to login on a specific server, and the public key of the user was registered previously on the server.

For some abstract continuation $R^P(x)\|R^S(x)$ and its idealized version $Q^P(x)\|Q^S(x)$, our goal would be to prove that

$$P_i^0; P_i^1(x_B, k)[R^P(k)]\|S_i^0; S_i^1(sid, k)[R^S(k)] \cong P_i^0; P_i^1(x_B, k)[Q^P(k)]\|S_i^0; S_i^1(sid, k)[Q^S(k)]$$

$P_i^0 :=$
  $\mathbf{out}(g^{a_i})$;
  $\mathbf{in}(x_B)$
  $\mathbf{let}\ k = x_B^{a_i}\ \mathbf{in}$
  0.
$P_i^1(x_B, k) :=$
  $\mathbf{in}(\langle \mathtt{pk}(sk_S), sign\rangle)$
    $\mathbf{let}\ sid = \mathtt{hash}(\langle g^a, x_B, k\rangle)\ \mathbf{in}$
    $\mathbf{if}\ \mathtt{verify}(sign, \mathtt{pk}(sk_S)) = sid\ \mathbf{then}$
      $\mathbf{out}(\mathtt{enc}(\mathtt{sign}(sid, sk_P), k))$
      $_P$.

$S_i^0 :=$
  $\mathbf{in}(x_A)$;
  $\mathbf{let}\ k = x_A^{b_i}\ \mathbf{in}$
  $\mathbf{let}\ sid = \mathtt{hash}(\langle x_A, g^{b_i}, k\rangle)\ \mathbf{in}$
    $\mathbf{out}(g^{b_i})$
$S_i^1(sid, k) :=$
    $\mathbf{out}(\langle \mathtt{pk}(sk_S), g^{b_i}, \mathtt{sign}(sid, sk_S)\rangle)$
    $\mathbf{in}(\mathtt{enc}(sign, k))$
    $\mathbf{if}\ \mathtt{verify}(sign, \mathtt{pk}(sk_P)) = sid\ \mathbf{then}$
      $_S$.

Figure 5.7: Divided SSH Key Exchange

Without specifying the continuation, a first step toward the security of the basic SSH key exchange is to obtain Conditions K-1 and K-3 of Section 5.8. Recall that if a key exchange satisfies those Conditions, it can be seen as a secure key exchange in the classical sense as it can be composed with any continuation that do not share any long term secrets. The proofs only need to ne adapted when it is not the case.

The behaviour of the protocol is very similar to the signed DDH key exchange (Figure 5.2) previously studied. We can once again see the DH shares $\{a_i, b_i\}_{i\in\mathbb{N}}$ as local session identifiers that can be used to pair sessions. For each session and each party, the messages signed by this party always depend strongly on the DH share. We can thus make all SSH sessions simulatable with the following tagging functions and corresponding signing oracles.

$$\begin{aligned} T_P(m, \overline{s}) :=&\quad \exists s \in \{a_i\}_{i\in\mathbb{N}}, \exists m_1, m = \mathtt{hash}(g^s, m_1, m_1^s) \\ T_S(m, \overline{s}) :=&\quad \exists s \in \{b_i\}_{i\in\mathbb{N}}, \exists m_1, m = \mathtt{hash}(m_1, g^s, m_1^s) \end{aligned}$$

We have that the set of axioms $Ax = \text{EUF-CMA}_{T_P, sk_P, \overline{s}} \wedge \text{EUF-CMA}_{T_S, sk_S, \overline{s}}$ is $\mathcal{O}^{\mathtt{sign}}_{T_P, F, sk_P, \overline{s}}, \mathcal{O}^{\mathtt{sign}}_{T_S, F, sk_Q, \overline{s}}, \mathcal{O}_{a_i, b_i}$ sound thanks to Proposition 6.1. We use those axioms to perform the proof of K-3, where the tagging essentially implies the authentication property. However, the proof must be slightly stronger, when we consider that the continuations P, Q are instantiated with a second round of SSH with a forwarding agent that uses the same long term secrets.

5.9.3 SSH with Forwarding Agent

For concision, we write FA for $ForwardAgent$, SF for $SForward$, and PD for $PDistant$. Let us consider an abstract continuation protocol, satisfying a security property of the form $R^P(k) \| R^S(k) \cong Q^P(k) \| Q^S(k)$ where k denotes a fresh name modelling an ideal key produced by a key exchange.

We once again assume that the agents are only willing to communicate with the honest identities, i.e., $\mathtt{pk}(sk_S)$ and $\mathtt{pk}(sk_P)$ are predefined in the processes. The goal is to prove the following equivalence.

$$\begin{array}{lll} \|^i & (P_i[FA(k)] & \cong \|^i \quad (P_i[FA(k)] \\ & \|S_i[PD(k); R^P(k_{PD})] & \quad\ \ \|S_i[PD(k); Q^P(k_{PD})] \\ & \|SF[R^S(k_{SF})]) & \quad\ \ \|SF[Q^S(k_{SF})]) \end{array}$$

It corresponds to the fact that we should have $R^P(k)\|R^S(k) \cong Q^P(k)\|Q^S(k)$, even if the ideal key k is replaced for each party by a key derived by a SSH key exchange (PD and SF) using an forwarding agent (FA) based on a previous SSH key exchange (P and S).

We apply twice the decomposition of Section 5.8, once to show the security of the first key exchange (as done in the previous paragraph), and that we can thus prove the security of the second key exchange using an ideal key derived instead of the one derive by the first exchange. The second application is then used to prove the security of this second key exchange.

First application The fist application is performed with the following Conditions (corresponding to the one of Section 5.8), which allow to derive the desired conclusion.
K-3):

$$\begin{array}{ll} P_0^0; & \textbf{if } x_B \notin \overline{s} \textbf{ then} \\ & \quad P_0^1(x_B,k); \textbf{out}(k) \\ & \textbf{else out}(k, g^{a_0}, x_B) \\ \|S_0^0; & \textbf{if } x_A \notin \overline{s} \textbf{ then} \\ & \quad S_i^1(x_A,k); \textbf{out}(k) \\ & \textbf{else out}(k, g^{b_0}, x_A) \end{array} \cong_{\mathcal{O}_{PS},\mathcal{O}_{forward}} \begin{array}{ll} P_0^0; & \textbf{if } x_B = g^{b_0} \textbf{ then} \\ & \quad \textbf{out}(k, g^{a_0}, x_B) \\ & \textbf{else if } x_B \notin \overline{s} \textbf{ then} \\ & \quad P^1(x_B,k); bad \\ & \textbf{else out}(k, g^{b_0}, x_A) \\ \|S_0^0; & \textbf{if } x_A = g^{a_0} \textbf{ then} \\ & \quad \textbf{out}(k, g^{b_0}, x_A) \\ & \textbf{else if } x_B \notin \overline{s} \textbf{ then} \\ & \quad S_0^1(x_A,k); bad \\ & \textbf{else out}(k, g^{b_0}, x_A) \end{array}$$

P-1):

$$\|^i P_i^1(k)[FA(k)]\|S_i^1(k)[PD(k); R^P]\|SF[R^S] \cong_{\mathcal{O}_{KE_1}} \|^i P_i^1(k)[FA(k)]\|S_i^1(k)[PD(k); Q^P]\|SF[Q^S]$$

We use the following oracles:

- $\mathcal{O}_{PS}$ allows to simulate (K-1) the other honest sessions of P and S, it corresponds to $\mathcal{O}^{\mathtt{sign}}_{T_P,F,sk_S,\overline{s}}, \mathcal{O}^{\mathtt{sign}}_{T_S,F,sk_P,\overline{s}}, \mathcal{O}_{a_i,b_i}$ of Section 5.9.2.
- $\mathcal{O}_{forward}$ allows to simulate (S-1) the continuation, i.e., protocols of the form $\mathbf{in}(k); P^1(k)[FA(k)]\|\mathbf{in}(k); S^1(k)[PD(k); R^P]\|SF[R^Q]$
- $\mathcal{O}_{KE_1}$ allows to simulate (S-2) $\|^i(P_i\|S_i)$ (it is identical to $\mathcal{O}_{PS}$).

All simulations are performed under $\nu sk_S, sk_P$. To define $\mathcal{O}_{forward}$, we need to settle an issue. Indeed, for hypothesis S-1, we need to provide an oracle that can simulate sessions of the forwarding protocols. However, in order to get the simulatability of $\mathbf{in}(k).FA(sk_P,k)$, one must give a generic signing oracles to the attacker, which would obviously make the protocol insecure. Based on the assumption that the forwarded sessions perform signatures tagged with "*fwd*" (as shown below), we can however provide a signing oracle for such messages only. It allows for the simulatability of the forwarding agent and of the forwarded client and server. More specifically, recall the the forwarding agent is of the form:

$$\begin{array}{l} FA(sk_P,k) := \\ \quad \mathbf{in}(\mathtt{enc}(sid,k)); \\ \quad\quad \mathbf{out}(\mathtt{enc}(\mathtt{sign}(\langle sid, \text{“}fwd\text{”}\rangle, sk_P), k)) \end{array}$$

We may obtain its simulatability with the following tagging function:

$$T_{for}(m,\overline{s}) := \quad \exists m_1.\ m = \langle m_1, \text{“}fwd\text{”}\rangle$$

Then, $\mathcal{O}_{forward}$ is simply $\mathcal{O}^{\mathtt{sign}}_{T_{for},F,sk_P,\overline{s}}, \mathcal{O}^{\mathtt{sign}}_{T_{for},F,sk_S,\overline{s}}, \mathcal{O}_{a'_i,b'_i}$. We prove Condition K-3 under the corresponding EUF-CMA axioms in Part IV.

Second application We further simplify Condition P-1 of the previous paragraph with a second application of the decomposition of Section 5.8. We now denote $\overline{s}' = \{a'_i, b'_i\}_{i\in\mathbb{N}}$. PD_i and SF_i are split into PD_i^0, PD_i^1 and SF_i^0, SF_i^1 similarly to the split of Figure 5.7 before and after the key confirmation. The tagging functions used are only slight variations of the tagging functions for the first SSH key exchange:

$$\begin{aligned} T'_P(m,\overline{s}') := &\quad \exists i, \exists X, m = \langle \mathtt{hash}(g^{a'_i}, X, X^{a'_i}), \text{“}fwd\text{”}\rangle \\ T'_S(m,\overline{s}') := &\quad \exists i, \exists X, m = \langle \mathtt{hash}(X, g^{b'_i}, X^{b'_i}), \text{“}fwd\text{”}\rangle \end{aligned}$$

We then need to prove the Conditions:

K-3):

$$\begin{aligned} P_0^1(k); FA(k) \| S_0^1(k); PD_0^0(k); \quad & \textbf{if } x_B \notin \overline{s}' \textbf{ then} \\ & \quad PD_0^1(x_B,k); out(k) \\ & \textbf{else } out(k, g^{a'_0}, x_B) \\ \| SF_0^0; \quad & \textbf{if } x_A \notin \overline{s}' \textbf{ then} \\ & \quad SF_0^1(x_A,k); out(k) \\ & \textbf{else } out(k, g^{b'_0}, x_A) \end{aligned}$$

$$\cong_{\mathcal{O}_{KE_1}, \mathcal{O}^k_{FPS}, \mathcal{O}_{RQ}}$$

$$\begin{aligned} P_0^1(k); FA(k) \| S_0^1(k); PD_0^0; \quad & \textbf{if } x_B = g^{b'_0} \textbf{ then} \\ & \quad out(k, g^{a'_0}, x_B) \\ & \textbf{else if } x_B \notin \overline{s}' \textbf{ then} \\ & \quad PD_0^1(x_B,k); bad \\ & \textbf{else } out(k, g^{b'_0}, x_A) \\ \| SF_0^0; \quad & \textbf{if } x_A = g^{a'_0} \textbf{ then} \\ & \quad out(k, g^{b'_0}, x_A) \\ & \textbf{else if } x_B \notin \overline{s}' \textbf{ then} \\ & \quad SF_0^1(x_A,k); bad \\ & \textbf{else } out(k, g^{b'_0}, x_A) \end{aligned}$$

Note that k is a fresh name that could be considered as a long term secret, i.e., in $\overline{p}$.

And P-1):

$$\|^i PD_i^1(k'); R^P(k') \| SF_0^1(k'); R^S(k') \cong_{\mathcal{O}_{KE_1}, \mathcal{O}_{FPS}} \|^i PD_i^1(k'); Q^P(k') \| SF_i^1(k'); Q^S(k')$$

With the oracles:

- $\mathcal{O}^k_{FPS}$ allows to simulate (K-1) the other honest sessions of PD and SF, it corresponds to $\mathcal{O}^{\mathtt{sign}}_{T'_P,sk_S,\overline{s}}, \mathcal{O}^{\mathtt{sign}}_{T'_S,sk_P,\overline{s}}, \mathcal{O}_{a'_i,b'_i}$ of Section 5.9.2.
- $\mathcal{O}_{RQ}$ allows to simulate (S-1) the continuation, i.e., protocols of the form

$$\mathbf{in}(k); PD^1(k); R^P(k) \| \mathbf{in}(k); SF^1(k); R^Q(k)$$

We once again prove Condition K-3 under the corresponding EUF-CMA axioms in Part IV. Remark that to ensure that the forwarding agent only signs the *sid* sent by PD, it is required that the encryption scheme is an authenticated encryption scheme.

6 The Framework in the BC Logic

> Nul n'est jamais assez fort pour ce calcul.
>
> *(Troisième théorème d'incomplétude)*

6.1 Introduction

Our framework allows to split the indistinguishability proof of a compound protocol into the indistinguishability proof of its components. The core idea is that instead of proving indistinguishability against PTTM attackers, we give some extra power to the attackers by giving them access to an oracle. A protocol secure against such attackers is secure in any context that the attacker can simulate thanks to the oracle. The composition framework does not depend on any particular technique to prove the hypothesis of its Theorem. We outlined how one can define axioms that are sound even for attackers with access to some oracle. Those axioms could be used to derive the compositional security of a protocol, for instance using the classical game hopping technique to perform proofs under those axioms. Keeping in mind the necessity for proofs to be formal and machine-checked, EasyCrypt could be adapted to perform such proofs.

As we believe that the BC logic bears promises of automation and ease of use (in the specific case of protocols), we rather focus on this specific model, and show in this Chapter how it can be used in the context of the composition framework. As we actually designed our framework with the BC logic in mind, the BC logic is easily extended to support attackers with oracles, and similarly for its axiomatic system.

Using the composition framework in the BC logic yields an interesting side result: we can for the first time derive proofs of security for an unbounded number of sessions from a proof in BC. Indeed, as BC models protocols as terms in a first-order logic, there is no simple way to model protocols with an unbounded number of sessions. Moreover, as the advantage of the adversary is not explicit, even when performing proofs by induction over the number of sessions, we cannot derive the security of an unbounded number of sessions.

Chapter Summary

In the composition framework of Chapter 5, protocols are proved secure against attackers with access to an oracle. Protocols secure under an oracle are then also secure in any simulatable context, i.e., any context that an attacker can simulate using the oracle. To perform such proofs, we use axioms that hold even for attackers with access to a given oracle.

In this Chapter, we extend the BC logic so that it supports such oracles. The BC logic then allows for further simplification of our framework, as its axiomatic system is easily adapted to the new cryptographic assumptions. This opens the way to mechanized proofs, and also allow for the first time to derive the security of an unbounded number of sessions of a protocol from a proof in BC. We present in Part IV a tool dedicated to the BC logic, and use it to showcase applications of our framework.

6.1.1 Our Contributions

We extend the BC logic so that it can capture attackers with access to oracles. Our composition framework can then be used side by side with the BC logic, aiming at providing formal and modular security proofs. As the BC logic is a first-order logic, it is amenable to mechanization, thus paving the way for mechanized modular proofs of security. We will provide such mechanization in Part IV.

In a second step, we also extend the BC logic so that one can use predicates depending on infinite sequences of names, as it is required by the key exchange application to be able to test, for instance given a variable x and an indexed name a_i, if $x \in \{a_i\}_{i\in\mathbb{N}}$.

Moreover, as our reductions from one session to multiple sessions are uniform, we may now complete proofs in the BC logic for a number of sessions that is parameterized by the security parameter. This was a limitation (and left as an open issue) in all previous BC works, as it was either only possible to

- perform a proof for a bounded number of sessions,
- or via an induction derive a proof for each number of sessions, but it does not depend on the security parameter.

6.1.2 Related Work

To the best of our knowledge, efforts towards the formal verification of protocols through both composition results and machine checked proofs are very nascent. Blanchet [Bla18] provides multiple composition theorems, and the CryptoVerif tool was used to prove the required proofs. It was used to prove the security of TLS. The composition theorems are dedicated to key exchanges. An attempt at casting the UC model in EasyCrypt was performed by Canetti et al. [CSV19]. The application are restricted to very basic toy protocols. The constructive cryptography paradigm has been formalized in the CryptHOL by Lochbihler et al. [LSB+19].

6.2 Oracles in the BC Logic

Section Summary

We extend the semantics of the BC logic so that it now refers to attackers that can have access to an extra oracle $\mathcal{O}$. We then lift the notion of soundness for the axioms to support oracles, defining the notion of $\mathcal{O}$-soundness.

6.2.1 Syntax and Semantics

We introduced the BC logic in Section 2.4. We here generalize the Definition and Propositions of this Section to handle attackers with access to oracles. While the functional model stays as is, the computational model must now also depend on some oracle that is given to the attacker, and the corresponding random oracle tape. Definition 2.15 now becomes as follows.

Definition 6.1. A *computational model* $\mathcal{M}$ is an extension of a functional model $\mathcal{M}_f$, which provides an oracle $\mathcal{O}$, and an additional PTOM $\mathcal{A}_g^{\mathcal{O}}$ for each symbol $g \in \mathcal{G}$, that takes as input an infinite random tape ρ_r, a security parameter 1^η and a sequence of bitstrings.

We define the interpretation of extended terms as, given $\mathcal{M}$, η, σ, ρ_s , $\rho_{\mathcal{O}}$ and ρ_r:

- $[\![n]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}} := \mathcal{A}_n(1^\eta, \rho_s)$ if $n \in \mathcal{N}$
- $[\![x]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}} = [\![x\sigma]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}}$ if $x \in \mathcal{X}$
- $[\![f(\overline{u})]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}} = \mathcal{A}_f(1^\eta, [\![\overline{u}]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}})$ if $f \in \Sigma$
- $[\![g(\overline{u})]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}} = \mathcal{A}_g^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}([\![\overline{u}]\!]^{\eta,\sigma}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}}, \rho_r, 1^\eta)$ if $g \in \mathcal{G}$

We also adapt Definition 2.16 of the interpretation of $\sim$.

Definition 6.2. Given a computational model $\mathcal{M}$, including an oracle $\mathcal{O}$, two sequences of terms $\overline{t}$, $\overline{u}$, and an assignment σ of the free variables of $\overline{t}, \overline{u}$ to ground terms, we have $\mathcal{M}, \sigma \models_{\mathcal{O}} \overline{t} \sim \overline{u}$ if, for every polynomial time oracle Turing machine $\mathcal{A}^{\mathcal{O}}$,

$$\begin{aligned}|\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}([\![\overline{t}]\!]^{\sigma,\eta}_{\rho_s;\rho_r;\rho_{\mathcal{O}}}, \rho_r, 1^\eta) = 1\} \\ -\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}([\![\overline{u}]\!]^{\sigma,\eta}_{\rho_s;\rho_r;\rho_{\mathcal{O}}}, \rho_r, 1^\eta) = 1\}|\end{aligned}$$

is negligible in η. Here, $\rho_s, \rho_r, \rho_{\mathcal{O}}$ are drawn according to a distribution such that every finite prefix is uniformly sampled.

6.2.2 Oracle Soundness

To perform proofs in the logic, we need to design axioms that are sound w.r.t. an attacker that has access to $\mathcal{O}$; we say that the axiom is $\mathcal{O}$-sound in this case. They should be easy to verify for actual libraries, yet powerful enough for the proofs that we intend to complete. The purpose of this Section is to provide such axioms. We first extend the notion of soundness to oracles.

Definition 6.3. Given a family of computational models $\mathbb{F}$ using oracle $\mathcal{O}$, a set of first order formulas A is $\mathcal{O}$-*sound* (w.r.t. $\mathcal{F}$) if, for every $\psi \in A$, every $\mathcal{M} \in \mathcal{F}$, $\mathcal{M} \models_{\mathcal{O}} \psi$.

With such a definition, if A is $\mathcal{O}$-sound (w.r.t. $\mathcal{F}$) and $A \models \phi$ (where ϕ is a closed formula), then, for every $\mathcal{M} \in \mathcal{F}$, $\mathcal{M} \models_{\mathcal{O}} \phi$.

Example 6.1 (Function application)**.** For any $\mathcal{O}$, $\mathcal{F}$, function f, terms $t_1, \ldots, t_n$, $u_1, \ldots, u_n$

$$t_1, \ldots, t_n \sim u_1, \ldots, u_n \implies f(t_1, \ldots, t_n) \sim f(u_1, \ldots, u_n)$$

is $\mathcal{O}$ sound.

Example 6.2. Given a single key encryption oracle $\mathcal{O}$ for key k, the formula

$$\mathtt{enc}(0, r, k) \sim \mathtt{enc}(1, r, k)$$

is

- not sound (nor $\mathcal{O}$-sound) in general,
- sound but not $\mathcal{O}$-sound for non randomized SPRP encryption,
- $\mathcal{O}$-sound for IND-CPA encryption.

Note that the axioms that are designed in [BC14a] cannot be borrowed directly. For instance, $n \sim n'$, where n, n' are names, is a standard axiom: two randomly generated numbers of the same length cannot be distinguished. However, if either n or n' is in the support of $\mathcal{O}$, some information on their interpretation can be leaked by the oracle. The axiom $n \sim n'$ is sound, but not $\mathcal{O}$-sound. We have to modify this axioms as follows:

Lemma 6.1. *For any oracle $\mathcal{O}$ with support $\overline{n}$, the axiom $\forall k, k' \notin \overline{n}, k \sim k'$ is $\mathcal{O}$-sound.*

Proof. We are given a functional model, and oracle $\mathcal{O}$ with support $\overline{n}$, and two names k, k' not in the support. We are also given $\mathcal{A}_{\mathcal{O}}$ which is a distinguisher over $k \sim k'$. We define a PTTM $\mathcal{A}'$ which on input $(m, \rho_r, 1^\eta)$:

- Splits ρ_r into three distinct infinite tapes ρ_{so}, ρ_{ra}, ρ_{ro}.
- Simulates $\mathcal{A}^{\mathcal{O}(\rho_{so},\rho_{ro})}(m, \rho_{ra}, 1^\eta)$.

Let us a prove that $\mathcal{A}'$ is a distinguisher over $k \sim k'$, which contradicts the unconditional soundness of this axiom when there is no oracle.

We denote by $\pi_{\overline{k}}(\rho_s, \eta)$ the tapes where every bit of ρ_s which does not correspond to a name of $\overline{k}$ is set to 0, and similarly $\pi_{\overline{k}^c}(\rho_s, \eta)$ where all bits for $\overline{k}$ are set to 0. We then have for any PTOM $\mathcal{A}_{\mathcal{O}}$:

$$\begin{aligned}
&\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}([\![\overline{k}]\!]^{\sigma,\eta}_{\rho_s}, \rho_r, 1^\eta) = 1\} \\
=^1\ &\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\pi_{\overline{k}}(\rho_s,\eta),\rho_{\mathcal{O}})}([\![\overline{n}]\!]^{\sigma,\eta}_{\pi_{\overline{k}^c}(\rho_s,\eta)}, \rho_r, 1^\eta) = 1\} \\
=^2\ &\mathbb{P}_{\rho_{s1},\rho_{s2},\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_{s1},\rho_{\mathcal{O}})}([\![\overline{n}]\!]^{\sigma,\eta}_{\rho_{s2}}, \rho_r, 1^\eta) = 1\} \\
=^3\ &\mathbb{P}_{\rho_{so},\rho_s,\rho_{ra},\rho_{ro}}\{\mathcal{A}^{\mathcal{O}(\rho_{so},\rho_{ro})}([\![\overline{k}]\!]^{\sigma,\eta}_{\rho_s}, \rho_{ra}, 1^\eta) = 1\} \\
=^4\ &\mathbb{P}_{\rho_s,\rho_r}\{\mathcal{A}'([\![\overline{k}]\!]^{\sigma,\eta}_{\rho_s}, \rho_r, 1^\eta) = 1\}
\end{aligned}$$

1. Thanks to the definition of support, the oracle answers the same on $\pi_{\overline{k}}(\rho_s, \eta)$ and ρ_s;
2. we split ρ_s in two, to replace independent tapes $\pi_{\overline{k}}(\rho_s, \eta)$ and $\pi_{\overline{k}^c}(\rho_s, \eta)$;
3. we rename random tapes;
4. by construction of $\mathcal{A}'$.

This shows that $\mathcal{A}'$ has the same advantage as $\mathcal{A}_{\mathcal{O}}$ against $k \sim k'$, which concludes the proof. ■

Other axioms in [BC14a] can be extended without problem. For instance the transitivity of $\sim$ or the function application axiom:

Lemma 6.2. *For any $\mathcal{O}$, $f \in \mathcal{F}$, terms $t_1, \ldots, t_n$, $u_1, \ldots, u_n$*

$$t_1, \ldots, t_n \sim u_1, \ldots, u_n \implies f(t_1, \ldots, t_n) \sim f(u_1, \ldots, u_n)$$

is $\mathcal{O}$ sound.

In general, what we have is that any axiom independent from the oracle support is sound.

Lemma 6.3. *For any $\mathcal{O}$, and terms t, s, such that all names in t, s do not appear in $\mathsf{supp}(\mathcal{O})$, we have that $t \sim s$ is sound if and only if $t \sim s$ is $\mathcal{O}$-sound.*

This allows us to derive, given an oracle and a recursive set of axiom, the set of axioms which is sound w.r.t. an oracle.

For instance, the general DDH axiom is, for any names a, b, c, $g^a, g^b, g^{ab} \sim g^a, g^b, g^c$. If we denote by $\overline{s}$ the support of some oracle, the $\mathcal{O}$-sound DDH version is simply the set of formulas $DDH_{\overline{s}}$ for all name $a, b, c \notin \overline{s}$, $g^a, g^b, g^{ab} \sim g^a, g^b, g^c$. Here, the notation g^x corresponds to $g(n)^{r(x)}$, where g is the function which extracts a group generator and r the function which evaluates names into exponents. We may consider that we have two interpretations of those function such that DDH holds.

EUF-CMA We define a BC version of the tagged EUF-CMA axiom. It is a direct adaptation of the BC EUF-CMA axiom (Definition 2.17) to match the behaviour of the tagged EUF-CMA axiom (Figure 5.1).

Definition 6.4. Given a name sk and a function symbol T, we define the generic axiom scheme EUF-CMA$_{T,sk}$ as, for any term t such that sk is only in key position:

$$\begin{array}{l}\textbf{if } (\mathtt{checksign}(t, pk(sk))) \\ \quad \textbf{then } T(\mathtt{getmess}(t)) \\ \qquad \dot{\bigvee}_{\mathtt{sign}(x,sk)\in \mathtt{St}(t)} (t \doteq \mathtt{sign}(x, sk)) \\ \quad \textbf{else } \top\end{array} \quad \sim \top$$

The tagged signing oracles is defined as previously, only adding the extra argument to the tagging function.

Definition 6.5. Given a name sk and a function T, we define the generic signing oracle $\mathcal{O}^{sign}_{T,sk}$ as follows:

$$\mathcal{O}^{\mathtt{sign}}_{T,sk}(m) := \textbf{if } T(m) \textbf{ then } \text{output}(\mathtt{sign}(m, sk)))$$

Proposition 6.1. *For any computational model in which the interpretation of* $\mathtt{sign}$ *is* EUF-CMA*, any name sk, and any boolean function T,* EUF-CMA$_{T,sk}$ *is* $\mathcal{O}^{\mathtt{sign}}_{T,sk}$*-sound.*

Proof. Let us assume that soundness is violated. We then have a term t and a computational model such that t does not satisfy EUF-CMA$_{T,sk}$. It means that the formula on the left hand side holds. As in t the secret key sk only occurs in key positions, we can simulate t by sampling all names, performing applications of function symbols, and sometimes calling the oracle $\mathcal{O}^{\mathtt{sign}}_{sk}$ to obtain a signature. t may also depend on attacker function symbols that have access to an oracle $\mathcal{O}^{\mathtt{sign}}_{T,sk}$. Thus, we can build a PTOM $\mathcal{A}^{\mathcal{O}^{\mathtt{sign}}_{T,sk},\mathcal{O}^{\mathtt{sign}}_{sk}}$ that produces exactly the same distribution of t for any fixed value of sk.

Let $\mathcal{B}^{\mathcal{O}^{\mathtt{sign}}_{sk}}$ be the PTOM which:

- simulates $\mathcal{A}^{\mathcal{O}^{\mathtt{sign}}_{T,sk}}$, by sampling all names itself, except sk;
- for every call made by A to $\mathcal{O}^{\mathtt{sign}}_{T,sk}$ with input m, $\mathcal{B}$ checks that $T(M)$ holds, and if it is the case query the signing oracle to get the signature, else fails.

The probability distribution of $\mathcal{B}^{\mathcal{O}^{\mathtt{sign}}_{sk}}$ is exactly the same as $\mathcal{A}^{\mathcal{O}^{\mathtt{sign}}_{T,sk},\mathcal{O}^{\mathtt{sign}}_{sk}}$, so $\mathcal{B}^{\mathcal{O}^{\mathtt{sign}}_{sk}}$ also produces an output o which violates the EUF-CMA$_{T,sk}$ axiom. We thus have that o is a valid signature, and is either not well tagged or does not correspond to a sub-term of t.

As all calls to $\mathcal{O}^{\mathtt{sign}}_{sk}$ made by $\mathcal{B}$ either correspond to a well tagged message or to a sub term of t, we know that o does not correspond to a signature produced by the signing oracle. $\mathcal{B}^{\mathcal{O}^{\mathtt{sign}}_{sk}}$ is thus an attacker which given access to a signing oracle can produce a signature for a message not signed by the oracle, i.e., an attacker which can win the EUF-CMA axiom. ■

6.3 Computational Soundness

Section Summary

We prove the computational soundness of the logic extended with oracles. It means that the extension of the BC logic still allows to derive computational guarantees, and is thus suited for our composition framework from Chapter 5.

We first prove the fact that the logic indeed allows to perform proofs of indistinguishability. Intuitively, if there exists a distinguisher in the computational model, the distinguisher wins with overwhelming probability on a specific trace. Indeed, as there is a finite number of traces, the advantage cannot be negligible over all of them. Then, given a specific trace, we can cut the distinguisher into multiple pieces, in order to construct a computational model which contradicts the BC indistinguishability of the frames corresponding to this trace.

Lemma 6.4. *Given two simple protocols P, Q with the same set T of observable traces, random tapes ρ_r, ρ_s, and oracle $\mathcal{O}$ and a functional model $\mathcal{M}_f$, we have:*

$$\forall \tau \in T, \forall \mathcal{M} \supset \mathcal{M}_f. \quad \mathcal{M} \models_{\mathcal{O}} \phi_P^\tau \sim \phi_Q^\tau \\ \Rightarrow \\ P \cong_{\mathcal{O}} Q$$

Proof. Let us proceed by contradiction. We are given a distinguisher $\mathcal{B}^{\mathcal{O}}$ against $P \cong_{\mathcal{O}} Q$. We thus have,

$$\mathrm{Adv}_{\mathcal{B}^{\mathcal{O}}}^{P \cong Q} = |\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = 1\} - \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = 1\}|$$

is non negligible. We must find τ and $\mathcal{M}$ such that $\mathcal{M} \not\models_{\mathcal{O}} \phi_P^\tau \sim \phi_Q^\tau$

Find a trace τ We split this probability, by conditioning over the observable trace (Definition 2.14) is executed by $\mathcal{B}$. We denote by E_τ the event "τ is the scheduling produced by $\mathcal{B}$".

By dichotomy, we have

$$\begin{aligned} \mathrm{Adv}_{\mathcal{B}^{\mathcal{O}}}^{P \cong Q} &= |\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = 1\}\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = 1\}| \\ &= \textstyle\sum_{\tau \in T} \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{E_\tau\} \times \\ &\quad |\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = 1 \mid E_\tau\} - \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = 1 \mid E_\tau\}| \end{aligned}$$

We of course have that for any τ, $\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{E_\tau\} \leq 1$. We thus obtained the following upper-bound.

$$\mathrm{Adv}_{\mathcal{B}^{\mathcal{O}}}^{P \cong Q} \leq \textstyle\sum_{\tau \in T} |\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = 1 \mid E_\tau\} - \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = 1 \mid E_\tau\}|$$

As the advantage is overwhelming, so is the sum. Recall that as P and Q are simple, they are finite, and in particular T is simple. Now, we classically have that a finite sum is overwhelming if and only if one of its element is overwhelming. Thus, there exists a trace τ such that $|\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = 1 \mid E_\tau\} - \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = 1 \mid E_\tau\}|$ is overwhelming.

Build the model $\mathcal{M}$ Let us consider this τ of length n given. We denote by $m_0, \ldots, m_n$ the consecutive calls of $\mathcal{B}$ to the oracle, once the scheduling is removed. Consider now the PTOM $\mathcal{A}_{g_k}^{\mathcal{O}}$, that, on input $b_1, \ldots, b_k, \eta, \rho_r$, executes the same code as $\mathcal{B}$, but

- replaces the ith call to the oracle $\mathcal{O}_P$ (resp. $\mathcal{O}_Q$), $i \leq k$, using b_i instead of the oracle reply;

- if the scheduling of $\mathcal{B}$ does not follow τ, stop and return some arbitrary fixed value.

The result of $\mathcal{A}^{\mathcal{O}}_{g_k}$ is then what $\mathcal{B}$ would have queried at the $k+1$ oracle call, if $\mathcal{B}$ is currently following trace τ. By their direct definition, $\phi^{\tau}_P = t_1, \ldots, t_n$ is the sequence of terms produced by the protocol, i.e., the protocol oracle, when interacting with $\mathcal{B}$. Thus, in the computational model $\mathcal{M}$ given by the $\mathcal{A}_{g_k}$, we have that given $\rho_r, \rho_s, \rho_{\mathcal{O}}$, for any η and k (less or equal to the length of τ, if $\mathcal{B}$ follows τ, then $\mathcal{A}^{\mathcal{O}}_{g_k}([\![t_1]\!]^{\eta}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}}, \ldots, [\![t_k]\!]^{\eta}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}}) = m_k$.

Thus, we have that whenever we condition over event E_τ, $\mathcal{B}$ and $\mathcal{A}_{g_n}$ distinguish with the same probability, i.e.,

$$|\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}}_{g_n}([\![\phi^{\tau}_P]\!]^{\sigma,\eta}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}}, \rho_r, 1^\eta) = 1 \mid E_\tau\} - \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}([\![\phi^{\tau}_Q]\!]^{\sigma,\eta}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}}, \rho_r, 1^\eta) = 1 \mid E_\tau\}|$$
$$= |\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_P}(\rho_r, 1^\eta) = 1 \mid E_\tau\} - \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O},\mathcal{O}_Q}(\rho_r, 1^\eta) = 1 \mid E_\tau\}|$$

Thus, both of these terms are overwhelming, and we have that

$$|\mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}}_{g_n}([\![\phi^{\tau}_P]\!]^{\sigma,\eta}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}}, \rho_r, 1^\eta) = 1\} - \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{\mathcal{A}([\![\phi^{\tau}_Q]\!]^{\sigma,\eta}_{\mathcal{M},\rho_s,\rho_r,\rho_{\mathcal{O}}}, \rho_r, 1^\eta) = 1\}|$$

is also overwhelming. In other terms, $\mathcal{M}$ is a computational model such that $\mathcal{M} \not\models_{\mathcal{O}} \phi^{\tau}_P \sim \phi^{\tau}_Q$, which concludes the proof. ■

Technical Details

Notice that compared to [BC14a], we do not prove the completeness at this step. This is for concision, but we do not actually lose much. Indeed, we lose completeness later on, as we will often use axioms that are not complete. We focus here on proof finding, rather than attack finding. Remark nonetheless that trying to perform a BC proof often allows to identify missing axioms, as it is completely formal, and thus potential attacks.

We finally have a computational soundness result. We write with the classical notation $Ax \models \phi$ if the set of formulas Ax and the formula $\neg\phi$ are inconsistent, i.e., if no model can satisfy both of them at the same time.

Theorem 6.1. *Given P, Q two protocols, $\mathcal{O}$ an oracle, A a set of axioms, $\mathcal{M}^f$ a functional model we assume that:*

- *A is $\mathcal{O}$-sound w.r.t. $\mathbb{F} = \{\mathcal{M} \supset \mathcal{M}^f\}$;*
- *for all $\tau \in T$, $A \models \phi^{\tau}_P \sim \phi^{\tau}_Q$.*

Then $P \cong_{\mathcal{O}} Q$.

Proof. Let us assume that we have a distinguisher for $P \cong_{\mathcal{O}} Q$, but that A is $\mathcal{O}$-sound.

By Lemma 6.4 we have a computational model $\mathcal{M} \supset \mathcal{M}^f$ and a trace τ such that $\mathcal{M} \models_{\mathcal{O}} \phi^{\tau}_P \not\sim \phi^{\tau}_Q$. As A is $\mathcal{O}$-sound, we also have $\mathcal{M} \models_{\mathcal{O}} A$, and this contradicts the fact that the formulas are inconsistent, i.e., that $A \models \phi^{\tau}_P \sim \phi^{\tau}_Q$. ■

We reduce computational indistinguishability to an inconsistency proof on the one hand and a soundness proof of the axioms on the other hand.

6.4 Extension to the Model for Unbounded Replication

Section Summary

Recall that for unbounded replications, we used notations such as $x \notin \overline{s}$, for infinite sequences of names $\overline{s}$. While the previous extension is enough to handle our composition results, we need for our applications to key exchanges to be able to express formally those predicates. To this end, for any name n of arity l, we give a formal interpretation to $\overline{n}$, that intuitively models the sequence of names $n_{1,\dots,1}, \dots, n_{r_1,\dots,r_l}$ of length polynomial in the security parameter.

We define the syntax and provide variations of the axioms that can be used to reason in this context. We then provide the concrete semantics so that these axioms are sound as technical details.

Technical Details

We provide a way to support infinite sequences in the BC logic, but note that our composition framework does not always require infinite sequences. When considering basic key exchanges, it is enough to use cofinite sequences. Basically, if the property

$$KE_0[\textbf{if } x^I_{lsid} = lsid^R_0 \textbf{ then out}(k) \textbf{ else out}(x^I_0), \textbf{if } x^R_{lsid} = lsid^I_0 \textbf{ then out}(k) \textbf{ else out}(x^R_0)]$$

holds even when the attacker can simulate corrupted sessions, it is enough to derive the security of multiple sessions. It is interesting, as this property does not rely on infinite sequences.

To understand this, let us briefly consider a basic unsigned Diffie Hellman key exchange. It must of course not verify the previous property. The exchange shares are g^{a_0}, g^{b_0}. To break the previous property, we can give as a share to I the correct g^{b_0}, I will then produce depending on the side k or $g^{a_0 b_0}$. If we provide R with $g^{a_0} \times g^{a_0}$, R does not believe to be paired with I and it then always output as key $g^{2a_0 b_0}$. One can then easily distinguish if the output of R is the square of the output of I.

Basically, this stems from the fact that always outputting the actual key leaks information to the attacker when agents are not paired together.

For key exchanges with key confirmation, we wish to test the real or random before we have any authentication (as the authentication may come from the key confirmation). So if we always leak the key of the agent, the property will not be verified. However, we do need to leak the key to enable to go from one session to multiple sessions (to give the attacker enough information for the simulatability). The idea is then, as expressed in the previous Theorems, to only leak the key when two "honest" parties are paired together. Else, we execute the key confirmation, which should fail. Here, we have an explicit need to be able to test which sessions are honest, whether they are corrupted or not, and this for an unbounded number of sessions. Hence the need for a test based on infinite sequences.

Syntax Recall that names are defined with an arity (Figure 2.1), where a name n of index arity l can be indexed by l integers, yielding a distinct copy of the name for each indexes. Moreover, in a protocol, the index variables occurring in names must all be bound through a parallel or a sequential binder, and thus once we consider the term corresponding to the protocol in the BC logic, all names appear without index variables.

For any name n of index arity l, the syntax of terms in the BC logic only contained all the copies $n_{k_1,\ldots,k_l}$ for $k_1,\ldots,k_l \in \mathbb{N}$ as symbols of arity 0 (a constants of the term algebra). For each name n, we add to the syntax of terms the symbol seq_n of arity 0. We also provide a function symbol $\in$ using infix notation, so that $t \in \mathsf{seq}_n$ is now in the syntax.

Axioms The classical α-renaming axiom still holds, but all copies of a name are renamed at once. Thus, for any sequences of terms $\overline{t}$, and any names n, n' of index arity l such that n' does not occur in t, we have:

$$(1)\ \overline{t} \sim \overline{t}\{\mathsf{seq}_n \mapsto \mathsf{seq}_{n'}\} \cup \{n_{k_1,\ldots,k_l} \mapsto n'_{k_1,\ldots,k_l} \mid k_1,\ldots,k_l \in \mathbb{N}\}$$

Furthermore, we also provide axioms that allow to reason about the membership predicate, defined as:

(2) $n_{k_1,\ldots,k_l} \in \mathsf{seq}_n \sim \mathsf{true}$ for any name n and all $k_1,\ldots,k_l \in \mathbb{N}$;
(3) $n'_{k_1,\ldots,k_l} \in \mathsf{seq}_n \sim \mathsf{false}$ for any name n' distinct of n and all $k_1,\ldots,k_l \in \mathbb{N}$.

Remark that as $\in$ is a boolean function symbol, it is in contradiction with its negation and we trivially have that that for any term t and name n,

$$t \in \mathsf{seq}_n \wedge t \notin \mathsf{seq}_n \sim \mathsf{false}$$

This is actually what is used in our proofs of indistinguishability, as tagged oracles in our applications provide messages m such that we have $f(m) \in \mathsf{seq}_n$ for some function f, and the security property raises bad if $f(m) \notin \mathsf{seq}_n$.

Technical Details

Semantics The idea is that seq_n should model all sequences $\mathsf{seq}_n = \{n_1,\ldots,n_{p(\eta)}\}$ for any polynomial p. Then, if an indistinguishability holds for all such sequences for all polynomials, it also holds when the polynomial is bigger than the running time of the distinguisher, and the sequence then models an infinite sequence. To model this, the interpretation of a term t may now depend on some polynomial p with one indeterminate and with positive integer coefficients given to the PTTMs, and the interpretation is denoted $[\![t]\!]^{\eta,\sigma}_{\mathcal{M},p,\rho_s,\rho_r,\rho_\mathcal{O}}$.

The indistinguishability predicate $\sim$ is now interpreted as indistinguishability for all distinguishers and all polynomials p. Definition 6.2 now becomes:

Definition 6.6. Given a computational model $\mathcal{M}$, including an oracle $\mathcal{O}$, two sequences of terms $\overline{t}$, $\overline{u}$, and an assignment σ of the free variables of $\overline{t}, \overline{u}$ to ground terms, we have $\mathcal{M}, \sigma \models_\mathcal{O} \overline{t} \sim \overline{u}$ if, for any strictly increasing polynomial p and every polynomial time oracle Turing machine $\mathcal{A}^\mathcal{O}$,

$$\begin{aligned}|&\mathbb{P}_{\rho_s,\rho_r,\rho_\mathcal{O}}\{\mathcal{A}^{\mathcal{O}(p,\rho_s,\rho_\mathcal{O})}([\![\overline{t}]\!]^{\sigma,\eta}_{\mathcal{M},p,\rho_s;\rho_r;\rho_\mathcal{O}}, \rho_r, 1^\eta) = 1\} \\ &-\mathbb{P}_{\rho_s,\rho_r,\rho_\mathcal{O}}\{\mathcal{A}^{\mathcal{O}(p,\rho_s,\rho_\mathcal{O})}([\![\overline{u}]\!]^{\sigma,\eta}_{\mathcal{M},p,\rho_s;\rho_r;\rho_\mathcal{O}}, \rho_r, 1^\eta) = 1\}|\end{aligned}$$

is negligible in η. Here, $\rho_s, \rho_r, \rho_\mathcal{O}$ are drawn according to a distribution such that every finite prefix is uniformly sampled.

So, we can now assume that the interpretation of terms may depend on a polynomial p. We previously assumed for a name n_i, that the functional model was providing a distinct Turing Machine for each copy of the name, i.e., a machine $\mathcal{A}_{n_k}$ for each $k \in \mathbb{N}$. However, to build a machine that can interpret seq_n, all the copies of the name must be extracted in a uniform way,

so that it is possible to collect all of them in polynomial time. To this end, we now consider that a functional model provides, for each name $n_{\tilde{i}}$ of index arity l, a Turing Machine $\mathcal{A}_n$ that takes as input the security parameter, the random tape ρ_S and l integers, and returns a sequence of bitstrings of length η extracted from ρ_s. Then, the interpretation of the name $n_{k_1,\ldots,k_l}$, with $k_1,\ldots,k_l \in \mathbb{N}$ is, given $\mathcal{M}$, η, σ, ρ_s , $\rho_{\mathcal{O}}$ and ρ_r.

$$[\![n_{k_1,\ldots,k_l}]\!]^{\eta,\sigma}_{\mathcal{M},p,\rho_s,\rho_r,\rho_{\mathcal{O}}} := \mathcal{A}_n(1^\eta, \rho_s, k_1, \ldots, k_l)$$

The set of all the $\mathcal{A}_n$ should use distinct parts of the random tape ρ_s, and each $\mathcal{A}_n$ should return distinct parts of the tape for each sequence of integers given as integers. This can be done for instance if ρ_s is seen as a folding of random tapes $\rho_{s,n}$ in a single tape, such that each $\mathcal{A}_n$ only accesses bits corresponds to $\rho_{s,n}$ through the inverse folding (this essentially corresponding to bijective mappings from $\mathbb{N}^k$ to $\mathbb{N}$). Then, for each sequence of integers $k_1,\ldots,k_l$, $\mathcal{A}_n$ extracts from $\rho_{s,n}$ a unique sequence of bits by computing a bijection f from $\mathbb{N}^l$ to $\mathbb{N}$, and extracting the bitstrings of length η at position $\eta \times f(k_1,\ldots,k_l)$.

Using this new interpretation for names, we now define the semantics of seq_n, for any name n of index arity l, as, given $\mathcal{M}$ (that now contains a polynomial p), η, σ, ρ_s, $\rho_{\mathcal{O}}$ and ρ_r,

$$[\![\mathsf{seq}_n]\!]^{\eta,\sigma}_{\mathcal{M},p,\rho_s,\rho_r,\rho_{\mathcal{O}}} := \mathcal{A}_{\mathsf{seq}_n}(1^\eta, p, \rho_s)$$

where $\mathcal{A}_{\mathsf{seq}_n}$ is the machine that:

- contains l nested loops over the l variables $c_1,\ldots,c_l$ all ranging from 1 to $p(\eta)$;
- at each iteration, simulate $\mathcal{A}_n(1^\eta, \rho_s, c_1,\ldots,c_l)$ and appends its result to the output tape.

Remark that given a model $\mathcal{M}$, and thus the machine $\mathcal{A}_n$, we completely fix the machine $\mathcal{A}_{\mathsf{seq}_n}$. Essentially, $\mathcal{A}_{\mathsf{seq}_n}$ will produce the sequence of bitstring corresponding to the interpretation of $n_{1,\ldots,1},\ldots,n_{p(\eta),\ldots,p(\eta)}$.

The BC axioms presented previously are still sound in this semantics. Essentially, this is because when the axiom scheme does not depend on any seq_n, all the occurrences of seq_n in terms satisfying the guards of the scheme can be simulated by an attacker who samples $p(\eta)$ randoms.

Lemma 6.5. *For any computational model in which the interpretation of* `sign` *is* EUF-CMA*, any name sk,* EUF-CMA$_{T,sk}$ *is* $\mathcal{O}^{\mathtt{sign}}_{T,sk}$*-sound even for terms that may depend on some* seq_n*.*

Proof. We have a term t, a computational model and a polynomial p such that the interpretation of t where all sequences seq_n are of length $p(\eta)$ contradicts the EUF-CMA$_{T,sk}$ axiom.

The proof is exactly the same as Proposition 6.1, as we can once again from t build a Turing Machine that samples all names but sk (and may thus sample $p(\eta)$ names for each sequence), and is then able to simulate all operations of t. ■

This means that we can safely consider a version of EUF-CMA$_{T,sk}$ where for instance $T(x)$ is of the form $x \in \mathsf{seq}_n$ and still have the soundness of the axiom. Remark that this proof would hold similarly for other cryptographic axioms.

We however have to prove the soundness of the axioms that are specific to seq.

Proposition 6.2. *Axioms (1),(2) and (3) are sound in all models where the interpretation of $\in$ is given by the machine $\mathcal{A}_\in(1^\eta, x_1, x_2)$ that checks if x_1 is a bitstring of length η and returns true if and only if x_1 is a sub-string of x_2 starting at a position which is a multiple of η.*

Proof.

1. The alpha-renaming axiom is sound, unconditionally. This is similar to the classical BC logic alpha-renaming axiom, which holds as all randomness for a given name (of any arity) are completely independent and uniform. Replacing all occurrences of a name by a another fresh one thus yields exactly the same distribution. In essence, we replace in the interpretation of $\overline{t}$ all occurrences of $\mathcal{A}_n$ and $\mathcal{A}_{\mathsf{seq}_n}$ by $\mathcal{A}_{n'}$ and $\mathcal{A}_{\mathsf{seq}_{n'}}$. As the machines for n' did not occur previously in the interpretation of $\overline{t}$, we indeed have that the machines of n and of n' produce the same independent distribution for the interpretation of $\overline{t}$.
2. Given $n_{k_1,\ldots,k_l}$ and seq_n, we have for any polynomial p strictly increasing that for η large enough, $k_i \leq p(\eta)$ for $1 \leq i \leq l$. Thus, for η large enough, the interpretation of seq_n contains the result of $\mathcal{A}_n(1^\eta, \rho_s, k_1, \ldots, k_l)$ (simulated by $\mathcal{A}_{\mathsf{seq}_n}$), and $\mathcal{A}_\in$ always output true. The advantage of any attacker then becomes 0 which is negligible.
3. The probability of collision between two sequences of bitstrings of length η is $\frac{1}{2^\eta}$. For any polynomial p, as seq_n is a uniform sampling of length $p(\eta) \times \eta$, and $n'_{k_1,\ldots,k_l}$ is an independent uniform sampling of length η, the probability that $n'_{k_1,\ldots,k_l}$ occurs in seq_n at a position which is a multiple of η is the probability $1 - (1 - \frac{1}{2^\eta})^{p(\eta)}$. Thus, $\mathcal{A}_\in$ will answer true with only a negligible probability.

■

As the interpretation $\mathcal{A}_\in$ given in the previous proposition corresponds to the interpretation required in the application to key exchanges (Section 5.6), we can indeed use those axioms in proofs of key exchange security.

Part III

Automated

In which we look at low level automation

7 Probabilistic Language and Problems

> The new always happens against the overwhelming odds of statistical laws and their probability, which for all practical, everyday purposes amounts to certainty; the new therefore always appears in the guise of a miracle.
>
> *(Hannah Arendt)*

7.1 Introduction

In Part II, and later in Part IV, we focus on simplifying the logical reasoning about protocols and the applications of cryptographic axioms. However, some protocols do not rely on cryptographic primitives, but rather on some properties of finite fields, groups and multi-linear maps, boolean operators, ... Proving the security of such protocols requires a systematic reasoning about the probability distributions of the messages produced by the protocol.

In this Part, we focus on this low-level reasoning by trying to derive automatically properties about the distribution of messages in a protocol. This automation could then be used to transform a low-level reasoning into a single proof step at the logical level. We study four probabilistic questions, *equivalence* and *up-to-bad*, classical properties that can be used in the security proof of a protocol, and *non-interference* and *differential privacy*, that are not directly related to protocols but are close to the two previous properties and are naturally studied together. Those properties can informally be described as follows:

- *equivalence* - it asks that the distributions of two messages are equal. If two messages follow the same distribution, one can be replaced by the other in a cryptographic proof.
- *non interference* - the security of a system may depend on the fact that a message does not leak any information about a secret. The system should then produce outputs with the same distribution, when running with two distinct secrets. This can also be seen as an *independence* property, where the output distribution should be independent from the secret.
- *up-to-bad* - given a system, we may need to verify that the probability of an event corresponding to a security breach is small, for instance that the secret is leaked with only a very small probability.
- *differential privacy* - it quantifies the privacy of a system: roughly, an algorithm is differentially private if when its inputs are close, its outputs are close. Intuitively, it means that the algorithm does not depend strongly on small changes over its input. If the input is a collection of private date about users, it means that the algorithm does not depend strongly on the information of a specific user.

Many cryptographic systems are based on probabilistic arithmetic circuit (circuits that encode operations over finite fields), or on small probabilistic programs that operate over booleans or bitstrings. To have a foundational approach, we translate the four previous questions into two relational properties between simple probabilistic programs, i.e., programs that operates over a domain D, and given a set of inputs in D, can perform some random samplings over D, some

operations (e.g., and, xor, addition, multiplication, conditional branchings), and finally return a tuple of values in D.

Given a probabilistic program P that expects m inputs and produces n outputs, an input $\vec{i} \in D^m$ and a possible output $\vec{o} \in D^n$, we denote by $[P]_D^{\vec{i}}(\vec{o})$ the probability that the output of P is equal to $\vec{o}$ when P takes as input $\vec{i}$. In this Part, the two relational properties between probabilistic programs that we consider are:

- *D-equivalence* (denoted by $P_1 \approx_D P_2$) requires that P_1 and P_2 define the same distributions, i.e., for every input $\vec{i} \in D^m$ and possible output $\vec{o} \in D_k^n$,

$$[P_1]_D^{\vec{i}}(\vec{o}) = [P_2]_D^{\vec{i}}(\vec{o})$$

- *D-majority* requires that for a fixed $r \in \mathbb{Q}$, and for every input $\vec{i} \in D^m$ and output $\vec{o} \in D^n$, we have

$$[P_1]_D^{\vec{i}}(\vec{o}) \leq r \cdot [P_2]_D^{\vec{i}}(\vec{o})$$

  D-0-majority (denoted by $P_1 \prec_D^r P_2$) is a variant of majority, where we only consider the output $b = 0^n$, rather than quantifying over all outputs.

The relationships between the previous security questions and our the two relational properties can be explained informally as follows:

- probabilistic non-interference: for simplicity, assume that P has two inputs x (secret) and y (public), and a single (public) output. For every x, let P_x be the unique program such that $P_x(y) = P(x, y)$. Then P is non-interfering iff for every x_1 and x_2, the two programs P_{x_1} and P_{x_2} are equivalent.
- up-to-bad: given a program P and an event E, we can produce a program P' such that the probability of the program being equal to zero is the probability of E in P. Then, we can bound the probability of E in P with 0-majority over P'.
- differential privacy: for simplicity consider the case where the domain is $\mathbb{F}_2$, i.e., the booleans. For every program P with n inputs, define the residual programs $P_{i,0}$ and $P_{i,1}$ obtained by fixing the i-th output to 0 and 1 respectively. Then the program P is $log(r)$-differentially private iff for every i, $P_{i,0}$ and $P_{i,1}$ (and $P_{i,1}$ and $P_{i,0}$) satisfy r-majority.

Equivalence and majority can then be used to reason about security problems, for instance to prove that two boolean programs are equivalent. However, in many cases, recall that security systems are parameterized by a security parameter η, and we must prove that the system is secure asymptotically w.r.t. η. From the point of view of the computational indistinguishability from Section 2.3.2, let c be a channel identifier and s, t terms over an arbitrary signature. When we reason about the indistinguishability property $\mathbf{out}(c, s) \cong \mathbf{out}(c, t)$, we look at all possible interpretations of s and t for all possible η. If s and t are bitstrings, η may correspond to the length of the bitstring. If they model elements of a finite field, the η may correspond to the size of the finite field. This introduces a new problem: to use the fact that two terms have the same distribution in an indistinguishability proof, we must actually prove that the two terms have the same distribution for all possible η.

We thus define, based on the previous relational properties, two *universal* variant, that we call universal equivalence and universal majority. In this setting, programs do not depend on a single interpretation domain D, but can depend on a family of interpretation $\{D_k\}_{k\in\mathbb{N}}$. For the case of finite fields, we would for instance consider the family of interpretations $\{\mathbb{F}_{q^k}\}_{k\in\mathbb{N}}$, e.g., the family of all extensions of a finite field. Formally, the universal variants are defined as follows, given a family $\{D_k\}_{k\in\mathbb{N}}$:

- D_∞*-equivalence* requires the property to hold on all domains of the family, i.e.,

$$P_1 \approx_{D_\infty} P_2 \text{ iff } \forall k.\ P_1 \approx_{D_k} P_2$$

- D_∞-*0-majority* requires the property to hold on all extensions of a field, i.e.,

$$P_1 \prec^r_{D_\infty} P_2 \text{ iff } \forall k.\ P_1 \prec^r_{D_k} P_2$$

Remark that the universal case is not a trivial extension of the fixed case. The following two programs, whose execution is parameterized by the domain and expressed using uniform sampling from the domain ($x \xleftarrow{\$} \mathbb{D}$) and a return instruction (return), illustrate the difference between equivalence and universal equivalence.

Example 7.1.

$$P_1 = x \xleftarrow{\$} \mathbb{D};\ \mathsf{return}\ (x^2 + x) \qquad P_2 = \mathsf{return}\ 0$$

are 2- but not 2^2-*equivalent*, and hence not 2^∞-equivalent. Indeed, when instantiating $\mathbb{D}$ with $\mathbb{F}_2$, the left hand side program simply evaluates to zero, which is not the case with $\mathbb{F}_4$. On the other hand, the programs

$$Q_1 = x \xleftarrow{\$} \mathbb{D};\ \mathsf{return}\ (x) \qquad Q_2 = x \xleftarrow{\$} \mathbb{D};\ \mathsf{return}\ (x + 1)$$

are $\mathbb{F}_{q^\infty}$-equivalent for any prime power q as both programs define the uniform distribution, whatever finite field is used for the interpretation of $\mathbb{D}$. These examples also illustrate the difference with the well-studied polynomial identity testing (PIT) problem, as the first two programs are 2-equivalent, while PIT does not consider $x^2 + x$ and 0 to be equal on $\mathbb{F}_2$, nor would Q_1 and Q_2 be considered identical.

In this Part, we investigate both theoretical and practical aspects of the fixed and the universal case, first studying the complexity and decidability of the probabilistic problems, and then trying to derive heuristics usable in practice.

Chapter Summary

We introduce syntax and semantics for probabilistic programs parameterized by an interpretation domain D. Our programming languages cover programs that arise in different application domains, primarily security and privacy, but potentially also in machine learning and algorithmic fairness.

Based on the probabilistic semantics, we formally define multiple relational properties that have direct applications in cryptography: they can be seen as low level proof steps inside protocol security proofs. Each of these properties is lifted to its universal version, where the property must hold on all elements of a family of interpretations, e.g., the family of bitstrings of all length.

As preliminaries to the next two Chapters, we study the links between those properties, showing that independence and equivalence are inter-reducible, and that in the case of equivalence one can consider programs without inputs, and only study the sub case where we ask if a program follows the uniform distribution. We finally introduce a generic semantic characterization of equivalence.

7.1.1 Our Contributions

We set the foundations for the study of the complexity and decidability of multiple relational properties. For their practical study through heuristics, we will leverage classical symbolic methods such as deducibility. To this end, after introducing the programming language, we formally

define its semantics and each of the properties. Like many other probabilistic programming languages, our language supports sampling from distributions, and conditioning distributions on an event[1]. Sampling is interpreted using the uniform distribution over sets defined by assertions, and branching and conditioning are relative to assertions.

The semantics are parameterized by the signature used to build terms and the interpretation domain. Although we will often consider in practice programs over finite fields, working over an abstract domain allows to derive more general results.

In this parameterized setting, we show under which conditions equivalence and independence are inter-reducible. We further reduce equivalence to equivalence over programs without inputs, and then to deciding if a program follows the uniform distribution. Those links provide insights and first intuitions about the nature of problems we are considering. We give a summary of the reductions in Figure 7.2.

We conclude by providing a semantic characterization of equivalence: two programs are equivalent if and only if there exists a bijective mapping between their random samplings such that they become equal point by point.

Limitations

Our probabilistic language does not support loops. We show in Chapter 8 that the addition of loops makes universal equivalence undecidable in the case of finite fields.

Moreover, our programs cannot perform samplings from non uniform distributions. This is because most of our results rely on a link between the probability of some event and counting the number of samplings such that the event happens. This limitation is slightly mitigated, as some non uniform distribution can be constructed using multiple uniform variables and dedicated function symbols.

7.1.2 Related Work

There are many works regarding semantics of probabilistic languages. In this Chapter, we rely on the formalism of [BGV18a], notably to handle the conditioning.

Fixed equivalence There is a vast amount of literature on proving equivalence of probabilistic programs. We only review the most relevant work here.

Murawski and Ouaknine [MO05] prove decidability of equivalence of second-order terms in probabilistic ALGOL. Their proof is based on a fully abstract game semantics and a connection between program equivalence and equivalence of probabilistic automata.

Legay *et al* [LMO+08] prove decidability of equivalence for a probabilistic programming language over finite sets. Their language supports sampling from non-uniform distributions, loops, procedure calls, and open code, but not conditioning. They show that program equivalence can be reduced to language equivalence for probabilistic automata, which can be decided in polynomial time.

[1] Conditionings over an event is classical construct of probabilistic languages, that allows to condition the distribution of the output over some event. It corresponds to only considering the distribution of the outputs over the execution that satisfy the event.

Barthe *et al* [BGZ09] develop a relational program logic for probabilistic programs without conditioning. Their logic has been used extensively for proving program equivalence, with applications in provable security and side-channel analysis.

Universal equivalence Without formalizing the question as a general problem, the case of linear programs (boolean programs with only XOR operations) is studied in [BDK+10]. The authors propose a decision procedure for universal equivalence based on the classic XOR-lemma [CGH+85].

The case of linear programs with random oracles is considered in [CR16]. The authors give a polynomial time decision procedure for computational indistinguishability of two inputless programs. Their proof is based on linear algebra.

Majority problems The closest related work develops methods for proving differential privacy or for quantifying information flow.

Frederikson and Jha [FJ14] develop an abstract decision procedure for satisfiability modulo counting, and then use a concrete instantiation of their procedure for checking representative examples from multi-party computation.

Barthe et al [BCJ+19] show decidability of ϵ-differential privacy for a restricted class of programs. They allow loops and sampling from Laplace distributions, but impose several other constraints on programs. An important aspect of their work is that programs are parameterized by $\epsilon > 0$, so their decision procedure establishes ϵ-differential privacy for all values of ϵ. Technically, their decision procedure relies on the decidability of a fragment of the reals with exponentials by McCallum and Weispfenning [MW12].

Strongly linked to probabilistic non-interference, masking is a countermeasure based on secret sharing used to protect arithmetic programs against differential power analysis. There exist generic masking compilers that take as input an arithmetic program P and output a masked program P_k, where the masking order k is a parameter corresponding to the desired level of protection. The parameterized program P_k uses for loops; however, for every fixed value of k, one can unroll loops in P_k to obtain a program in our language. Over the last few years, there has been an active line of work to prove masking automatically using type systems, relational logics and model counting [KR19]. All these works target verification for a fixed k. It would be interesting to obtain decidability results for the parameterized verification problem. However the interpretation of P_k is over a fixed field $\mathbb{F}$, for all values of k. Therefore, the problem has a distinct flavour from ours.

7.2 Probabilistic Programming Language

Section Summary

We define a probabilistic programming language, based on terms built over a signature Σ equipped with an interpretation, called a Σ-algebra D. We first give a general surface language, and then consider a simpler core language. We define for this core language deterministic and probabilistic semantics.

Recall that a signature Σ is an indexed set of function symbols with their arity. Given a set $\mathcal{X}$ of variables, the set $\mathcal{T}_\Sigma(\mathcal{X})$ of terms is defined inductively as previously (see Figure 7.1).

t	::=	x	variables
	|	$f(t_1, \ldots, t_n)$	function of arity n
b	::=		*boolean conditions*
	|	$t_1 = t_2$	atomic formula
	|	$b_1 \wedge b_2$	and
	|	$b_1 \vee b_2$	or
	|	$\neg b$	not
e	::=		*program expressions*
	|	$x := t$	assignment
	|	$r_1, \ldots, r_m \xleftarrow{\$} \{X \in \mathbb{D}^m \mid b\}$	sampling
	|	observe b	observe
	|	$e_1; e_2$	sequential composition
	|	**if** b **then** e_1 **else** e_2	conditional branching
	|	return $(t_1, \ldots, t_n)$	return of arity n

Figure 7.1: Program Syntax

Definition 7.1. A Σ-algebra $\underline{D}$ for the signature Σ is given by a set D and the interpretation of Σ, which consists of a total function $f^D : D^n \mapsto D$ for each $f \in \Sigma$ of arity n.

For any ground term t, t^D corresponds to the interpretation of t defined inductively by $f(t_1, \ldots, t_n) = f^D(t_1^D, \ldots, t_n^D)$ for $f \in \Sigma$ of arity n.

We will always consider that Σ contains a constant 0, and that D provides a corresponding value also denoted by 0[2]. For a given interpretation domain D and a signature Σ, there is often a single natural Σ-algebra. In such cases, we denote $\underline{D}$ by simply D.

For instance, we often use $\Sigma_{\mathbb{F}_q} = \{+, \times\} \cup \mathbb{F}_q$ and instantiate D with $\mathbb{F}_q$. Remark that this is the simplest model of a finite field, where we directly integrate all the constants of the field as function symbols of the signature, rather than providing a minimal generating set of constants. We denote by $\mathbb{F}_q$ the (unique) finite field with q elements, where $q = p^k$ for some integer k and a prime p. The Σ-algebra $\underline{\mathbb{F}_q}$ (also denoted $\mathbb{F}_q$) corresponding to $\mathbb{F}_q$, instantiates multiplication and addition with the corresponding field operations. Given a polynomial $P \in \mathbb{F}_q[x_1, \ldots, x_m]$ and $X \in \mathbb{F}_{q^k}^m$, we denote by $P(X)$ the evaluation of P given X in $\mathbb{F}_{q^k}$.

7.2.1 Syntax and Informal Semantics

We consider a probabilistic programming language with sampling from subsets and conditionings, as well as a more pure, yet equi-expressive, core language that can encode all previous constructs and define its formal semantics.

We define in Figure 7.1 the syntax for probabilistic programs without loops nor recursion. Programs are parameterized by an abstract Σ-algebra $\mathbb{D}$, that can be instantiated by a Σ-algebra D. The expressions of our programs provide constructs for assigning a term t to a variable ($x := t$), as well as for randomly sampling values. The expression $r_1, \ldots, r_m \xleftarrow{\$} \{X \in \mathbb{D}^m \mid b\}$ uniformly samples

[2]This avoids the corner case of empty algebras.

m values from the set of m-tuples of values in $\mathbb{D}$ such that the condition b holds, and assigns them to variables $r_1, \ldots, r_m$. For example, $r \xleftarrow{\$} \{x \in \mathbb{D} \mid 0 = 0\}$ (which we often simply write $r \xleftarrow{\$} \mathbb{D}$) uniformly samples a random element in $\mathbb{D}$, while $r_1, r_2 \xleftarrow{\$} \{x_1, x_2 \in \mathbb{D}^2 \mid \neg(x_1 = 0)\}$ samples two random variables, ensuring that the first one is not 0. In the case of finite fields, note that the use of polynomial conditions allows to express any rational distribution over the base field $\mathbb{F}_q$, i.e., any distribution that assigns rational probabilities.

The construct $\mathsf{observe}\ b$ allows to condition the continuation by b: if b evaluates to false the program fails; the semantics of a program is the conditional distribution where b holds. Expressions also allow classical constructs for sequential composition, conditional branching and returning a result.

Given two disjoint sets of variables I and R, we denote by $\mathcal{P}_\Sigma(I, R)$ the set of well-formed programs, where a program P is well-formed if:

- variables in R are sampled only once;
- variables in R only appear in the program after they have been sampled;
- each branch of P ends with a return instruction that returns the same number n of elements; n is then called the arity of the program and denoted by $|P|$.

I is the set of free variables of the programs, that corresponds to input variables. We will often omit the Σ, when it is explicit from the context.

Example 7.2. Consider the following simple program over finite fields

$$inv(i) ::= \textbf{if}\ i = 0\ \textbf{then}\ \mathsf{return}\ 0\ \textbf{else}\ r \xleftarrow{\$} \mathbb{D}; \mathsf{observe}\ r \times i = 1; \mathsf{return}\ r$$

When $\mathbb{D}$ is instantiated by a finite field, this program defines a probabilistic algorithm for computing the inverse of a field element i. If i is 0, by convention the algorithm returns 0. Otherwise, the algorithm uniformly samples an element r. The observe instruction checks whether r is the inverse of i. If this is the case we return r, otherwise the program fails. As we will see below, our semantics normalizes the probability distribution to only account for non-failing executions. Hence, this algorithm will return the inverse of any positive i with probability 1. This is obviously not a practical procedure for computing an inverse, but we use it to illustrate the semantics of conditioning. Equivalently, this program can be written by directly conditioning the sample

$$inv'(i) ::= \textbf{if}\ i = 0\ \textbf{then}\ \mathsf{return}\ 0\ \textbf{else}\ r \xleftarrow{\$} \{x \in \mathbb{D} \mid x \times i = 1\}; \mathsf{return}\ r$$

7.2.2 A Core Language

While the above introduced syntax is convenient for writing programs, we introduce a more pure, core language that is actually equally expressive and ease the technical developments. To define this core language, we add an explicit failure instruction $\bot$, similarly to [BGV18b]. It allows us to get rid of conditioning in random samples and observe instructions. Looking ahead, and denoting by $[P]_D$ the semantics of the program P in D, we will have that

$$\begin{aligned} \left[r_1, \ldots, r_m \xleftarrow{\$} \{X \in \mathbb{D}^m \mid b\}; e\right]_D &= \left[r_1, \ldots, r_m \xleftarrow{\$} \mathbb{D}^m; \textbf{if}\ b\ \textbf{then}\ e\ \textbf{else}\ \bot\right]_D \text{ and} \\ [\mathsf{observe}\ b; e]_D &= [\textbf{if}\ b\ \textbf{then}\ e\ \textbf{else}\ \bot]_D \end{aligned}$$

Without loss of generality, we can inline deterministic assignments, and use code motion to perform all samplings eagerly. In other terms, we assume that all random samplings are performed upfront

at the beginning of the program. Therefore we can simply consider that each variable in R is implicitly uniformly sampled in D. Programs are then tuples of simplified expressions $(e_1, \dots, e_n)$ defined as follows.

e	::=		*simplified expressions*
	$\mid$	P	term
	$\mid$	$\bot$	failure
	$\mid$	**if** b **then** e_1 **else** e_2	conditional branching

We suppose that all nested tuples are flattened and write (P, Q) to denote the program which simply concatenates the outputs of P and Q. When clear from the context, we may also simply write $\vec{0}$ instead of the all zero tuple $(0, \dots, 0)$. We denote by $\overline{\mathcal{P}}(I, R)$ the set of functional programs, that are simply tuples of terms. In the case of finite fields, functional programs are arithmetic programs. Remark that functional programs cannot fail.

Technical Details

One may note that the translation from the surface language to the core language is not polynomial in general. Indeed, constructs of the form (**if** b **then** $x := t_1$ **else** $x := t_2; P$), i.e., sequential composition after a conditional, implies to propagate the branching over the assignment to all branches of P, and doubles the number of conditional branchings of P. All complexity results will be given for the size of the program given in the core language. Remark that in a functional style version of the surface language, where we replace $x := t$ by **let** $x = t$ **in** and removed sequential composition, the translation would however be polynomial. Similarly, for the class of programs without sequential composition after conditional branchings, the translation is also polynomial.

7.2.3 Semantics

We now define the semantics of our core language. The precise translation from the high level syntax previously presented and our core language is standard and omitted.

Deterministic semantics. We first define a *deterministic* semantics where all random samplings have already been defined.

For a set $\mathcal{X}$ of variables, with $t \in \mathcal{T}(\Sigma, \mathcal{X})$ and $\vec{v} \in D^{|X|}$, $t(\vec{v})$ denotes the evaluation of t in D, where $\mathcal{X}$ is then seen as an ordered tuple of variables. We choose to use the classical notation for polynomial functions, as we will often reason about finite fields. Variables in t are evaluated with the value given by $\vec{v}$, and each function symbol f is evaluated with f^D.

We also denote by $b(\vec{v})$ the evaluation of a boolean test, where all terms are evaluated according to $\vec{v}$. For a program $e \in \mathcal{P}(I, R)$ and $\vec{v} \in D^{|I \cup R|}$, we define the evaluation of e, denoted $[\![e]\!]^{\vec{v}}_D$, which is a value in $D^{|P|} \times \{\bot\}$:

$$
\begin{aligned}
[\![t]\!]^{\vec{v}}_D &= t(\vec{v}) \text{ where } t \in \mathcal{T}(\Sigma, I \uplus R) \\
[\![\bot]\!]^{\vec{v}}_D &= \bot \\
[\![\textbf{if } b \textbf{ then } e_1 \textbf{ else } e_2]\!]^{\vec{v}}_D &= \begin{cases} [\![e_1]\!]^{\vec{v}}_D \text{ if } b(\vec{v}) \text{ holds on } D \\ [\![e_2]\!]^{\vec{v}}_D \text{ if } b(\vec{v}) \text{ does not hold on } D \end{cases} \\
[\![(e_1, \dots, e_n)]\!]^{\vec{v}}_D &= \begin{cases} \bot \text{ if } [\![e_i]\!]^{\vec{v}}_D = \bot \text{ for some } i \\ ([\![e_1]\!]^{\vec{v}}_D, \dots, [\![e_n]\!]^{\vec{v}}_D) \text{ else} \end{cases}
\end{aligned}
$$

Intuitively, the set of executions corresponding to non failure executions represent the set of possible executions of the program. We next define probabilistic semantics by sampling uniformly the valuations of the random variables while conditioning on the fact that the program does not fail.

Remark that we explained everything for clarity, but we essentially reuse the semantics for terms of Chapter 2, that was denoted by $[\![t]\!]^\sigma$ for some substitution σ over the free variables of t. As it is now an important parameter, we make the domain of interpretation D explicit, and for concision, we denote $[\![t]\!]^{\vec{x}\mapsto\vec{v}}$ by $[\![t]\!]^{\vec{v}}_D$.

Probabilistic semantics. For any n, we denote by $\mathsf{Distr}(D^n)$ the set of distributions over D^n. For a program $P \in \mathcal{P}(I,R)$ with $|P| = n$, and $|I| = m$, we define its semantics, denoted by $[P]^{\vec{i}}_D$ to be the distribution of the output corresponding to the inputs $\vec{i}$. We assume that programs in $P \in \mathcal{P}(I,R)$ do not fail all the time, i.e., for any possible input and any program its probability of failure is strictly less than 1. For program P and input $\vec{i} \in D^m$, we set

$$[P]^{\vec{i}}_D \;:\; \vec{o} \mapsto \frac{\mathbb{P}_{\vec{r}\xleftarrow{\$} D^{|R|}}\{[\![P]\!]^{\vec{i},\vec{r}}_D = \vec{o}\}}{\mathbb{P}_{\vec{r}\xleftarrow{\$} D^{|R|}}\{[\![P]\!]^{\vec{i},\vec{r}}_D \neq \bot\}}$$

Note that the normalization by conditioning on non-failing programs is well defined as we supposed that programs do not always fail.

Example 7.3. Over $\overline{\mathcal{P}}_{\mathbb{F}_2}(\{x\},\{u,v,w\})$, let[3]

- $P_1 = x + v$
- $P_2 = xv$
- $P_3 = uv + vw + wu$

P_1 is the uniform distribution for any input x. In other terms, we have that $[P_1]^0_{\mathbb{F}_2}(0) = \frac{1}{2} = [P_1]^1_{\mathbb{F}_2}(0)$. However, P_2 is always equal to zero when x is null, i.e., $[P_2]^0_{\mathbb{F}_2}(0) = 1$. When, x is one, P_2 simply follows the uniform distribution and $[P_2]^1_{\mathbb{F}_2}(0) = \frac{1}{2}$.

$[\![P_3]\!]$ only depends on the random variables. We can easily compute its distribution by writing the truth table of the program:

u	0	0	0	0	1	1	1	1
v	0	0	1	1	0	0	1	1
w	0	1	0	1	0	1	0	1
P_3	0	0	0	1	0	1	1	1

We see that $[\![P_3]\!]^{\mathbb{F}_2}$ is actually the uniform distribution and we have that $[P_3]_{\mathbb{F}_2}(0) = \frac{1}{2}$.

[3] We denote $x \times v$ by xv.

7.3 Decision Problems and Universal Variants

Section Summary

We introduce formally several decision problems on probabilistic programs:

- D-equivalence, $P \approx_D Q$: two programs must produce the same distribution over all inputs;
- D-0-majority, $P \prec^r_D Q$: the probability that P equals to 0 divided by the probability that Q equals 0 is bounded by the rational r;
- D-independence, $\perp^Y_D (P_1, \ldots, P_n)$: the distributions of the n programs must be independent.

We introduce their respective universal variants, where for instance universal equivalence for a family of algebra $\{D_k\}_{k\in\mathbb{N}}$ asks that the D_k-equivalence hold for all k. We often reason over the family of all extensions of a finite field, i.e., $\{\mathbb{F}_{q^k}\}_{k\in\mathbb{N}}$. Each of those properties is associated with a decision problem.

Equivalence D-equivalence requires equality of distributions between the outputs of two programs for all possible inputs. Notably, two equivalent programs are indistinguishable by any attacker, and a program equivalent to the uniform distribution does not leak any information about its inputs to the attacker.

Definition 7.2. Two programs P_1 and P_2 are *D-equivalent*, denoted by $P_1 \approx_D P_2$, if P_1 and P_2 define the same distributions over all inputs, i.e., for every input $\vec{i} \in D^m$: $[P_1]^{\vec{i}}_D = [P_2]^{\vec{i}}_D$.

Example 7.4. Continuing Example 7.3, we have that $u+x \approx_{\mathbb{F}_2} u \approx_{\mathbb{F}_2} uv+vw+wu$, but $ux \not\approx_{\mathbb{F}_2} u$.

Majority D-majority bounds the quotient of the distributions of two programs by a rational. Our results focus on the case where we compare the two distributions on a single point. It can be used to decide vanilla (i.e. not approximate) ϵ-differential privacy, that quantifies the privacy leaked by some mechanism.

Definition 7.3. *D-majority* between two programs P_1, P_2 requires that for a fixed $r \in \mathbb{Q}$, and for every input $\vec{i} \in D^m$ and output $\vec{o} \in D^n$, we have

$$[P_1]^{\vec{i}}_D (\vec{o}) \leq r \cdot [P_2]^{\vec{i}}_D (\vec{o})$$

D-0-majority, denoted by $P \prec^r_D Q$, is the variant where we only consider the output $\vec{o} = 0^n$, rather than quantifying over all output.

We only provide a notation of D-0-majority, as our results for the general majority are very limited. Notably, D-majority in the case of $r = 1$ is the same as equivalence, and D-majority is in fact harder than equivalence.

Example 7.5. Over $\overline{\mathcal{P}}_{\mathbb{F}_2}(\{x\}, \{u, v, w\})$, we have that $[u + x]^1_{\mathbb{F}_2}(0) = [ux]^1_{\mathbb{F}_2}(0)$, but $[u + x]^0_{\mathbb{F}_2}(0) = \frac{1}{2}$ and $[ux]^0_{\mathbb{F}_2}(0) = 1$. Thus, we have that $ux \prec^2_{\mathbb{F}_2} u+x$. This means that the probability that ux is equal to zero, is always at most twice as big as the probability that $u + x$ equals zero. The closer to 1 the coefficient r is, the closer the two distributions are.

Independence A distribution is independent from a given variable if the distribution is the same for any fixed value of the variable. It implies that the distribution of one of the programs does not provide any information about the distribution of the other one. This can for instance be used to bound the advantage of an attacker.

Definition 7.4 (*D-conditional independence*)**.** Let $P_1, \ldots, P_n \in \mathcal{P}(I, R)$. Given $Y \subset R$, we say that $P_1, \ldots, P_n$ are independent conditioned by Y, denoted by $\perp_D^Y (P_1, \ldots, P_n)$, if:

$$\forall \vec{i} \in D^{|I|}.\ \forall \vec{i'} \in D^{|Y|}.\ [(P_1, \ldots, P_n)]_D^{\vec{i},\vec{i'}} = ([P_1]_D^{\vec{i},\vec{i'}}, \ldots, [P_n]_D^{\vec{i},\vec{i'}})$$

We write $\perp_D (P_1, \ldots, P_n)$ for $\perp_D^{\emptyset} (P_1, \ldots, P_n)$, which simply denotes independence of the programs.

Example 7.6. In particular, considering boolean programs in $\mathcal{P}_{\mathbb{F}_2}(\{i_1, i_2\}, \{r\})$, we have that $\perp_{\mathbb{F}_2} (i_1(i_2 + r), i_2)$, which means that $i_1(i_2 + r)$ leaks no information about i_2. However, $\not\perp_{\mathbb{F}_2} (i_1(i_2 + r), i_1)$.

Universal variants Given a fixed signature, one can provide distinct interpretations. For instance, given the signature associated to finite fields, for a given q, D can be instantiated by any $\mathbb{F}_{q^k}$. In cryptographic proofs, the power of the finite field is often a parameter that can be instantiated by any value. To leverage the equivalence between two programs in a cryptographic proof, we need to have the equivalence of the two programs for all possible $\mathbb{F}_{q^k}$. We define a variant of all our probabilistic relations, where the relation must hold over a family of Σ-algebras.

Definition 7.5. Let $\{D_k\}_{k\in\mathbb{N}}$ be a family Σ-algebra. For any relation R_D in $\{\approx_D, \prec_D^r, \perp_D^Y\}$, we define the corresponding universal relation, denoted by R_{D_∞}, to hold if for all k, R_{D_k} holds.

In the case of finite fields, when $D_k = \mathbb{F}_{q^k}$, we denote D_∞ by $\mathbb{F}_{q^\infty}$.

When we consider a finite interpretation D, all problems previously introduced are decidable in the non universal case: it is always possible to compute the distributions by enumerating all possibilities. For the universal case, decidability is however not trivial anymore. Remark that, for some programs, universal equivalence can be easily obtained.

Example 7.7. We have that $u + x \approx_{\mathbb{F}_{q^k}} u$ for any k, as adding a random variable to any fixed input always produces the uniform distribution. Thus, we have that $u + x \approx_{\mathbb{F}_{q^\infty}} u$.

Decision problems We define the corresponding decision problems, for $k \in \mathbb{N} \cup \{\infty\}$ (I, R are also always part of the input):

D_k-*equivalence*
INPUT: $P, Q \in \mathcal{P}(I, R)$
QUESTION: $P \approx_{D_k} Q$?

D_k-*majority*
INPUT: $P, Q \in \mathcal{P}(I, R)$, $r \in \mathbb{Q}$
QUESTION: $P \prec_{D_k}^r Q$?

D_k-*conditional independence*
INPUT: $P \in \mathcal{P}(I, R)$, $Y \subset R$
QUESTION: $\perp_{D_k}^Y P$?

7.4 First Results

Section Summary

We provide reductions between some of our problems, studying the links between equivalence and independence, and simpler version of equivalence. Finally, we provide a characterization of equivalence, that can be used to prove equivalence by providing an explicit bijection. In the following two chapters, those results will be used to derive the complexity or decidability of the associated decision problems, or provide multiple approaches to tackle the same problem.

7.4.1 Links between Problems

We now study several links between our problem. All omitted proofs can be found in Appendix C.1. We provide in Figure 7.2 a summary of the reductions provided in this Section.

For technical reasons, our later work requires that we introduce a slight generalization of equivalence, conditional equivalence, that allows for a simpler reasoning. *D-conditional equivalence* is a generalization of equivalence, where we require that the distributions of the programs are equal when conditioned by some other program being equal to zero.

Definition 7.6 (*D-conditional equivalence*)**.** Let $P_1, Q_1 \in \mathcal{P}(I,R)$ and $P_2, Q_2 \in \overline{\mathcal{P}}(I,R)$ with $|P_1| = |Q_1| = n$. We write $P_1 \mid P_2 \approx_D Q_1 \mid Q_2$, if:

$$\forall \vec{i} \in D^{|I|}.\ \forall \vec{o} \in D^n.\ [(P_1,P_2)]_D^{\vec{i}}(\vec{o},\vec{0}) = [(Q_1,Q_2)]_D^{\vec{i}}(\vec{o},\vec{0})$$

The universal version D_∞-*conditional equivalence* is defined similarly to D_∞-*equivalence*, and the associated decision problem is for $k \in \mathbb{N} \cup \{\infty\}$:

D_k-*conditional equivalence*
INPUT: $P_1, Q_1 \in \mathcal{P}(I,R)$, $P_2, Q_2 \in \overline{\mathcal{P}}(I,R)$
QUESTION: $P_1 \mid P_2 \approx_{D_k} Q_1 \mid Q_2$?

Note that conditional equivalence is a direct generalization of equivalence, as for $P, Q \in \mathcal{P}(I,R)$ and $k \in \mathbb{N} \cup \{\infty\}$, $P \approx_{D_k} Q$ if and only if $P \mid 0 \approx_{D_k} Q \mid 0$.

An interesting feature of equivalence is that inputs do not make the problem harder, as we can encode inputs using randoms that are concatenated to the output of the programs.

Lemma 7.1. *Let $P_1, Q_1 \in \mathcal{P}(I,R)$, $P_2, Q_2 \in \overline{\mathcal{P}}(I,R)$. When $\sigma : I \to R_I$ is the substitution that replaces each variable in I by a fresh random variable in R_I, we have:*

$$P_1 \mid P_2 \approx_{D_k} Q_1 \mid Q_2 \Leftrightarrow (P_1\sigma, R_I) \mid P_2\sigma \approx_{D_k} (Q_1\sigma, R_I) \mid Q_2\sigma$$

Sketch of Proof. We replace all input variables in I by fresh random variables in R_I. As we concatenate the "inputs" variables R_I to the outputs of both programs, whenever we consider the probability that one program is equal to some value, this value conditions the value of the "inputs". Asking that the probability of the two programs is equal on all given points then asks for each point, that given the value of the "inputs" variables fixed by the point, the probabilities are equal. This is the expected behaviour of equivalence for inputs. ■

Next, we see that conditional independence is in fact as easy as non-conditional independence.

Lemma 7.2. *Let $P_1, \ldots, P_n$ be programs over $\mathcal{P}(I,R)$, and $Y \subset R$.*

$$\perp_{D_k}^{Y}(P_1,\ldots,P_n) \Leftrightarrow \perp_{D_k}(P_1\sigma,\ldots,P_n\sigma)$$

where $\sigma : Y \to I_Y$ is the substitution that replaces each variable in Y by a fresh input variable in I_Y.

Sketch of Proof. Conditioning over the random variables Y is similar to only considering the distributions of the two programs for any fixed given value of the variables in Y. This corresponds to seeing Y as inputs variables. ■

To reduce independence to equivalence, the idea is that if n programs (as a tuple) are equivalent to a copy of the n programs where they all sample independently their randomness, they are independent. This translates into the following Lemma.

Lemma 7.3. *Let $P_1, \dots, P_n$ be programs over $\mathcal{P}(I, \{r_1, \dots, r_m\})$*

$$\bot_{D_k} (P_1, \dots, P_n) \Leftrightarrow (P_1, \dots, P_n) \approx_{D_k} (P_1\sigma 1, \dots, P_n\sigma_n)$$

where σ_i is the substitution that to any r_j associates a fresh random variable r_j^i.

We now provide a Lemma which states that a program P follows the uniform distribution if and only if it can be used to hide the value of some secret s. This follows the intuition that in the boolean case, any secret xored with the uniform distribution yields the uniform distribution. To provide the general version, we ask that D_k contains a symbol with a property similar to the xor.

Definition 7.7. A function $f : D_k^2 \mapsto D_k$ is right invertible if there exists f^{-1} such that for any x, y, we have $f(f^{-1}(x, y), y) = x$, i.e. for any y, $x \mapsto f(x, y)$ is a bijection.

In any algebra with such a function, uniformity can be reduced to independence.

Lemma 7.4. *Let D_k be a Σ-algebra that contains a binary symbol $+$ such that $+^{D_k}$ is right invertible. Let $P = (p_1, \dots, p_k)$ be a program in $\mathcal{P}(I, R)$ with $\{r_1, \dots, r_k\} \subseteq R$ and $x_1, \dots, x_k$ fresh input variables. We have that*

$$P \approx_{D_k} r_1, \dots, r_k \Leftrightarrow \bot_{D_k} ((p_1 + x_1, \dots, p_k + x_k), (x_1, \dots, x_k))$$

Finally, we show that equivalence reduces to deciding the equivalence to a uniform distribution. However, we only do so for linear programs, i.e., programs in $\overline{\mathcal{P}}(I, R)$.

Lemma 7.5. *Assume that D_k is at least of size two, and contains a right invertible symbol $+$. There exists $T(P_1, P_2, Q_1, Q_2)$ such that for any $P_1, Q_1, P_2, Q_2 \in \overline{\mathcal{P}}(I, R)$ with $r \in R$,*

$$P_1|P_2 \approx_{D_k} Q_1|Q_2 \Leftrightarrow T(P_1, P_2, Q_1, Q_2) \approx_{D_k} r$$

Sketch of Proof. We only show the simpler case of Boolean algebras, i.e., the case where $D_k = \mathbb{F}_2$, and linear programs returning a single value. Let

$$T = (\textbf{if } r = 1 \textbf{ then } P \textbf{ else } 1 - Q)$$

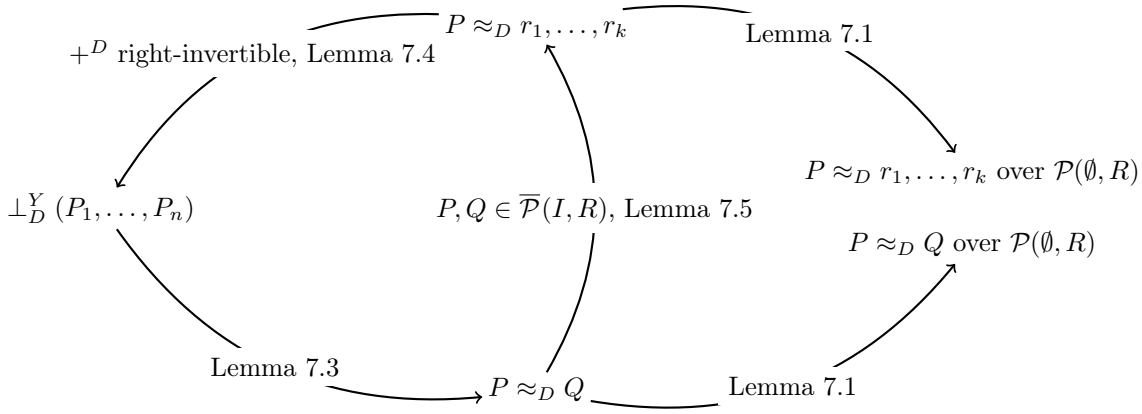
We show that

$$P \approx_{\mathbb{F}_2} Q \text{ iff } T \approx_{\mathbb{F}_2} r$$

where r is a fresh random variable. We fix a valuation $\vec{i} \in \mathbb{F}_2^{|P|}$. By disjunction on the possible values of r, we have that

$$[T]^{\vec{i}}_{\mathbb{F}_2}(0) = \frac{1}{2}[P]^{\vec{i}}_{\mathbb{F}_2}(0) + \frac{1}{2}[Q]^{\vec{i}}_{\mathbb{F}_2}(1) = \frac{1}{2} + \frac{[P]^{\vec{i}}_{\mathbb{F}_2}(0) - [Q]^{\vec{i}}_{\mathbb{F}_2}(0)}{2}.$$

It follows that: $T \approx_{\mathbb{F}_2} r \Leftrightarrow \forall \vec{i}. [T]^{\vec{i}}_{\mathbb{F}_2}(0) = \frac{1}{2} \Leftrightarrow \forall \vec{i}. [P]^{\vec{i}}_{\mathbb{F}_2}(0) = [Q]^{\vec{i}}_{\mathbb{F}_2}(0) \Leftrightarrow P \approx_{D_k} Q$ ■



Lemma 7.3: $\perp_D (P_1, \ldots, P_n) \Leftrightarrow (P_1, \ldots, P_n) \approx_D (P_1\sigma 1, \ldots, P_n\sigma_n)$
Lemma 7.4: if $+^D$ is right-invertible,
$P \approx_D r_1, \ldots, r_k \Leftrightarrow \perp_D ((p_1 + x_1, \ldots, p_k + x_k), (x_1, \ldots, x_k))$
Lemma 7.5: if $+^D$ is right-invertible and D is not a singleton,
$P_1|P_2 \approx_D Q_1|Q_2 \Leftrightarrow T(P_1, P_2, Q_1, Q_2) \approx_D (r)$
Lemma 7.1: $P \approx_D Q \Leftrightarrow (P\sigma, R_I) \approx_D (Q\sigma, R_I)$

Figure 7.2: Summary of Reductions

7.4.2 Semantic Characterization of Equivalence

For any set S, we denote by bij^S the set of bijections over S. We characterize the equivalence of two programs through the existence of a bijection over their random sampling, so that they verify point-wise equality.

Proposition 7.1. *Let $P, Q \in \mathcal{P}(I, R)$.*

$$P \approx_{D_k} Q \Leftrightarrow \forall \vec{i} \in D_k^{|I|}, \forall \vec{o} \in D_k^{|P|}.\ \left|\{\vec{r} \in D_k^{|R|} | [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = \vec{o}\}\right| = \left|\{\vec{r} \in D_k^{|R|} | [\![Q]\!]^{\vec{i},\vec{r}}_{D_k} = \vec{o}\}\right|$$
$$\Leftrightarrow \exists f \in \mathsf{bij}^{D_k^{|R|}}, \forall \vec{i} \in D_k.\ [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = [\![Q]\!]^{\vec{i},f(\vec{r})}_{D_k}$$

Sketch of Proof. The programs are equivalent if their distributions are equal. It means that for each possible output value, the set of random samplings such that the two programs return this value have the same size. If they have the same size, it means that a bijective mapping can be built between those two sets. Doing this for all outputs produces a bijection f over the sampling space, such that if P is equal to c when sampling values $\vec{r}$, Q is equal to c when sampling values $f(\vec{r})$. Remark that all the previous implications are in fact equivalence. ■

Providing a bijection can be a very concise way to prove equivalence, as providing the truth table is not always convenient. Remark that however, proving the existence of a bijection, or the fact that a given function is a bijection, can be difficult.

Example 7.8. We consider sets of input variables $I = \{x\}$ and random variables $R = \{u, v, w\}$. Using Proposition 7.1 we can prove that $x + v \approx_{\mathbb{F}_2} v$ using the bijection $f : v \mapsto x + v$ (we may see f as a function over the terms, rather than the valuations) which satisfies $x + v = f(v)$.

Consider now the equivalence $P_3 \approx_{\mathbb{F}_2} u$ where $P_3 = uv + vw + wu$. We define the function

$$f(r,s,t) := \begin{cases} (1,0,0) \text{ if } (r,s,t) = (0,1,1) \\ (0,1,1) \text{ if } (r,s,t) = (1,0,0) \\ (r,s,t) \text{ otherwise} \end{cases}$$

where a valuation is denoted by a tuple of values. Obviously, f is a bijection and with the following truth table, shows that f verifies $\forall \vec{r} \in \mathbb{F}_2^3, [\![uv + vw + wu]\!]^{\vec{r}}_{\mathbb{F}_2} = [\![u]\!]^{f(\vec{r})}_{\mathbb{F}_2}$

u	0	0	0	0	1	1	1	1
v	0	0	1	1	0	0	1	1
w	0	1	0	1	0	1	0	1
P_3	0	0	0	1	0	1	1	1
f_u	0	0	0	1	0	1	1	1

where f_u denotes the projection of $f(u,v,w)$ on the first component. Intuitively, f_u simply swaps the fourth and the fifth colons of the truth table, in order to produce the distribution of P_3.

Relying on Proposition 7.1 we now introduce a characterization of uniformity based on the notion of R-bijection. Intuitively, a program is R-bijective if for any fixed value of its inputs, the output produced by the program can be seen as a bijection over its random variables. For $P \in \mathcal{P}(I,R)$, it is possible if and only if $|P| = |R|$, i.e., the number of outputs returned by the program is equal to its number of random samplings.

Definition 7.8. Given D, $P \in \mathcal{P}(I,R)$ is R-bijective if and only if

$$\forall \vec{i} \in \mathcal{D}^{|I|}.\ \vec{r} \mapsto [\![P]\!]^{\vec{i},\vec{r}}_D \in \mathsf{bij}^R$$

Example 7.9. We have that $x+u \in \mathcal{P}_{\mathbb{F}_2}(\{x\},\{u\})$ is $\{u\}$-bijective as for any $\vec{i} \in \mathbb{F}_2$, $\vec{r} \mapsto [\![u+x]\!]^{\vec{i},\vec{r}}_{\mathbb{F}_2}$ is a bijection, with itself as inverse.

Corollary 7.1. *If $P \in \mathcal{P}(I,R)$ with $|P| = |R|$ and $R = \{r_1,\dots,r_k\}$, then*

$$P \approx_D r_1,\dots,r_k \quad \Leftrightarrow \quad P \text{ is } R\text{-bijective}$$

Proof. We prove both directions separately.
($\Rightarrow$) Let $r = (r_1,\dots,r_k)$ We must prove that $\forall \vec{i} \in \mathcal{D}^{|X|}.\ \vec{r} \mapsto [\![P]\!]^{\vec{i},\vec{r}}_D$ is bijective. Using Proposition 7.1, we have

$$\forall \vec{i} \in \mathcal{D}^{|I|}.\ \exists f \in \mathsf{bij}^R.\ \forall \vec{r} \in \mathcal{D}^{|R|}.\ [\![P]\!]^{\vec{i},\vec{r}}_D = [\![r]\!]^{\vec{i},f(\vec{r})}_D$$

As f is a bijection, $\vec{r} \mapsto [\![r]\!]^{\vec{i},f(\vec{r})}_D$ is bijective, and as $[\![P]\!]^{\vec{i},\vec{r}}_D = [\![r]\!]^{\vec{i},f(\vec{r})}_D$, so is $\vec{r} \mapsto [\![P]\!]^{\vec{i},\vec{r}}_D$.
($\Leftarrow$) The proof of this direction is similar: for each $\vec{i} \in \mathcal{D}^{|X|}$, $\vec{r} \mapsto [\![P]\!]^{\vec{i},\vec{r}}_D$ is the bijection that allows us to apply Proposition 7.1. ■

Note that for uniformity to imply R-bijectivity the condition that $|R| = |P|$ is necessary.

Example 7.10. As seen before, $x+u \approx_{\mathbb{F}_2} u$ (Example 7.8), and $x+u$ is $\{u\}$-bijective (Example 7.9). Note that $|R| = |P|$ is important here, as for instance $uv + vw + wu$ is not $\{u,v,w\}$-bijective, and $|uv + vw + wu|$, the arity of the program that returns $uv + vw + wu$, is equal to 1, and not to $|\{u,v,w\}|$, but we do have that $uv + vw + wu \approx_{\mathbb{F}_2} u$.

8 Complexity and Decidability

Independence is happiness.

(Susan B. Anthony)

8.1 Introduction

Driven by the security applications, we focus on programs that operate over finite fields, i.e., the case where $D = \mathbb{F}_q$. In this Chapter, we perform a theoretical study of the previous problems, equivalence, majority and independence, both in the finite and the universal case.

In the finite case, the decidability is trivial, but the complexity study allows to derive the exact complexity of the aforementioned problems. It notably provides some insights about whether there exist or not efficient algorithms to solve the given problems. As we place the problems in non trivial complexity classes (above NP and bellow PSPACE), it hints at the fact that there does not exist any efficient algorithm to decide for instance probabilistic non-interference, even in the boolean case.

In the universal case, the family of interpretations considered is $\{\mathbb{F}_{q^k}\}_{k\in\mathbb{N}}$. The integer k can then be seen as the security parameter used for computational indistinguishability. Even before asking whether an efficient algorithm exists for the universal equivalence problem in this setting, one must settle the question of decidability for the universal variants of the problems. In the equivalence case we answer this question positively, and provide insights about the majority problem. This is done through a general reduction from our problems to problems over simple Linear Recurrence Sequences. Thanks to this reduction, we also provide some first insights about approximate equivalence, corresponding to the notion of program indistinguishability. Remark that universal equivalence of two programs implies that they are perfectly indistinguishable by any attacker, without any assumptions on its computational power. They are thus indistinguishable by any polynomial time attacker, i.e., computationally indistinguishable.

Chapter Summary

In the case of probabilistic programs over finite fields, we study the complexity and decidability of equivalence, majority and independence problems both in the finite and universal case. We derive non trivial complexity for the finite case, suggesting that efficient algorithms do not exist. We show decidability of universal equivalence, and devise a general way to draw links between the universal probabilistic problems and widely studied problems on linear recurrence sequences. This yields decidability of independence, and majority over a single point. We also define and provide some insights about program indistinguishability, proving that it is decidable for programs always returning 0 or 1.

8.1.1 Our Contributions

The first contribution of this Chapter is a systematic study of the complexity of the aforementioned problems in the fixed setting. Some background about the complexity classes discussed in the following is given Page 147. We prove that the $\mathbb{F}_{q^k}$-*equivalence* problem is $\mathsf{coNP}^{\mathsf{C=P}}$-complete for any fixed k. We also study the special case of *linear* programs, i.e., without multiplication, conditional nor conditioning, for which the problem can be decided in polynomial time. For the majority problem, we consider two settings: programs with and without inputs. We show that the k-majority problem for inputless programs is PP-complete, whereas the k-majority for arbitrary programs is $\mathsf{coNP}^{\mathsf{PP}}$-complete—thus the second problem is strictly harder than the first, unless $\mathrm{PH} \subset \mathsf{PP}$[1]. The proofs are given by reductions of MAJSAT and $\mathsf{E{-}MAJSAT}$ respectively. These results complement recent work on the complexity of checking differential privacy for arithmetic circuits [GNP19], see Related Work below.

The second, and main contribution, is the study of universal equivalence, $\mathbb{F}_{q^\infty}$-equivalence for short, and universal (0-)majority, $\mathbb{F}_{q^\infty}$-(0-)majority for short. First, we show that the $\mathbb{F}_{q^\infty}$-*equivalence* problem is in 2-EXP and $\mathsf{coNP}^{\mathsf{C=P}}$-hard.

Our proof is based on local zeta Riemann functions, a powerful tool from algebraic geometry, that characterizes the number of zeros of a tuple of polynomials in all extensions of a finite field. Lauder and Wan [LW06] notably propose an algorithm to compute such functions, whose complexity is however exponential. Interestingly, this local zeta function also provides us with a way to reduce our problems to problems over Linear Recurrence Sequences (LRS)[2]: properties of the local zeta function imply that the sequence of number of zeros of a tuple of polynomials over each extension of a finite field is a LRS.

Based on this result, our proof proceeds in two steps. First, we give a reduction for arithmetic programs (no conditionals, nor conditioning) from universal equivalence to checking that some specific local zeta Riemann functions are always null, or equivalently that two LRS are equal. Then, we reduce the general case to programs without conditioning, and programs without conditioning to arithmetic programs. To justify the use of the local zeta Riemman functions, we also provide counterexamples why simpler methods fail or only provide sufficient conditions. Our decidability result significantly generalizes prior work on universal equivalence [BDK+10], which considers the case of linear programs, see Related Work below. In the special case of *arithmetic* programs, i.e., programs without conditionals nor conditioning, equivalence can be decided in EXP-time, rather than 2-EXP.

Second, we give an exponential reduction from the universal 0-majority problem to the positivity problem for Linear Recurrence Sequences (LRS), which given a LRS, asks whether it is always positive. Despite its apparent simplicity, the positivity problem remains open. Decidability has been obtained independently by Mignotte et al [MST84] and by Vereshchagin [Ver85] for LRS of order ≤ 4 and later by Ouaknine and Worrell [OW14a] for LRS with order ≤ 5. Moreover, Ouaknine and Worrell prove in the same paper that deciding positivity for LRS of order 6 would allow to solve long standing open problems in Diophantine approximation. In the general case, the best known lower bound for the positivity problem is NP-hardness [OW12].

Unfortunately, the order of the linear recurrence sequence is related to the degree of the local zeta Riemann function, and thus decidability results for small orders do not apply. This suggests that the problem may not have an efficient solution. We remark that the LRS obtained from the majority problem is *simple*, and we can thus decide a close problem, which is ultimate positivity (is the LRS always positive after some point), that was proven decidable in [OW14b]. Using the

[1] As $\mathrm{PH} \subset \mathsf{coNP}^{\mathsf{PP}}$, $\mathsf{PP} = \mathsf{coNP}^{\mathsf{PP}}$ would imply $\mathrm{PH} \subset \mathsf{PP}$ which is commonly believed to be false.
[2] An LRS is a sequence of integers satisfying a recurrence relation.

Complexity Background - The counting hierarchy

Exact counting The exact counting complexity class, denoted by $\mathsf{C_{=}P}$, is the set of decision problems solvable by a NP Turing Machine whose number of accepting paths is equal to the number of rejecting paths. halfSAT is the natural $\mathsf{C_{=}P}$-complete problem, defined as follows.

halfSAT
INPUT: CNF boolean formula ϕ QUESTION: Is ϕ true for exactly half of its valuations?

$\mathsf{coNP}^{\mathsf{C_{=}P}}$ is the set of decision problems whose complement can be solved by a NP Turing Machine with access to an oracle deciding problems in $\mathsf{C_{=}P}$. The canonical $\mathsf{coNP}^{\mathsf{C_{=}P}}$ problem is (using results from [Tor88, Sec. 4] and [LGM98]):

A−halfSAT
INPUT: CNF boolean formula $\phi(X, Y)$ QUESTION: For all valuations of X, is $\phi(X, Y)$ true for exactly half of the valuations of Y?

Majority counting The complexity class PP is the set of languages accepted by a probabilistic polynomial-time Turing Machine with an error probability of less than $1/2$ for each instance, i.e., a word in the language is accepted with probability at least $1/2$, and a word not in the language is accepted with probability less than $1/2$. Alternatively, one can define PP as the set of languages accepted by a non-deterministic Turing Machine where the acceptance condition is that a majority of paths are accepting. Notably, PP contains both NP and coNP, as well as $\mathsf{C_{=}P}$. Also, PP is closed under finite intersection. A natural PP-complete problem is MAJSAT, the set of boolean formulae true for at least half of their valuations:

MAJSAT
INPUT: CNF boolean formula ϕ QUESTION: Is ϕ true for at least half of its valuations?

$\mathsf{coNP}^{\mathsf{PP}}$ is the class of problems whose complement is decided by a non deterministic polynomial time Turing Machine with access to an oracle deciding problems in PP. The classical $\mathsf{NP}^{\mathsf{PP}}$ problem is E−MAJSAT [LGM98] :

E−MAJSAT
INPUT: CNF boolean formula $\phi(X, Y)$ QUESTION: Is there a valuation of X such that, $\phi(X, Y)$ is true for at least half of the valuations of Y?

Its complement, A−MINSAT is then the classical $\mathsf{coNP}^{\mathsf{PP}}$ problem.

	x-(*conditional*)-{*equivalence*, *independence*, *uniformity*}		
	linear	arithmetic	general
$x = \mathbb{F}_{q^k}$	PTIME	coNP$^{\mathsf{C=P}}$-complete	coNP$^{\mathsf{C=P}}$-complete
$x = \mathbb{F}_{q^\infty}$	PTIME	EXP coNP$^{\mathsf{C=P}}$-hard	2-EXP coNP$^{\mathsf{C=P}}$-hard

Figure 8.1: Summary of Results Related to Equivalence

	$\mathbb{F}_{q^k}$-0-*majority*	$\mathbb{F}_{q^k}$-*majority*	$\mathbb{F}_{q^\infty}$-0-*majority*	$\mathbb{F}_{q^\infty}$-*majority*
without inputs	PP-complete	coNP$^{\mathsf{PP}}$-complete	PP-hard	
			$\leq_{\mathsf{EXP}}$ POSITIVITY	
with inputs	coNP$^{\mathsf{PP}}$-complete		?	
			coNP$^{\mathsf{PP}}$-hard	

Figure 8.2: Summary of Results Related to Majority

results from [Kie76], we observe that the reduction extends to a more general form of universal majority problem.

We obtain lower complexity bounds by reducing the finite case to the universal case. It remains an interesting open question whether the universal case is strictly harder than the finite case.

As side contributions, we provide some first tentative attempts for verifying program indistinguishability, that can be seen as an approximate universal equivalence. We define and prove the decidability of the LRS negligibility problem, but leave the question of program indistinguishability open. We however obtain the decidability of program indistinguishability for programs that only return 0 or 1. Finally, we also prove that enriching the programming language with loops makes the universal equivalence problem undecidable over finite fields.

Figures 8.1 and 8.2 summarize our results for the equivalence and majority problems.

Limitations

The algorithm derived for the decidability of the universal equivalence has an exponential running time, and cannot be considered efficient. Furthermore, the complexity results in the universal case are not tight. Finally, the reduction to the positivity problem for majority is also exponential, and we thus cannot transpose the coNP$^{\mathsf{PP}}$ lower bound for universal majority to the positivity problem.

8.1.2 Related Work

Universal equivalence Compared to [BDK+10], that propose a decision procedure for universal equivalence in the linear case, we give an alternative decision procedure and analyze its complexity.

Majority problems To the best of our knowledge, the universal majority question is novel. For the fixed case, the closest related work develops methods for proving differential privacy or for quantifying information flow.

Gaboardi, Nissim and Purser [GNP19] study the complexity of verifying pure and approximate (ϵ, δ)-differential privacy for arithmetic programs, as well as approximations of the parameters ϵ and δ. The parameter δ quantifies the approximation and $\delta = 0$ corresponds to the pure case. Our majority problem can be seen as a subcase of differential privacy, where r corresponds to ϵ, and $\delta = 0$. In particular, the complexity class they obtain for pure differential privacy coincides with the complexity of our 0-majority problem, even when restricted to the case $r = 1$. This means that the ϵ parameter does not essentially contribute to the complexity of the verification problem. Also, while they consider arithmetic programs, we consider the more general case of programs with conditioning.

Chistikov, Murawski and Purser [CMP19] also study the complexity of approximating differential privacy, but in the case of Markov Chains.

Theory of fields A celebrated result by Ax [Ax68] shows that the theory of finite fields is decidable. In a recent development based on Ax's result, Johnson [Joh16] proves decidability of the theory of rings extended with quantifiers $\mu_k^n x.\ P$, stating that the number of x such that P holds is equal to k modulo n. Although closely related, these results do not immediately apply to the problem of equivalence.

? Future Work

We leave several questions of interest open:

- the exact complexity of universal equivalence is open. It is even unknown whether the universal problem strictly harder than the non-universal one;
- the decidability of universal majority is open. The decidability of POSITIVITY would yield decidability of universal 0-majority and equivalently, undecidability of universal majority would also solve negatively the POSITIVITY problem;
- the decidability of program indistinguishability is open, for programs that do not return a boolean. It asks if the statistical distance between the distributions of two programs is negligible in k. This would have direct applications in provable security.

8.2 Complexity in the Finite Case

Section Summary

We start by studying the complexity of several problems over a given finite field. In this case, all problems are decidable by explicitly computing the distributions of the programs. We however provide precise complexity results and show that these problems have complexities in the counting hierarchy [Tor91].

8.2.1 Conditional Equivalence

To reason about equivalence, we focus on $\mathbb{F}_{q^k}$-*conditional equivalence*, and we proceed in four steps, showing that:

1. without loss of generality, we can consider programs without inputs; (Lemma 7.1)
2. verifying if the conditioned distributions of two inputless programs coincide on some given value is in $\mathsf{C_{=}P}$; (Lemma 8.1)
3. verifying if the conditioned distribution of inputless programs coincide on all values is in $\mathsf{coNP}^{\mathsf{C_{=}P}}$; (Corollary 8.1)
4. and finally, even equivalence for programs over $\mathbb{F}_2$ is $\mathsf{coNP}^{\mathsf{C_{=}P}}$-hard. (Lemma 8.2)

This allow us to conclude that $\mathbb{F}_{q^k}$-*equivalence* and $\mathbb{F}_{q^k}$-*conditional equivalence* are both $\mathsf{coNP}^{\mathsf{C_{=}P}}$-complete.

In the finite case, equivalence is very close to comparing and counting the number of solutions of polynomial systems. We remark that $\mathbb{Z}$-equivalence is undecidable: this is a consequence of Hilbert's 10th problem, as a polynomial over randomly sampled variables will be equivalent to zero if and only if it does not have any solutions.

Complexity results for conditional equivalence Conditional equivalence is a direct generalization of equivalence. We thus trivially have, for any $k \in \mathbb{N} \cup \{\infty\}$, that $\mathbb{F}_{q^k}$-*equivalence* reduces in polynomial time to $\mathbb{F}_{q^k}$-*conditional equivalence*.

As we can without loss of generality ignore the inputs (see Lemma 7.1), we study the complexity of deciding equality of distributions of two inputless programs on a specific value. To this end, we build a polynomial time Turing Machine that accepts half of the time if and only if the programs given as input have the same probability to be equal to some given value. Essentially, it is based on the fact that over $\mathbb{F}_2$,

$$\textbf{if } r = 0 \textbf{ then } P \textbf{ else } (Q+1) \approx_{\mathbb{F}_2} r \;\Leftrightarrow\; P \approx_{\mathbb{F}_2} Q$$

.

Lemma 8.1. *Let $P_1, Q_1 \in \mathcal{P}_{\mathbb{F}_q}(\emptyset, R)$ and $P_2, Q_2 \in \overline{\mathcal{P}}_{\mathbb{F}_q}(\emptyset, R)$ with $|P_1| = |Q_1| = n$. For any $\vec{o} \in \mathbb{F}_{q^k}^n$, we can decide in $\mathsf{C_{=}P}$ if:*

$$[(P_1, P_2)]_{\mathbb{F}_{q^k}}(\vec{o}, \boldsymbol{0}) = [Q_1, Q_2]_{\mathbb{F}_{q^k}}(\vec{o}, \boldsymbol{0})$$

Proof. As a shortcut, for $P \in \mathcal{P}_q(\emptyset, R)$ (a program without inputs) and $\vec{o} \in \mathbb{F}_{q^k}^{|P|} \times \{\bot\}$, we denote by $\tilde{P}^{\vec{o}}$, the probability that P evaluates to $\vec{o}$. Let $P_1, Q_1 \in \mathcal{P}_q(\emptyset, R)$, $P_2, Q_2 \in \overline{\mathcal{P}}_q(\emptyset, R)$ with $|P_1| = |Q_1| = n$. For any $c \in \mathbb{F}_q^n$, let us consider the probabilistic polynomial time Turing Machine M which, on input $P_1, P_2, Q_1, Q_2, \vec{o}$, is defined by:

$x \xleftarrow{\$} \{0,1\}; \vec{r} \xleftarrow{\$} \mathbb{F}_q^{|R|}; \vec{r'} \xleftarrow{\$} \mathbb{F}_q^{|R|};$
if $x = 0$ then
  if $\neg(P_1(\vec{r}) = \vec{o} \wedge P_2(\vec{r}) = \mathbf{0} \wedge Q_1(\vec{r'}) \neq \bot)$ then
    ACCEPT
  else REJECT
else
  if$(Q_1(\vec{r}) = \vec{o} \wedge Q_2(\vec{r}) = \mathbf{0} \wedge P_1(\vec{r'}) \neq \bot)$ then
    ACCEPT
  else REJECT

Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$. The probability that M accepts is, by case disjunction on the value of x:

$$\tfrac{1}{2}(1 - \tilde{P}^{(\vec{o},\mathbf{0})}(1 - \tilde{Q_1}^{\bot})) + \tfrac{1}{2}(\tilde{Q}^{(\vec{o},\mathbf{0})}(1 - \tilde{P_1}^{\bot})) = \tfrac{1}{2} + \frac{\tilde{Q}^{(\vec{o},\mathbf{0})}(1-\tilde{P_1}^{\bot}) - \tilde{P}^{(\vec{o},\mathbf{0})}(1-\tilde{Q_1}^{\bot})}{2}$$

And thus:

$$
\begin{aligned}
[P]_{\mathbb{F}_{q^k}}(\vec{o},\mathbf{0}) = [Q]_{\mathbb{F}_{q^k}}(\vec{o},\mathbf{0}) &\Leftrightarrow \frac{\mathbb{P}_{\vec{r}\xleftarrow{\$}\mathbb{F}_q^{|R|}}\{[\![P]\!]^{\vec{i},\vec{r}}_{\mathbb{F}_{q^k}}=(\vec{o},\mathbf{0})\}}{\mathbb{P}_{\vec{r}\xleftarrow{\$}\mathbb{F}_q^{|R|}}\{[\![P]\!]^{\vec{i},\vec{r}}_{\mathbb{F}_{q^k}}\neq\perp\}} = \frac{\mathbb{P}_{\vec{r}\xleftarrow{\$}\mathbb{F}_q^{R}}\{[\![Q]\!]^{\vec{i},\vec{r}}_{\mathbb{F}_{q^k}}=(c,\mathbf{0})\}}{\mathbb{P}_{\vec{r}\xleftarrow{\$}\mathbb{F}_q^{R}}\{[\![Q]\!]^{\vec{i},\vec{r}}_{\mathbb{F}_{q^k}}\neq\perp\}} \\
&\Leftrightarrow \frac{\tilde{P}^{(\vec{o},\mathbf{0})}}{1-\tilde{P_1}^{\perp}} = \frac{\tilde{Q}^{(\vec{o},\mathbf{0})}}{1-\tilde{Q_1}^{\perp}} \\
&\Leftrightarrow \tilde{Q}^{(\vec{o},\mathbf{0})}(1-\tilde{P_1}^{\perp}) - \tilde{P}^{(\vec{o},\mathbf{0})}(1-\tilde{Q_1}^{\perp}) = 0 \\
&\Leftrightarrow M \text{ accepts exactly half of the time}
\end{aligned}
$$

As we consider that the size of a polynomial is the size of the corresponding formula, polynomials can be evaluated in polynomial time. Thus, the previous Turing Machine does run in polynomial time.

■

As $\mathsf{C}_{=}\mathsf{P}$ is closed under finite intersection [Tor88], we can decide in $\mathsf{C}_{=}\mathsf{P}$ if two distributions over a set of fixed size are equal, by testing the equality over all possible values. When we only consider inputless programs of fixed arity, the set of values to test is constant, and the equivalence problem is in $\mathsf{C}_{=}\mathsf{P}$ (see Corollary C.1 for details). However, when we extend to inputs, or to programs of variable arity, we need to be able to check for all possible value if the distribution are equal over this element. (Note that our encoding that allows to only consider inputless programs increases the arity.) Checking all possible values is typically in coNP. We thus obtain that:

Corollary 8.1. *For any $k \in \mathbb{N}$, $\mathbb{F}_{q^k}$-*equivalence *and $\mathbb{F}_{q^k}$-*conditional equivalence *are in $\mathsf{coNP}^{\mathsf{C}_{=}\mathsf{P}}$.*

To conclude completeness for both $\mathbb{F}_{q^k}$-*equivalence* and $\mathbb{F}_{q^k}$-*conditional equivalence*, it is sufficient to show the hardness of $\mathbb{F}_2$-*equivalence*, which we do by reducing $\mathsf{A}{-}\mathsf{halfSAT}$. We simply transform a CNF boolean formula into a polynomial over $\mathbb{F}_2$. This is a purely technical operation (see Lemma C.1).

Lemma 8.2. $\mathbb{F}_2$-equivalence *is* $\mathsf{coNP}^{\mathsf{C}_{=}\mathsf{P}}$*-hard.*

Proof. Given a CNF formula $\phi(I,R)$ over two sets of variables and $(\vee,\wedge)$ we set $P = \phi' \in \mathcal{P}_{\mathbb{F}_2}(I,R)$ obtained according to Lemma C.1. Given a fresh random variable r:

$$
\begin{aligned}
P(I,R) \approx_{\mathbb{F}_2} r &\Leftrightarrow \text{for all valuations of } I,\ \phi \text{ is true for half of the valuations of } R \\
&\Leftrightarrow \phi(I,R) \in \mathsf{A}{-}\mathsf{halfSAT}
\end{aligned}
$$

■

8.2.2 Independence

The results for $\mathbb{F}_{q^k}$-*conditional equivalence* naturally translate to the independence problem: are the distributions of multiple programs independent? We show here that equivalence and (conditional) independence have the same complexity. Conditional independence (Definition 7.4) asks if for any fixed value of some variables Y, the programs are independent, i.e., if the product of their distributions is equal to the distribution of their product.

Using Lemmas 7.2 and 7.3, we obtain the same complexity as equivalence.

Corollary 8.2. $\mathbb{F}_{q^k}$-conditional independence *is in* $\mathsf{coNP}^{\mathsf{C_=P}}$.

We now show the hardness of conditional independence. The key idea comes from Lemma 7.4: for any program P and fresh random r, we have that $\perp^{\emptyset}_{\mathbb{F}_2}(P + r, r)$ if and only if P follows the uniform distribution. Intuitively, P perfectly masks the dependence in r only if it is a uniform value. Thus, we reduced uniformity to independence, and as we previously reduced $\mathsf{A{-}halfSAT}$ to uniformity, we conclude.

Theorem 8.1. $\mathbb{F}_{q^k}$-conditional independence *is* $\mathsf{coNP}^{\mathsf{C_=P}}$*-complete.*

8.2.3 Majority

The goal of this Section is to show that the majority problem is $\mathsf{coNP}^{\mathsf{PP}}$-complete. To this end, we study the complexity of $\mathbb{F}_{q^k}$-0-*majority*, showing:

- PP-completeness for inputless programs;
- $\mathsf{coNP}^{\mathsf{PP}}$-completeness in general.

The proof in both cases uses similar ideas as for equivalence. Note that we actually use the same Turing Machine for the Membership. As both complexity classes are closed under finite intersection, it yields the complexity of $\mathbb{F}_{q^k}$-*majority*, which can be decided using $\mathbb{F}_{q^k}$ times $\mathbb{F}_{q^k}$-0-*majority*.

Complexity results for the majority problem To obtain the complexity of $\mathbb{F}_{q^k}$-0-*majority* over inputless programs, we notice that the Turing Machine we used to obtain the complexity of the equivalence problem are easily adapted for our purpose. Indeed, it accepted half of the time if the two distributions were equal on a single value, but it actually accepts with probability greater than half only if the value of the first distribution is greater than the second one on the given point.

The only difficulty is that we are comparing with a rational. We thus briefly show how one can assume without loss of generality that $r = 1$ (in which case we omit r from the notation). The idea is, given $r, s \in \mathbb{N}$, that $P \prec^{\frac{r}{s}}_{\mathbb{F}_{q^k}} Q \Leftrightarrow (P, T_r) \prec_{\mathbb{F}_{q^k}} (Q, T_s)$, if T_j is a program that is equal to zero with probability $\frac{1}{j}$.

Technical Details

Depending on the value of j, the program T_j is more or less concise. For instance, for $j = q^l$, the program

$$x_1, \ldots, x_l \xleftarrow{\$} \mathbb{D}^l; \textbf{if } x_1 = 0 \ \wedge \ldots \wedge \ x_l = 0 \textbf{ then return } \ 0 \textbf{ else return } \ 1$$

is equal to zero with probability $\frac{1}{q^l}$. However, more complex integers may require precise encoding using an exponential number of conditionals, and we must thus consider that r, s is given in input as two integers written in unary. Remark that in practice, it is natural to use particular rationals such as $\frac{1}{q^l}$, for which there is no exponential blow up.

Lemma 8.3. *For any* $k \in \mathbb{N}$, $\mathbb{F}_{q^k}$-0-majority *reduces in polynomial time to* $\mathbb{F}_{q^k}$-0-majority *with* $r = 1$.

The proof showing that $\mathbb{F}_{q^k}$-0-*majority* is in PP is similar to proving that testing if two distributions are equal over a point is in $\mathsf{C_=P}$.

Lemma 8.4. *For any $k \in \mathbb{N}$, $\mathbb{F}_{q^k}$-0-majority restricted to inputless programs is in* PP.

We prove PP-completeness by deriving the hardness from MAJSAT, with a proof similar to the one of Lemma 8.2.

Lemma 8.5. $\mathbb{F}_2$-0-majority *is* PP*-hard (even for inputless programs).*

Finally, as PP is closed under finite intersection, we also get that $\mathbb{F}_{q^k}$-*majority* is PP-complete.

Let us now turn to the general version, for programs with inputs. By using some fresh inputs variables, let us remark that one can easily reduce $\mathbb{F}_{q^k}$-*majority* to $\mathbb{F}_{q^k}$-0-*majority*. Indeed, for $P, Q \in \mathcal{P}_{\mathbb{F}_q}(I, R)$ and $c \in \mathbb{F}_q^{|P|}$, with a fresh $x \in I$:

$$\forall \vec{i} \in \mathbb{F}_{q^k}^{|I|}.\ [P]^{\vec{i}}_{\mathbb{F}_{q^k}}(c) \leq r\, [Q]^{\vec{i}}_{\mathbb{F}_{q^k}}(c) \Leftrightarrow (P - x) \prec^r_{\mathbb{F}_{q^k}} (Q - x)$$

We show that $\mathbb{F}_{q^k}$-*majority* is $\mathsf{coNP}^{\mathsf{PP}}$ complete, and thus is most likely[3] harder than its version without inputs. The membership and hardness proofs are similar to the equivalence problem when going from $\mathsf{C}_{=}\mathsf{P}$ to $\mathsf{coNP}^{\mathsf{C}_{=}\mathsf{P}}$.

Lemma 8.6. $\mathbb{F}_{q^k}$-majority *is* $\mathsf{coNP}^{\mathsf{PP}}$ *complete.*

8.3 The Universal Case

Section Summary

We first give some general insights on universal equivalence showing important differences with the case of a fixed field. We then derive a general way to study the universal properties by reducing them to Linear Recurrence Sequences problems. This allows us to provide our main decidability result for universal equivalence, first for arithmetic programs, then arithmetic programs enriched with conditionals, and finally for general programs. We continue by studying two other problems in the universal case, that follow easily from the reduction to LRS: independence and 0-majority. For independence and equivalence, the universal problem is in 2-EXP.

8.3.1 General Remarks

In this Section we try to provide some insights on the difficulty of deciding $\mathbb{F}_q^\infty$-*equivalence*. First of all, we note that equivalence and universal equivalence do *not* coincide.

Example 8.1. The program $x^2 + x$ (with x a random variable) and the program 0 are equivalent over $\mathbb{F}_2$ (they are then both equal to zero), but not over $\mathbb{F}_4$.

In the case of a given finite field, equivalence can be characterized by the existence of a bijection, see for instance [BGJ+19]. We denote by $\mathsf{bij}^{\mathbb{F}_q^m}$ the set of bijections over $\mathbb{F}_q^m$. Any element $\sigma \in \mathsf{bij}^{\mathbb{F}_q^m}$ can be expressed as a tuple of polynomials (see e.g., [Nip90]), and can be applied as a substitution.

[3] As PH $\subset \mathsf{coNP}^{\mathsf{PP}}$, $\mathsf{PP} = \mathsf{coNP}^{\mathsf{PP}}$ would imply PH $\subset$ PP which is commonly believed to be false.

The characterization can then be stated as follows, where we denote by $=_{\mathbb{F}_q}$ equality between polynomials modulo the rule of the field (i.e., $X^q = X$).

$$P \approx_{\mathbb{F}_q} Q \Leftrightarrow \exists \sigma \in \mathsf{bij}^{\mathbb{F}_q^m}, P =_{\mathbb{F}_q} Q\sigma$$

However, there are universally equivalent programs such that there does *not* exist a uniform σ suitable for all extensions.

Example 8.2. Consider $P = xy + yx + zx$ where all variables are randomly sampled. With $\sigma : (x, y, z) \mapsto (x, y + x, z + x)$, we get that $P \approx_{\mathbb{F}_{2^\infty}} x^2 + yz$. Now, $x \mapsto x^2$ is a bijection over all $\mathbb{F}_{2^k}$, so we also have $P \approx_{\mathbb{F}_{2^\infty}} x + yz$ and finally $P \approx_{\mathbb{F}_{2^\infty}} x$.

But here, a bijection between $x^2 + yz$ and x must use the inverse of x^2 whose expression depends on the size of the field. Thus, there isn't a universal polynomial σ which is a bijection such that on all $\mathbb{F}_{2^k}$, $P =_{\mathbb{F}_{2^k}} Q \circ \sigma$.

Nevertheless, we can note that for linear programs this characterization allows us to show that $\mathbb{F}_q$-*equivalence* and $\mathbb{F}_{q^\infty}$-*equivalence* are equivalent. Intuitively, the bijection allowing to obtain the equality between two linear programs is also a bijection valid for all extensions of the finite field, as the bijection is linear, and is thus a witness of equivalence over all extensions. For linear programs, there exists a polynomial time decision procedure for equivalence, and hence for universal equivalence.

Lemma 8.7. $\mathbb{F}_{q^\infty}$-equivalence *restricted to linear programs is in* PTIME.

Moreover, building on results from [Mau01] on Tame automorphisms, we can use the above characterization to design a sufficient condition which implies universal equivalence for general programs. Even though not complete this sufficient condition may be useful to verify universal equivalence more efficiently in practice.

A Sufficient Condition

In the univariate case, our notion is also strongly linked to exceptional polynomials, permutation polynomials over $\mathbb{F}_q[x]$ that are permutations over infinitely many $\mathbb{F}_{q^k}[x]$.

A univariate polynomial that is uniform is then an exceptional polynomial of $\mathbb{F}_q[x]$. They have been fully characterized [MP13, p237]. The multivariate case appears unsolved, but an efficient algorithm for this case would provide new insights about our problems.

With the characterization through bijections of Proposition 7.1, we can however easily obtain the following condition, for any function σ:

$$\sigma \in \bigcap_k \mathsf{bij}^{\mathbb{F}_{q^k}^m} \Rightarrow P \approx_{\mathbb{F}_{q^\infty}} P\sigma$$

Notably, any linear bijection in $\mathsf{bij}^{\mathbb{F}_q^m}$ is also in $\bigcap_k \mathsf{bij}^{\mathbb{F}_{q^k}^m}$. Leveraging some mathematical results classifying the bijections over $\mathbb{F}_{q^k}^m$, we can also provide some insights about functions that are bijections over all extensions of a finite field.

Technical Details

We derive two Lemmas that provide an easy way to generate bijections that are bijections over all extensions of a finite field, and can thus serve as a witness for a universal equivalence.

We first use Theorem 3.2 of [Mau01] to classify what are the bijections over $\mathbb{F}_{q^k}^m$. For a finite field $\mathbb{F}$, $\mathsf{bij}^{\mathbb{F}^n}$ denotes the set of bijections over $\mathbb{F}^n$; and $\mathcal{E}(T(\mathbb{F},n))$ denotes the set of bijections obtained through

- permutations: $(x_1,\ldots,x_n)\mapsto(x_{\pi(1)},\ldots,x_{\pi(n)})$),
- scalar multiplications: for any $a\in\mathbb{F}^*$, $(x_1,\ldots,x_n)\mapsto(ax_1,\ldots,x_n)$),
- and linear transformations: for any $P\in\mathbb{F}[x_2,\ldots,x_n]$,

$$(x_1,\ldots,x_n)\mapsto(x_1+P(x_2,\ldots,x_n),\ldots,x_n)$$

$\mathcal{E}(T(\mathbb{F},n))$ is called the set of the tame automorphisms.

Theorem 8.2 (2.3 of [Mau01])**.** *We have:*

- *if $n=1$, and $\mathbb{F}=\mathbb{F}_2$ or $\mathbb{F}_3$, then $\mathcal{E}(T(\mathbb{F},n))=\mathsf{bij}^{\mathbb{F}^n}$,*
- *if $n\geq 2$ and $\mathbb{F}\neq\mathbb{F}_{2^m}$ for $m>1$, $\mathcal{E}(T(\mathbb{F},n))=\mathsf{bij}^{\mathbb{F}^n}$,*
- *else, $\mathcal{E}(T(\mathbb{F},n))\neq\mathsf{bij}^{\mathbb{F}^n}$.*

This allows us to obtain that:

Lemma 8.8. *For any prime $p>2$, integers $k\geq 1$ and $n>1$, for any function f:*

$$f\in\mathsf{bij}^{\mathbb{F}_{p^k}^n}\Rightarrow\forall k'>k.f\in\mathsf{bij}^{\mathbb{F}_{p^{k'}}^n}$$

Proof. Let $f\in\mathsf{bij}^{\mathbb{F}_{p^k}}$. With Theorem 8.2, we have that for all prime p not equal to 2:

$$\mathcal{E}(T(\mathbb{F}_{p^k},n))=\mathsf{bij}^{\mathbb{F}_{p^k}^n}$$

Thus, f can be written as a composition of substitutions, scalar multiplications and linear transformations. All those operations are directly bijections over any $\mathbb{F}_{p^k}^n$, we thus conclude:

$$\forall k'>k.f\in\mathsf{bij}^{\mathbb{F}_{p^{k'}}^n}$$

■

The case $p=2$ must be handled differently:

Lemma 8.9. *For any $k>1$ and $n>1$, for any function f:*

$$f\in\mathsf{bij}^{\mathbb{F}_{2^{2(2k+1)}}^n}\Rightarrow\forall k'>2(2k+1).f\in\mathsf{bij}^{\mathbb{F}_{2^{k'}}^n}$$

Proof. For any m, we denote by $\mathcal{F}(T(\mathbb{F}_{2^m},n))$ the set generated by $\mathcal{E}(T(\mathbb{F}_{2^m},n))$ and the permutation $\sigma=(X_1,\ldots,X_n)\mapsto(X_1^2,\ldots,X_n)$. It is shown in [LN83, p. 351] that x^n is a bijection in $\mathbb{F}_q$ if n and $q-1$ are coprime. We have that for any k, 2 and 2^k-1 are coprime, and then, we have $\mathcal{F}(T(\mathbb{F}_{2^{2(2k+1)}},n))=\mathsf{bij}^{\mathbb{F}_{2^{2(2k+1)}}}$

Let us fix k and let $f\in\mathsf{bij}^{\mathbb{F}_{2^{2(2k+1)}}}$.

Thus, f can be written as a composition of substitutions, scalar multiplications, linear transformations and σ. Recall that σ is a bijection over all $\mathbb{F}_{2^k}^n$, and the others trivially are. We thus conclude:

$$\forall k' > 2(2k+1). f \in \mathsf{bij}^{\mathbb{F}_{2^{k'}}^n}$$

■

8.3.2 From Arithmetic Programs without Inputs to LRS

We first consider the case of arithmetic programs without inputs, $P, Q \in \overline{\mathcal{P}}_{\mathbb{F}_q}(\emptyset, R)$. In this subcase, P and Q are simply tuples of polynomials over a finite field. Thanks to the properties of the local zeta Riemann functions, we are able to link the distributions of P and Q to some simple Linear Recurrence Sequences (LRS). We may then leverage results on LRS to reason about our problems.

Local zeta Riemann functions We recall the definition and relevant properties of local zeta Riemann functions. For a tuple P of polynomials $P_1, \ldots, P_m \in \mathbb{F}_q[X_1, \ldots, X_n]$, the local zeta Riemann function over T is the formal series

$$Z(P, T) = \mathsf{exp}\left(\sum_{k \in \mathbb{N}^*} \frac{|N_k(P)|}{k} T^k\right)$$

where $N_k(P) = \{\vec{x} \in \mathbb{F}_{q^k}^n \mid \bigwedge_{1 \leq i \leq m} P_i(\vec{x}) = 0\}$.

Remark that given $P \in \overline{\mathcal{P}}_{\mathbb{F}_q}(\emptyset, R)$,

$$[P]_{\mathbb{F}_{q^k}}(0) = \frac{|N_k(P)|}{q^{k \times |R|}}$$

Weil's conjecture [Wei49] states several fundamental properties of local zeta Riemann functions over algebraic varieties. Dwork [Dwo60] proves part of Weil's conjecture stating that the local zeta Riemann functions over algebraic varieties is a rational function with integer coefficients—recall that $Z(T)$ is a rational function iff there exist polynomials $R(T)$ and $S(T)$ such that $Z(T) = R(T)/S(T)$. Bombieri [Bom66] shows that the sum of the degrees of R and S is upper bounded by $4(d+9)^{n+1}$, where d is the total degree of $(P_1, \ldots, P_m)$. It follows that the values of N_k for $k \leq 4(d+9)^{n+1}$ suffice for computing Z; since these values can be computed by brute force, this yields an algorithm for computing Z.

We will by abuse of notations write $Z(P)$ instead of $Z(P, T)$ for the local zeta function of P. $Z(P)$ completely characterizes the number of times P is equal to zero on all the different extensions. For instance, $Z(P) = Z(Q)$ allows us to conclude that P and Q always evaluate to zero for the same number of valuations, and this over any $\mathbb{F}_{q^k}$. A classical algorithm to compute Z is provided in [LW06].

Technical Details

Weil's conjecture actually only applies to non-singular projective varieties. However, as outlined by [LW06], Dwork's proof can be used to obtain the stronger result of the rationality of the local zeta function for any algebraic variety.

Linear recurrence sequences We recall that the Linear Recurrence Sequence (LRS) denoted by $\langle u_k \rangle$ is an infinite sequence of reals $u_1, u_2, \ldots$ such that there exist real constants $a_1, \ldots, a_n$ such that for all $k \geq 0$,

$$u_{k+n} = a_1 u_{k+n-1} + \cdots + a_n u_k$$

The order of a LRS $\langle u_k \rangle$ is the smallest positive n such that the equation above holds. The recurrence relation can be associated to a polynomial, called the characteristic polynomial. We then say that a LRS is simple if its characteristic polynomial does not have any repeated roots. As outlined in [OW14b], a LRS of order n is notably simple if there exist algebraic constants $\gamma_1, \ldots, \gamma_n$ and non-zero real algebraic constants $c_1, \ldots ..., c_n$ such that, for all $k \geq 0$:

$$u_k = \sum_{1 \leq i \leq n} c_i \gamma_i^k$$

Remark that given two simple LRS of order n, it is enough to test the equality of the first n terms to obtain equality of the two LRS. Some other problems related to our study are:

- the positivity problem: for all $k \in \mathbb{N}$, does it holds that $u_k \geq 0$? It is only known to be decidable for LRS of order 5, and of order 9 in the case of simple LRS.
- the ultimate positivity problem: does there exists K such that for all $k > K$, $u_k \geq 0$? It is decidable for simple LRS but its decidability in the general case is open.

From programs to LRS Summing up the results from Dwork, Bombieri and Deligne, [CL06] allows us to characterize $\langle N_k(P) \rangle$ as a simple LRS. Given a tuple P of m polynomials in n variables with maximal degree a, there exist integers a_1, a_2 such that $a_1 + a_2 \leq (4a+9)^{n+m}$ and algebraic numbers $\alpha_1, \ldots, \alpha_{a_1}, \beta_1, \ldots, \beta_{a_2}$ such that for any $k \geq 1$:

$$N_k(P) = \sum_{j=1}^{a_2} \beta_j^k - \sum_{j=1}^{a_1} \alpha_j^k$$

Furthermore, we know that there exist integers s_j^α, s_j^β between 0 and $2K_P$ (K_P is a constant that depends on the dimension of the variety of P) such that $|\alpha_j| = q^{s_j^\alpha/2}$ and $|\beta_j| = q^{s_j^\beta/2}$.

We thus have that $\langle N_k(P) \rangle$ is a simple LRS. Remark that given P, computing the LRS corresponding to $N_k(P)$ or computing $Z(P)$ is equivalent (recall that $Z(P)$ is the formal power series corresponding to the LRS $\langle N_k(P) \rangle$, and the reductions given in this Chapter are thus exponential. Based on the previous discussions, we obtain the following Corollary:

Corollary 8.3. *Let $P_1, \ldots, P_k \in \overline{\mathcal{P}}_{\mathbb{F}_q}(\emptyset, R)$, any linear combination of the $\{N_k(P_i)\}_{1 \leq i \leq k}$ is a LRS. So is any linear combination of the $\{[P_i]_{\mathbb{F}_{q^k}}(0)\}_{1 \leq i \leq k}$.*

LRS, which have been widely studied, provide a uniform way to reason about our relational properties:

- try to encode the relational property as a property of some linear combinations of $\{[P_i]_{\mathbb{F}_{q^k}}(0)\}_{1 \leq i \leq k}$;
- reasoning about the corresponding properties of the simple LRS.

This directly implies that, given $P, Q \in \overline{\mathcal{P}}_{\mathbb{F}_q}(\emptyset, R)$, one can decide if:

- $\exists K,\ \forall k > K.\ [P]_{\mathbb{F}_{q^k}} \geq 0$. This is because ultimate positivity is decidable for simple LRS [OW14b]. This implies decidability of a variant of the q^k-0-majority, that we may call ultimate q^k-0-majority: $\exists K,\ \forall k > K.\ [P]_{\mathbb{F}_{q^k}} \geq [Q]_{\mathbb{F}_{q^k}}$.

- $\forall k \geq 0.\ [P]_{\mathbb{F}_{q^k}}(0) = [Q]_{\mathbb{F}_{q^k}}(0)$. This is because we can decide if two LRS are equal. Remark that this is only a reformulation of testing if $Z(P) = Z(Q)$. Hence, testing if two programs have the same probability to return 0 over all finite fields is decidable.

Furthermore, for arithmetic programs without inputs, we have reduced $\mathbb{F}_{q^\infty}$-0-*majority* to the positivity problem for LRS, as the question is if for all k, $[P]_{\mathbb{F}_{q^k}} - [Q]_{\mathbb{F}_{q^k}} \geq 0$ where the left hand side is a LRS.

8.3.3 Decidability of Universal Equivalence

We show decidability of $\mathbb{F}_{q^\infty}$-*equivalence*, leveraging tools from algebraic geometry, showing that[4]:

1. $\mathbb{F}_{q^\infty}$-*conditional equivalence* is decidable for arithmetic programs; (Lemma 8.10)
2. it is also decidable for programs with conditionals; (Lemma 8.11)
3. it is finally decidable for programs with conditioning, e.g., failures. (Lemma 8.12)

Notice that, given two programs P and Q, the local zeta function directly allows us to conclude if they are equal to some value with the same probability for all extensions of the base field. Moreover, thanks to [Kie76], the computability of the local zeta function can be extended from counting the number of points such that $P = 0$ for a tuple of polynomials, to counting the number of points such that ϕ holds, where ϕ is an arbitrary first order formula over finite fields.

Corollary 8.4. *Let ϕ and ψ be two first order formulae built over atoms of the form $P = 0$ with $P \in \mathbb{F}_q[X]$, and with free variables $F \subset X$. One can decide if for all $k \in \mathbb{N}$:*

$$\left|\vec{f} \in \mathbb{F}_{q^k}^{|F|} \mid \phi(\vec{f}) = 1\right| = \left|\vec{f} \in \mathbb{F}_{q^k}^{|F|} \mid \psi(\vec{f}) = 1\right|$$

Thus, for any two events that can be expressed as a first order formula over a finite field one can verify if they happen with the same probability over all extensions of the base field. Remark that this cannot be used to decide universal equivalence, as equivalence cannot be expressed by a first order formula.

We first show that $\mathbb{F}_{q^\infty}$-*equivalence* is decidable for arithmetic programs, i.e. programs without conditionals or conditioning. The difficulty is to make it so that we check equality of the distributions over all possible outputs, and not over the output 0. To this end, we express the distributions as vectors, and show how to encode the two-norm of the distances between the distributions.

Given an enumeration $1 \leq j \leq q^{kn}$ of the elements c_j of $\mathbb{F}_{q^k}^n$, for any programs $P_1, P_2 \in \overline{\mathcal{P}}_{\mathbb{F}_q}(\emptyset, R)$ where $|P_1| = n$, we denote by $\overrightarrow{P_1, P_2}^k = ([P_1, P_2]_{\mathbb{F}_{q^k}}(\vec{c_1}, \vec{0}), \ldots, [P_1, P_2]_{\mathbb{F}_{q^k}}(\vec{c_{q^{kn}}}, \vec{0}))$, that completely characterizes the distribution of P_1 conditioned by $P_2 = 0$. Notice that when $|R| = m$, we have:

$$q^{km}\overrightarrow{P_1, P_2}^k = (N_k(P_1 + \vec{c_1}, P_2), \ldots, N_k(P_1 + \vec{c_{q^{kn}}}, P_2))$$

The core of the reduction to LRS is that the squared norm-two of $\overrightarrow{P_1, P_2}^k - \overrightarrow{Q_1, Q_2}^k$ is a LRS. As we have that

$$P_1 \mid P_2 \approx_{\mathbb{F}_{q^\infty}} Q_1 \mid Q_2 \ \Leftrightarrow\ \forall k \in \mathbb{N}.\ \|\overrightarrow{P_1, P_2}^k - \overrightarrow{Q_1, Q_2}^k\|_2^2 = 0$$

This allows us to directly conclude decidability, as we can decide if the corresponding LRS is always zero.

[4] The following reductions do not hold for equivalence, it is the reason why we considered conditional equivalence. It works as equivalence trivially reduces to conditional equivalence.

Lemma 8.10. *Let P_1, P_2, Q_1, Q_2 in $\overline{\mathcal{P}}_{\mathbb{F}_{q^k}}(\emptyset, R)$. We have that $\|\overrightarrow{P_1, P_2}^k - \overrightarrow{Q_1, Q_2}^k\|_2^2$ is a LRS.*

Proof. Using the classical inner product $\vec{x} \cdot \vec{y} = \sum_i x_i y_i$, for any k and programs $U, V, U', H' \in \overline{\mathcal{P}}_{\mathbb{F}_{q^k}} \emptyset, R$, we have, when σ is a mapping from variables in R to fresh variables in R' and $|R| = m$:

$$\begin{aligned} N_k((U - V\sigma, U', V')) &= \left|\left\{ X, X' \in \mathbb{F}_{q^k}^m \mid \ U(X) = V(X') \wedge (U'(X), V'(X)) = \mathbf{0} \ \right\}\right| \\ &= \sum_{c \in \mathbb{F}_{q^k}^n} \left|X \in \mathbb{F}_{q^k}^m \mid U(X) = c \wedge U'(X) = \mathbf{0}\right| \\ &\qquad \times \left|X \in \mathbb{F}_{q^k}^m \mid V(X) = c \wedge V'(X) = \mathbf{0}\right| \\ &= \sum_i q^{km} \times \overrightarrow{U, U'}_i^k \times q^{km} \times \overrightarrow{V, V'}_i^k \\ &= \overrightarrow{U, U'}^k \cdot \overrightarrow{V, V'}^k \end{aligned}$$

So now,

$$\begin{aligned} &N_k(P_1 - P_1\sigma, P_2, P_2\sigma) - 2N_k(P_1 - Q_1\sigma, P_2, Q_2\sigma) + N_k(Q_1 - Q_1\sigma, Q_1, Q_1\sigma) \\ &\quad = q^{2km} \times (\overrightarrow{P_1, P_2}^k \cdot \overrightarrow{P_1, P_2}^k - 2\overrightarrow{P_1, P_2}^k \cdot \overrightarrow{Q_1, Q_2}^k + \overrightarrow{Q_1, Q_2}^k \cdot \overrightarrow{Q_1, Q_2}^k) \end{aligned}$$

In other terms:

$$\begin{aligned} &N_k(P_1 - P_1\sigma, P_2, P_2\sigma) - 2N_k(P_1 - Q_1\sigma, P_2, Q_2\sigma) + N_k(Q_1 - Q_1\sigma, Q_1, Q_1\sigma) \\ &\quad = q^{2km} \times \|\overrightarrow{P_1, P_2}^k - \overrightarrow{Q_1, Q_2}^k\|_2^2 \end{aligned}$$

Corollary 8.3 finally allows us to conclude. ∎

We can now conclude decidability of $\mathbb{F}_{q^\infty}$-*equivalence* for arithmetic programs, as we can decide if the corresponding LRS is always zero. Computing the LRS is in fact equivalent to computing the associated local zeta functions, and thus check if the following is equal to 0:

$$Z(P_1 - P_1\sigma, P_2, P_2\sigma) - 2Z(P_1 - Q_1\sigma, P_2, Q_2\sigma) + Z(Q_1 - Q_1\sigma, Q_1, Q_1\sigma)$$

Using the complexity for the computation of the local zeta function provided by [LW06, Corollary 2] we obtain the following corollary.

Corollary 8.5. $\mathbb{F}_{q^\infty}$-conditional equivalence *and* $\mathbb{F}_{q^\infty}$-equivalence *restricted to arithmetic programs are in* EXP.

Removing the conditionals We now wish to remove conditionals, in order to reduce equivalence for programs with conditional to arithmetic programs (which are simply tuples of polynomials). To remove the conditionals, the first idea is to use a classical encoding in finite fields:

$$\left[\textbf{if } B \neq 0 \textbf{ then } P_1^t \textbf{ else } P_1^f\right]_{\mathbb{F}_{q^k}} = \left[P_1^f + B^{q^k-1}(P_1^t - P_1^f)\right]_{\mathbb{F}_{q^k}}$$

This works nicely as B^{q^k-1} is equal to 0 if $B = 0$, else to 1. However, for the universal case, we need to have an encoding which does not depend on the size of the field, i.e., it must be independent of k. The key idea is that for any variable t and polynomial B:

$$(B(Bt-1) = 0 \wedge t(Bt-1) = 0) \ \Leftrightarrow \ t = B^{q^k-2}$$

And thus, we can for instance write, for any program Q and output $\vec{o}$:

$$\begin{aligned}
&\left[\textbf{if } B \neq 0 \textbf{ then } P_1^t \textbf{ else } P_1^f\right]_{\mathbb{F}_{q^k}}(\vec{o}) = [Q]_{\mathbb{F}_{q^k}}(\vec{o}) \\
&\quad\Leftrightarrow \left[P_1^f + B^{q^k-1}(P_1^t - P_1^f)\right]_{\mathbb{F}_{q^k}}(\vec{o}) = [Q]_{\mathbb{F}_{q^k}}(\vec{o}) \\
&\quad\Leftrightarrow \left[P_1^f + Bt(P_1^t - P_1^f), (B(Bt-1), t(Bt-1))\right]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) = [Q]_{\mathbb{F}_{q^k}}(\vec{o})
\end{aligned}$$

An induction on the number of conditionals yields our second lemma.

Lemma 8.11. *For any $k \in \mathbb{N} \cup \{\infty\}$, $\mathbb{F}_{q^k}$-*conditional equivalence *restricted to programs without failures reduces in exponential time to $\mathbb{F}_{q^k}$-*conditional equivalence *restricted to arithmetic programs.*

Removing failures Recall that failures define the probabilistic semantics through normalization. For instance, for a program $(\textbf{if } b = 0 \textbf{ then } P_1 \textbf{ else } \bot, P_2)$ where P_1 and P_2 do not fail and b is a polynomial, for any $\vec{o}$, we have:

$$[(\textbf{if } b = 0 \textbf{ then } P_1 \textbf{ else } \bot, P_2)]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) = \frac{\mathbb{P}\{P_1 = \vec{o} \wedge P_2 = \mathbf{0} \wedge b = 0\}}{\mathbb{P}\{\neg(b = 0)\}}$$

Handling this division by itself would be difficult if we wanted to compute the distribution. However, in our setting, we are comparing the equality of two distributions, so we can simply multiply on both side by the denominator, and try to express once again all factors as an instance of conditional equivalence. We will be able to push in conditional equivalence some probabilities, as $[P]_{\mathbb{F}_{q^k}}(\vec{o}) \times \mathbb{P}\{b = 0\} = [P, b]_{\mathbb{F}_{q^k}}(\vec{o}, 0)$ when all variables in b do not appear in P.

As an illustration of how to remove the failures, with some program Q, we have:

$$\begin{aligned}
\textbf{if } b = 0 \textbf{ then } P_1 \textbf{ else } \bot \mid P_2 \approx_{\mathbb{F}_{q^k}} Q \mid 0 &\Leftrightarrow \forall \vec{o}.\, [(\textbf{if } b \textbf{ then } P_1 \textbf{ else } \bot, P_2)]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) = [Q]_{\mathbb{F}_{q^k}}(\vec{o}) \\
&\Leftrightarrow \forall \vec{o}. \mathbb{P}\{P_1 = \vec{o} \wedge P_2 = \mathbf{0} \wedge b = 0\} = \mathbb{P}\{\neg(b = 0)\}\, [Q]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) \\
&\Leftrightarrow \forall \vec{o}.\, [P_1, P_2, b]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) = \mathbb{P}\{\neg(b = 0)\}\, [Q]_{\mathbb{F}_{q^k}}(\vec{o})
\end{aligned}$$

To reduce to an instance of conditional equivalence, the issue is that we need to express as an equality the disequality $b \neq 0$. With some fresh variable t, multiplying by $\mathbb{P}\{\neg(b = 0)\}$ or conditioning on $tb - 1 = 0$ is equivalent, as b has an inverse if and only if it is different from zero. We can thus have:

$$\begin{aligned}
\textbf{if } b = 0 \textbf{ then } P_1 \textbf{ else } \bot \mid P_2 \approx_{\mathbb{F}_{q^k}} Q \mid 0 &\Leftrightarrow \forall \vec{o}.\, [P_1, P_2, b]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) = \mathbb{P}\{\neg(b = 0)\}\, [Q]_{\mathbb{F}_{q^k}}(\vec{o}) \\
&\Leftrightarrow \forall \vec{o}.\, [P_1, P_2, b]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) = [Q, tb - 1]_{\mathbb{F}_{q^k}}(\vec{o}, 0) \\
&\Leftrightarrow P_1 \mid P_2, b \approx_{\mathbb{F}_{q^k}} Q \mid tb - 1
\end{aligned}$$

Universal equivalence Using those techniques, we obtain:

Lemma 8.12. *For any $k \in \mathbb{N} \cup \{\infty\}$, $\mathbb{F}_{q^k}$-*conditional equivalence *reduces to $\mathbb{F}_{q^k}$-*conditional equivalence *restricted to programs without failures in exponential time.*

The previous Lemmas allows us to conclude.

Theorem 8.3. $\mathbb{F}_{q^\infty}$-equivalence *and* $\mathbb{F}_{q^\infty}$-conditional equivalence *are in* 2-EXP.

Independence Using once again Lemmas 7.2 and 7.3, we obtain the same complexity results for the independence problem.

Corollary 8.6. $\mathbb{F}_{q^\infty}$-conditional independence *is in* 2-EXP.

Moreover, we can also extend the lower bound obtained for *q-equivalence*.

Lemma 8.13. $\mathbb{F}_q$-equivalence *reduces in polynomial time to* $\mathbb{F}_{q^\infty}$-equivalence.

Universal zero-majority without inputs For arithmetic programs, we have reduced $\mathbb{F}_{q^\infty}$-0-*majority* to the positivity problem for LRS. The generalization to general programs with conditionings and branchings is similar to the reductions for universal equivalence. We thus obtain the following result.

Theorem 8.4. $\mathbb{F}_{q^\infty}$-0-majority *for inputless programs reduces in exponential time to the positivity problem for simple LRS.*

The reduction can also be applied with the generalization of [Kie76], and thus, for any two events about programs over finite fields, one can, given an oracle for the positivity problem, decide if the probability of the first event is greater than the second one for all extensions of the base field.

We also remark that similarly to $\mathbb{F}_{q^\infty}$*-equivalence*, the complexity of the problem strongly comes from the presence of multiplications. Indeed, in the linear case, majority implies equivalence and we obtain the following.

Lemma 8.14. $\mathbb{F}_{q^\infty}$*-0*-majority *restricted to linear programs is in* PTIME.

Similarly to the equivalence case, we can derive some hardness from the non universal case, but we do not obtain any completeness result.

Lemma 8.15. $\mathbb{F}_{2^\infty}$-0-majority *is* PP*-hard.*

Compared to equivalence, we do not have a way to reduce majority or 0-majority programs without inputs. Thus, we are not able to generalize the reduction to the positivity problem for those cases.

8.4 Program Indistinguishability

Section Summary

To reason about computational indistinguishability of programs, we study and define program indistinguishability and the LRS negligibility problem. We make some first steps towards the decidability of program indistinguishability by proving the decidability of the LRS negligibility problem, but leave the decidability of program indistinguishability open, except for binary programs, that only return a boolean. In the special case of binary programs, we show the decidability of program indistinguishability, using the same reduction to LRS as for the case of universal equivalence.

A variant of equivalence that is of interest for security proofs is indistinguishability. Intuitively, it means that the statistical distance between two programs is negligible w.r.t. some security parameter.

Definition 8.1. We say that two programs P, Q are indistinguishable, denoted by $P \sim Q$, if for all $d \in \mathbb{N}$ there exists K_d such that:

$$\forall k > K_d. \sum_{\vec{o} \in \mathbb{F}_{q^k}^n} |\, [P]_{\mathbb{F}_{q^k}}(\vec{o}) - [Q]_{\mathbb{F}_{q^k}}(\vec{o})| \leq \frac{1}{k^d}$$

Or equivalently:

$$P \sim Q \Leftrightarrow \forall k > K_d.\ \|\vec{P}^k - \vec{Q}^k\|_1 \leq \frac{1}{k^d}$$

Indistinguishability of programs is of course implied by equivalence, but the converse is not true. Consider for instance the program that always outputs 0, $P :=$ return 0, and the program $Q := x \xleftarrow{\$} \mathbb{D}$, **if** $x = 0$ **then** return 1 **else else** 0.

This is very close to the definition of computational indistinguishability. A widely known fact (see e.g. [Gol05]) is that the negligibility of the statistical distance implies the computational indistinguishability of the two programs: no attacker can guess with which of two programs they interact. In other terms, abusively denoting programs as protocols outputting some value, we have $P \sim Q \Rightarrow P \cong Q$.

We provide some first insight about the program indistinguishability problem by showing that the corresponding LRS problem is decidable (this relies heavily on the techniques of [OW14b] and on some of its notations, that we do not recall here). The decidability for LRS implies that if one can find a way to express $\|\vec{P}^k - \vec{Q}^k\|_1$ as a LRS, program indistinguishability is decidable. The reduction would also work if any polynomial over $\|\vec{P}^k - \vec{Q}^k\|_1$ can be seen as a LRS, as any function is negligible if and only if any polynomial in this function is negligible.

Technical Details

Classically, a positive function $f : k \mapsto f(k)$ is negligible if:

$$\forall d, \exists K_d, \forall k > K_d.\ f(k) \leq \frac{1}{k^d}$$

Notably, for any $x < 1$, $k \mapsto x^k$ is negligible.

Definition 8.2. A simple integer LRS $\langle u_k \rangle$ is negligible if:

$$\forall d, \exists K_d, \forall k > K_d.\ |u_k| \leq \frac{1}{k^d}$$

Theorem 8.5. *Let M be the maximal modulus of the roots of a simple integer LRS $\langle u_k \rangle$. $\langle u_k \rangle$ is negligible if and only if $M < 1$.*

Proof. We perform a case study on the maximal modulus of the roots, after removing the case of degenerate LRS.

Degenerate LRS First, remark that for any M, $\langle u_k \rangle$ is negligible if and only if for all d between 0 and $M-1$, $\langle u_{kM+d} \rangle$ is negligible. Indeed, if any of the sub-LRS is non negligible, the LRS is also non negligible. If all the sub-LRS are negligible, so is the LRS.

From now on, we only consider non degenerate LRS.

General form of non degenerate LRS There exist integers a_1, a_2 and algebraic numbers $\alpha_1, \ldots, \alpha_{a_1}, \beta_1, \ldots, \beta_{a_2}$ such that for any $k \geq 1$:

$$u_k = \sum_{j=1}^{a_2} \beta_j^k - \sum_{j=1}^{a_1} \alpha_j^k$$

Maximal module $M < 1$ We have by triangle inequality that $\forall k.\ |u_k| \leq (a_1 + a_2)M^k$. If m is smaller than one, $k \mapsto (a_1 + a_2)M^k$ is a negligible function, and thus the LRS is negligible.

Maximal module $M \geq 1$ We will use Braverman's Lemma [Bra06], and therefore assume that the LRS is not always zero (else it is trivially negligible).

Let M be the maximum module of $\Lambda = \{\alpha_i, \beta_j\}$. We consider $\Lambda_{max} = \{x \in \Lambda \mid |x| = M\}$.

Assume that $M \notin \Lambda$. This means that $\langle u_k \rangle$ has no dominant real root. Then, we can write $\sum_{j=1}^{a_2} \beta_j^k - \sum_{j=1}^{a_1} \alpha_j^k$ as $\sum_{x \in \Lambda_{max}} x^k + r_k$, where $r_k = o(M^k)$. Applying Braverman's Lemma to $\sum_{x \in \Lambda_{max}} \frac{x^k}{M^k}$, we get c such that infinitely often, $|\sum_{x \in \Lambda_{max}} \frac{x^k}{M^k}|$ goes above c and below $-c$. Finally, there exists some ϵ such that infinitely often $|\sum_{j=1}^{a_2} \beta_j^k - \sum_{j=1}^{a_1} \alpha_j^k| \leq (-c + \epsilon)M^k$ and $|\sum_{j=1}^{a_2} \beta_j^k - \sum_{j=1}^{a_1} \alpha_j^k| >= (c - \epsilon)M^k$. As $M \geq 1$, we have that there exists an infinite number of k such that $|u_k| \geq (c - \epsilon)$. Then, infinitely often, $|u_k|$ is bigger than a constant, and hence non-negligible.

Assume that $M \in \Lambda$. We denote by $c_\alpha = \{1 \leq i \leq a_1 \mid \alpha_i = M\}$ and $c_\beta = \{1 \leq i \leq a_\beta \mid \beta_i = M\}$. Then, with $c = c_\alpha - c_\beta$, we consider $v_k = u_k - c \times M$. If v_k has no dominant real root, we can apply Braverman's Lemma to v_k. Then, infinitely often $v_k > 0$ and $v_k < 0$, and thus infinitely often $u_k - c \times M > 0$ and $u_k - c \times M < 0$, i.e., $u_k > c \times M$ and $u_k < -c \times M$. No matter whether c is positive or negative, we have that $|u_k| > |c \times M|$ and $\langle u_k \rangle$ is not negligible.

■

We are only able to show that the negligibility of $\|\vec{P}^k - \vec{Q}^k\|_2^2$ is decidable, as thanks to Lemma 8.10 it is a simple LRS. This provides a necessary condition and a weak sufficient condition for $P \sim Q$, as for any k:

$$\|\vec{P}^k - \vec{Q}^k\|_2 \leq \|\vec{P}^k - \vec{Q}^k\|_1 \leq q^{km/2}\|\vec{P}^k - \vec{Q}^k\|_2$$

Notice however that if we consider programs that always return either 0 or 1, we have that $\|\vec{P}^k - \vec{Q}^k\|_1 = \|\vec{P}^k - \vec{Q}^k\|_2^2$. This observation allows to obtain the following corollary, combining Lemma 8.10 and the fact that the negligibility of a simple LRS is decidable.

Corollary 8.7. *Program indistinguishability restricted to programs that always output either 0 or 1 is decidable.*

Unfortunately, we leave the decidability in the general case as an open question.

8.5 Undecidability with Loops

Section Summary

We prove undecidability of universal equivalence for programs with loops over finite fields. This is done by reduction from the halting problems of two counter machines.

Assuming the same guards b as in the conditionals (Figure 7.1), we add the $\mathsf{while}\ b\ \mathsf{do}\ c$ construct to our language. The associated semantics is natural and not detailed (note that we also extend the semantics of variables to be used in loops). The semantics of a program that does not terminate is given a specific value $\bot^*$. Then, the uniform equivalence problem of this enriched language is undecidable. We reduce the halting problem for two counter Minsky Machines.

A Minsky Machine, or counter machine, is a 3 tuple (C, L, I) where

- $C = \{c_1, \ldots, c_l\}$ is a set of counters;
- $L = \{l_1, \ldots, l_m\}$ is an ordered set of labels;
- and $I = \{i_1, \ldots, i_m\}$ is an ordered set of instructions.

For each instruction i_j, l_j is the associated label, used for jumps. Instructions are of the form:

$$\begin{aligned} i := \quad & \mathsf{incr}(c_k); \mathsf{JUMP}(l_j) \\ \mid\ & \mathsf{decr}(c_k); \mathsf{JUMP}(l_j) \\ \mid\ & \textbf{if } c_k = 0 \textbf{ then } \mathsf{JUMP}(l_s) \textbf{ else } \mathsf{JUMP}(l_t) \\ \mid\ & \mathsf{HALT} \end{aligned}$$

A configuration of the machine is given as a couple $(n_1, \ldots, n_l), i$ where $(n_1, \ldots, n_l) \in \mathbb{N}^l$ and $i \in I$. Intuitively, the configuration gives explicitly a value for all the counters of the machine, and stores in a dedicated register the current instruction to be executed. The one step reduction of a machine M is denoted by $\to_M$, defined by:

$$\begin{aligned} &(n_1, \ldots, n_l), (\mathsf{incr}(c_k); \mathsf{JUMP}(l_j)) \to_M (n_1, \ldots, n_k + 1, \ldots, n_l), i_j \\ &(n_1, \ldots, n_l), (\mathsf{decr}(c_k); \mathsf{JUMP}(l_j)) \to_M (n_1, \ldots, n_k - 1, \ldots, n_l), i_j \text{ (when } n_k > 0) \\ &(n_1, \ldots, n_l), (\textbf{if } c_k = 0 \textbf{ then } \mathsf{JUMP}(l_s) \textbf{ else } \mathsf{JUMP}(l_t)) \to_M (n_1, \ldots, n_l), i_s \text{ (when } n_k = 0) \\ &(n_1, \ldots, n_l), (\textbf{if } c_k = 0 \textbf{ then } \mathsf{JUMP}(l_s) \textbf{ else } \mathsf{JUMP}(l_t)) \to_M (n_1, \ldots, n_l), i_t \text{ (when } n_k \neq 0) \end{aligned}$$

We denote by $\to_M^*$ its transitive closure.

The halting problem for two counter machines is undecidable, i.e., given a machine M and an initial configuration C, r, one cannot decide if there exists a value C' of the counters such that $C, r \to_M^* C', \mathsf{HALT}$.

Theorem 8.6. $\mathbb{F}_{q^\infty}$-equivalence *is undecidable for programs with loops.*

Proof. Let $M = (C, r, L, I)$ be a two counter machine where $C = (\{c_1, c_2\}$, $L = \{l_1, \ldots, l_m\}$ and $I = \{i_1, \ldots, i_m\}$, and an initial configuration $(n_1, n_2), i_s$.

We build a program over $\mathbb{F}_q$ which emulates the counter machine execution, and which will be such that it never terminates in all interpretations if and only if M does not terminate. Then, the

program will be universally equivalent to a program which never halts if and only if M does not terminate, which is the expected reduction.

We choose q to be the smallest prime number bigger than m, and we assume, without loss of generality, that the only halt instruction of M is l_1. We can then emulate r with a single variable, where the macro $\mathsf{JUMP}(l_i)$ is simply $r := i$.

If we denote by $[i]$ the encoding of an instruction i defined later, the core of the program is then:

$$
\begin{array}{l}
r := s \\
\mathsf{while}\ r \neq 1\ \mathsf{do} \\
\quad \textbf{if}\ r = 2\ \textbf{then} \\
\quad\quad [i_2] \\
\quad \ldots \\
\quad \textbf{if}\ r = m\ \textbf{then} \\
\quad\quad [i_m]
\end{array}
$$

We now provide encodings for each instruction. We first define a dummy non halting program $\mathit{Loop} := \mathsf{while}\ 0 = 0\ \mathsf{do}\ t \xleftarrow{\$} \mathbb{D}$ (the sampling of t is an alias for no operation).

To model the counters, we sample a pair of variables $x_1, x_2 \xleftarrow{\$} \{\mathbb{D} \mid x_1 x_2 = 1\}$, and a counter of value n is represented by x_1^n. In this representation incrementing the counter corresponds to multiplication by x_1, and decrementing is achieved by multiplication with x_2 (the inverse of x_1).

Assuming that we are given some variables x_1, x_2 and c_1, c_2, we define a function $[i]$ such that:

$$
\begin{array}{ll}
[\mathsf{incr}(c_k); \mathsf{JUMP}(l_j)] & = c_k := c_k \times x_1; \textbf{if}\ c_k = 1\ \textbf{then}\ \mathit{Loop}\ \textbf{else}\ r := j \\
[\mathsf{decr}(c_k); \mathsf{JUMP}(l_j)] & = \textbf{if}\ c_k = 1\ \textbf{then}\ \mathit{Loop}\ \textbf{else}\ c_k := c_k \times x_1; r := j \\
[\textbf{if}\ c_k = 0\ \textbf{then}\ \mathsf{JUMP}(l_s)\ \textbf{else}\ \mathsf{JUMP}(l_t)] & = \textbf{if}\ c_k = 0\ \textbf{then}\ r := s\ \textbf{else}\ r := t
\end{array}
$$

The final program P is then:

$$
\begin{array}{l}
x_1, x_2 \xleftarrow{\$} \{x_1 x_2 = 1\} \\
c_1 := x_1^{n_1}; c_2 := x_1^{n_2}; r := s \\
\mathsf{while}\ r \neq 1\ \mathsf{do} \\
\quad \textbf{if}\ r = 2\ \textbf{then} \\
\quad\quad [i_2] \\
\quad \ldots \\
\quad \textbf{if}\ r = m\ \textbf{then} \\
\quad\quad [i_m] \\
\mathsf{return}\ 0
\end{array}
$$

To conclude the proof, we now prove that

$$P \approx_{\mathbb{F}_{q^\infty}} \mathit{Loop} \Leftrightarrow \text{M does not halt on input } (n_1, n_2), i_s$$

It is clear that without an overflow, i.e., when the multiplicative group generated by x_1 is big enough to avoid the case $c_k = 1$ in the encodings of incr, P perfectly simulates the behaviour of M, and terminates if and only if M terminates.

Let us assume that M does not halt on input $(n_1, n_2), i_s$. Given an interpretation $\mathbb{F}_{q^k}$, and a sampled value x_1, x_2, we have counters that can evolve in the cyclic multiplicative group generated

by x_1, of some size $q^{k'}$. For any such k', either the simulation of M will create an overflow (increasing a counter over k'), and then P does not terminate. Else, there is a loop of instructions in M, which will be perfectly mimicked by P, which then does not terminate. Thus, for any interpretation and any random samplings, P does not terminate, and then $P \approx_{\mathbb{F}_{q^\infty}} \mathit{Loop}$.

Let us assume that M does halt on this input. We have an upper bound K on the values of the counter during the execution. Thus, there exists some k such that $q^k > K$, and there exists a random sampling of x_1 such that its generated multiplicative group is of size q^k. Then, the execution of P simulating P will not overflow, and P will terminate, going out of the while loop and returning 0. This execution is then a witness that $P \not\approx_{\mathbb{F}_{q^k}} \mathit{Loop}$, and thus $P \not\approx_{\mathbb{F}_{q^\infty}} \mathit{Loop}$

■

9 In Practice

> I never expect men to give us liberty. No, women, we are not worth it until we take it.
>
> *(Voltairine de Cleyre)*

9.1 Introduction

We previously defined relational properties between probabilistic programs that can be used to perform elementary proof steps in the computational model. We studied the complexity and decidability of such problems, and notably obtained the decidability of universal equivalence. The complexity of the decision procedure is however exponential.

As we have shown that no efficient property is likely to exist we will design heuristics to ease the process of proving security protocols. To be used in practice, we claim that heuristics should be principled: they must have clear theoretical foundations, so that we understand how we may use them and what are their limitations. Following those guidelines, tools based on such heuristics should be easy to use, extend and maintain.

To provide such heuristics, our approach is to leverage widely studied techniques from symbolic cryptography. We can through symbolic reasoning abstract away all probabilities, and simplify some of the properties of finite fields, for instance abstracting them by commutative rings of a given characteristic. Our approach is modular, defining first a symbolic characterization of equivalence, that is then used to link the equivalence and independence problem to deducibility and static equivalence.

Motivated by those new links, we also extend the state of the art of deducibility and static equivalence for groups, finite fields and ring theories. We then put in practice our heuristics, by developing a library integrated into two mechanized cryptographic provers, EasyCrypt [BGH+11; BDG+13] and MaskVerif [BBD+15].

Chapter Summary

In this Chapter, we focus on deriving principled and automated proof methods for universal equivalence and independence that can be used in cryptographic proofs. To decide these problems, our approach is to leverage in a modular way existing techniques from symbolic cryptography. This methodology completely abstracts away probabilities and provides syntactic rather than semantic reasoning techniques.

We then extend the decision procedures for some of the techniques from symbolic cryptography, and implement our heuristics in a library. The library is integrated in two cryptographic provers, improving their automation.

9.1.1 Our Contributions

Based on the semantic characterization of equivalence through bijections (Proposition 7.1), we give a sound and complete *syntactic* characterization of equivalence. The characterization is based on the notion of primal algebra used previously for proving decidability of unification in the theory of finite fields [Nip90]. The syntactic characterization replaces the existence of a bijection by the existence of a term satisfying specific syntactic properties.

We then leverage (and sometimes extend) methods from symbolic cryptography, including deducibility, deduction constraints and static equivalence, to check the syntactic characterization of our properties.

Our abstract framework allows us to derive sound and complete algorithms, as well as heuristics that may be only sound or only complete. Given the high complexity, or lack of decision procedures, such heuristics are of particular interest in practice. Previously mentioned tools for proof mechanization do use some heuristics, but they often lack theoretical foundations, leading to a misunderstanding regarding the precision and limitations. Our results clarify these questions for the heuristics we propose.

In particular, in the case of finite fields of a fixed size we obtain sound and complete algorithms. Even though we showed that this problem has high computational complexity, our algorithms appear to be more efficient in practice than the straightforward ones. While the case of finite fields of a fixed size is already useful in some cases, cryptographic proofs

- are often performed for an abstract size of the finite field (universal equivalence),
- may require complex combinations of function symbols, where non interpreted function symbols may capture attacker actions (such as in the BC logic).

Thanks to our framework, we can however derive the soundness and/or completeness of many different heuristics for this universal settings. For instance, to prove program equivalence over $\mathbb{F}_{2^n}$ for all n, it follows from our results that it is sound to prove their equivalence over a commutative ring of characteristic 2.

To leverage our results, we prove the decidability of deducibility in the theory of the Diffie-Hellman exponentiation, based on decision procedures for rings and finite fields. The decision procedures are based on techniques from Gröbner bases. This is a contribution of independent interest, as it can also be leveraged to automate the application of the Decisional Diffie-Hellman assumption or generalized to reason about matrices and the Learning With Error assumption [BGS15; BFG+18].

We demonstrate the usefulness of our approach in practice through the implementation of a library that we interfaced with two existing tools: EasyCrypt [BGH+11], and MaskVerif [BBD+15]. We do not implement all heuristics or decision procedures discussed in this Thesis, but the ones implemented are sufficient to improve the existing tools. The source codes of the library and modified tools are available online: [Seq; Ecs; Mvs]. We consider examples in the area of masking, which provide challenging examples of probabilistic information flow. In particular, unlike the original MaskVerif tool, our extension allows for insightful feedback as it may provide attack witnesses when proofs fail. Furthermore, the integration of our approach into the EasyCrypt proof assistant improves automation.

9.1.2 Related Work

Our work explores the relationship between probabilistic and symbolic approaches to cryptography. The probabilistic approach focuses on computational or information-theoretic notions of security, which are modelled using probabilistic experiments. The symbolic approach uses methods from universal algebra, automated reasoning and logic to model and reason about security. Both models have been used extensively in the literature, and there is active research to develop formal methods and tools for proving security in these models.

The connection between these two approaches was first established by Abadi and Rogaway [AR02], who prove computational soundness of symbolic security proofs for symmetric key encryption: under specific assumptions, protocols that are secure in the symbolic model are also secure in the computational model. Their seminal work triggered a long series of results for other cryptographic constructions [CKW10]. The difficulty in computational soundness results stems from the fact that the soundness of a security proof requires that every possible behaviour of a computational adversary is captured by a symbolic adversary. In our work, we exploit soundness of symbolic attacks: every symbolic attack (e.g., an attacker deduction) corresponds to a computational attack. This form of soundness is generally obtained by construction, as every symbolic term induces a probabilistic algorithm. This connection originates from the work of Barthe et al [BCG+13] on automatically verifying and synthesizing RSA-based public-key encryption and was further extended in [BGS15] and [BFG+18] to deal with pairing-based and lattice-based cryptography.

Our work is also closely related to approaches to reason about equivalence and simulatability of probabilistic programs. Barthe et al [BDK+10] show decidability of equality for probabilistic programs (without conditionals or oracle calls) over fixed-length bitstrings. Jutla and Roy [JR10] show decidability of simulatability for programs (with conditionals but no oracle calls) over finite-length bitstrings.

Applications of symbolic methods to masking were considered by Barthe et al in [BBD+15; BBD+16], who develop specialized logics to prove different notions of (threshold) non-interference.

Previous work for deducibility in the Diffie-Hellman exponentiation theory only provide partial solutions: for instance, Chevalier et al [CKR+03] only consider products in the exponents, whereas Dougherty and Guttman [DG14] only consider polynomials with maximum degree of 1 (linear expressions).

9.2 Symbolic Characterization

Section Summary

To leverage existing methods from symbolic cryptography, which abstracts probabilities away and only consider syntactic constructs, we need to be able to reason in terms of syntax rather than semantics on our different problems. In this Section, we identify precisely what are the properties required from our algebras to be able to reason only on the syntax, yielding the notion of effective algebra.

An effective algebra is a primal algebra [Nip90], where the syntax is powerful enough to express all possible functions, equipped with sound and faithful equational theories, that models perfectly the equality of the algebra. Using those effective algebras, we are able to translate the semantic characterization of Proposition 7.1 into a purely syntactic characterization, as the

existence of a bijection is replaced by the existence of a term that models a bijection. We say that such a term is R-bijective.

9.2.1 Symbolic Abstraction

We introduce a framework to completely axiomatize Σ-algebras: in some cases the term algebra equipped with an equational theory E gives a fully abstract representation of the Σ-algebra. Recall that we often assimilated the domain D over which we interpret function symbols in Σ, with the Σ-algebra $\underline{D}$ that provides the actual interpretation of the function symbols. We shall not make the same confusion in the remainder of the chapter, as it may now become misleading.

Primal Algebra A central notion of this Section is primality of an algebra in [Nip90]. In the following Definition, recall that t^D corresponds to the interpretation of the term t w.r.t. D (Definition 7.1).

Definition 9.1. $\underline{D}$ is said to be primal if and only if

$$\forall n.\ \forall f : D^n \mapsto D.\ \exists t \in \mathcal{T}(\Sigma, (x_1, ..., x_n)).\ f = t^D$$

Primality expresses that for any function over the interpretation domain, there exists a term whose interpretation is equal to the function. Intuitively, it means that the syntax is expressive enough to capture all possible operations.

Term algebra for $\mathbb{F}_q$ We consider a variant of the term algebra we used up to now for finite fields, in order to have a primal representation of finite fields. Let P be an irreducible polynomial over $\mathbb{F}_p[\alpha]$ of degree k. The $\underline{\mathbb{F}_{p^k}}$-algebra is defined by the signature

$$\Sigma_{\mathbb{F}_{p^k}} = \{0, 1, \alpha, +, *\}$$

and their usual mathematical interpretation with $\mathbb{F}_{p^k}$ seen as ${}^{\mathbb{F}_p[\alpha]}/_{(P(\alpha))}$. Nipkow [Nip90] has shown that the $\Sigma_{\mathbb{F}_{p^k}}$-algebra $\underline{\mathbb{F}_{p^k}}$ is a primal algebra:

Proposition 9.1 ([Nip90])**.** $\underline{\mathbb{F}_{p^k}}$ *is a primal algebra.*

The main idea underlying the proof relies on the encoding of conditionals of the form **if** $x = i$ **then** t_1 **else** t_2. We already presented such an encoding for $\mathbb{F}_{p^k}$ in Section 7.4.1. This allows for a basic encoding of any function as

$$\textbf{if } x = 0 \textbf{ then } f(0) \textbf{ else } \ldots \textbf{if } x = i \textbf{ then } f(i) \textbf{ else } \ldots$$

As we are working on finite sets, this encoding completely captures a function. In the case of booleans ($q = 2$), we basically write down the truth table of the function in a term.

Term algebra for $\mathbb{F}_q^m$ It is interesting to note that $\mathbb{F}_q^m$ can be made primal. We write tuples directly as sequences of elements, for example we denote 000 by $(0, 0, 0)$, or 0_3. The $\mathbb{F}_q^m$-algebra is then defined by the signature

$$\Sigma_{\mathbb{F}_q^m} = \{0_m, (0_k 1 0_{m-1-k})_{0 \leq k \leq m-1}, (0_k \alpha 0_{m-1-k})_{0 \leq k \leq m-1}, +, *\}$$

and their mathematical interpretations, where we extend multiplication and addition to tuples component by component. This is similar to the classical notation for xor on bitstrings.

Defined this way, we still have primality for those algebras:

Proposition 9.2. *$\underline{\mathbb{F}_q^m}$ is a primal algebra.*

Proof. We see f as a function $(\mathbb{F}_q^m)^k \mapsto \mathbb{F}_q^m$ denoted $f(x_1, ..., x_k) = f_1(x_1, \ldots, x_k)...f_n(x_1, \ldots, x_k)$. For $1 \leq i \leq k$ we may also decompose x_i as $x_i^1...x_i^m$ and see each f_i as a function $(\mathbb{F}_q)^{km} \mapsto \mathbb{F}_q$ Then, by primality of $\underline{\mathbb{F}_q}$ (Proposition 9.1), there exists $t_i \in \mathcal{T}(\Sigma_{\mathbb{F}_q}, x_i^j)$ such that $f_i = t_i^{\underline{\mathbb{F}_q}}$. We define t_i' as t_i where we replace:

- 0 with 0_m
- 1 with $0_{i-1}10_{m-i-1}$
- α with $0_{i-1}\alpha 0_{m-i-1}$
- x_l^i with $0_{i-1}10_{m-i-1} \times x_l$ $(= 0_{i-1}x_l^i 0_{m-i-1})$.

We observe that $t_i'^{\underline{\mathbb{F}_q^m}} = (0_{i-1}(t_i)^{\underline{\mathbb{F}_q}}0_{m-i-1})$ and we may have $f = t^{\underline{\mathbb{F}_q^m}}$ with $t = \sum_i t_i'$. ■

Equational theories To fully abstract our algebras, we need to be able to capture equalities between terms. We achieve this using equational theories. Recall that an equational theory E is a set of equalities $\{t_i = u_i\}_i$ where $t_i, u_i \in \mathcal{T}(\Sigma, V)$ for some set of variables V. E induces a relation $=_E$ on terms defined as the smallest equivalence relation that contains equalities in E and that is closed under substitutions of variables by terms, and application of function symbols.

Definition 9.2. An equational theory E is said to be sound ($\Leftarrow$) and faithful ($\Rightarrow$) with respect to D if and only if

$$\forall t_1, t_2 \in \mathcal{T}(\Sigma).\ t_1^D = t_2^D \Leftrightarrow t_1 =_E t_2$$

When E is both sound and faithful we may use $=$ for $=_E$, as the equality over the domain then corresponds exactly to the equality w.r.t. the equational theory.

The equational theory $E_{\mathbb{F}_q}$ We consider the equational theory $E_{\mathbb{F}_q}$ parameterized by n and P such that $q = p^n$, and P is an irreducible polynomial $P \in \mathbb{F}_p[\alpha]$ of degree n. Denoting by $P(\alpha)$ the term corresponding to this polynomial, we define $E_{\mathbb{F}_q}$ as follows.

$$\begin{array}{cc} x + 0 = x & x * 0 = 0 \\ x * 1 = x & x + \cdots + x = 0 \ \textit{(p times)} \\ x + y = y + x & x * \cdots * x = 1 \ \textit{(q-1 times)} \\ x * y = y * x & x * (y + z) = x * y + x * z \\ x + (y + z) = (x + y) + z & x * (y * z) = (x * y) * z \\ P(\alpha) = 0 & \end{array}$$

Proposition 9.3. *$E_{\mathbb{F}_q}$ is sound and faithful with respect to $\underline{\mathbb{F}_q}$.*

Proof. We consider the classical representation of $\mathbb{F}_q$ as $\mathbb{F}_p[\alpha]/(P)$. The Euclidian division by $P(\alpha)$ provides a normal form for any element in $\mathbb{F}_q$.

Soundness: By definition of a field, every equation in $E_{\mathbb{F}_q}$ holds for all field elements. Soundness is thus immediate.

Faithfulness: Let t_1, t_2 be two ground terms such that $t_1^D = t_2^D$. There exists a polynomial $T \in \mathbb{F}_p[\alpha]/(P)$ such that $t_1^D = T = t_2^D$. We can see t_1 and t_2 as polynomials over $\mathbb{F}_p[\alpha]$. Using the equations capturing associativity, commutativity and distributivity, a polynomial can be written in the developed form, and then using the Euclidian division,with $i \in \{1, 2\}$ under the form $t_i =_E$

$Q_iP_i + R_i$ with $deg(R_i) < deg(P)$. As $P(\alpha) =_E 0$, both polynomials reduce to $t_i =_E Q_i * 0 + R_i$, which by with $x * 0 =_E 0$ reduces to $t_i =_E 0 + R_i$, which finally reduces to $t_i =_E R_i$. Now, with the correctness of E, we have that $t_i =_E T$, which by transitivity implies $R_i = T$. We thus have $t_1 =_E R_1 =_E T =_E R_2 =_E t_2$, which concludes the proof. ■

Effective algebra We can finally define the new general class of algebras we will be able to reason about.

Definition 9.3. $(\underline{D}, \Sigma, E)$ is called an *effective algebra* if $\underline{D}$ is a finite primal term algebra over Σ, and E is sound and faithful with respect to $\underline{D}$.

We consider the example where D is a finite field, denoted $\mathbb{F}_q^m$, where q is an explicit value, and m is a parameter. We may for instance study programs manipulating bitstrings of length m using $\mathbb{F}_2^m$

An example of an effective algebra is, for an explicit q, $(\underline{\mathbb{F}_q}, \Sigma_{\mathbb{F}_q}, E_{\mathbb{F}_q})$. From this algebra corresponding to finite fields, we could also obtain an effective algebra $(\underline{\mathbb{Z}_p}, \Sigma_{\mathbb{Z}_p}, E_{\mathbb{Z}_p})$ for commutative rings of characteristic p by removing some equations from $E_{\mathbb{F}_q}$.

We remark that effective algebras provide in the following work equivalences between probabilistic programs and symbolic methods. Yet, if the equational theory is sound but not not faithful, or if the algebra is not primal, we lose completeness of our reductions, but we still keep sound proof techniques. This is what is used to derive, from complete algorithms in the finite case, sound algorithms in the universal case.

9.2.2 Symbolic Characterization

We provide here an extension of Proposition 7.1 based on effective algebras. This is the abstraction that allows us to reason only at a syntactic level. Notice that we cast Definition 7.8 of programs that are R-bijective to terms, simply by considering the definition over straight line programs. Intuitively, if we have an effective algebra, we simply replace the existence of a bijection by the existence of a term that models a bijection. We show in the next Section, how we will be able to check if a term is a bijection using symbolic methods.

Lemma 9.1. *Let $(\underline{D}, \Sigma, E)$ be an effective algebra and $P, Q \in \mathcal{P}_\Sigma(I, R)$*

$$P \approx_D Q \quad \Leftrightarrow \quad \begin{array}{c} \exists T \in \mathcal{T}(\Sigma, X \cup R)^{|R|}. \\ T \ R\text{-bijective} \wedge P =_E Q\{T \mapsto R\} \end{array}$$

Moreover, if E is sound, but not faithful, then the implication from right to left ($\Leftarrow$) still holds.

Proof.

$$
\begin{aligned}
P \approx_D Q \quad & \Leftrightarrow \textit{(Cor 7.1)} \\
& \quad \forall \vec{i} \in \mathcal{D}^{|I|}.\ \exists f_{\vec{i}} \in \mathsf{bij}^R.\ \forall \vec{r} \in \mathcal{D}^{|R|}.\ [\![P]\!]_D^{\vec{i} \cup \vec{r}} = [\![Q]\!]_D^{\vec{i} \cup f_{\vec{i}}(\vec{r})} \\
& \Leftrightarrow \textit{(we set point by point } f \textit{ so that } \forall \vec{i}, \vec{r}, f(\vec{i}, \vec{r}) = f_{\vec{i}}(\vec{r}) \\
& \quad \exists f \ s.t. \forall \vec{i}, (\vec{r} \mapsto f(\vec{i}, \vec{r})) \text{ bijection}. \forall (\vec{i}, \vec{r}) \in \mathcal{D}^{|I \cup R|}. \\
& \qquad [\![P]\!]_D^{\vec{i} \cup \vec{r}} = [\![Q]\!]_D^{f(\vec{i} \cup \vec{r})} \\
& \Leftrightarrow (D \textit{ is primal}) \\
& \quad \exists T \in \mathcal{T}(\Sigma, I \cup R), T \textit{ R-bijective}.\ \forall \vec{s} \in \mathcal{D}^{|I \cup R|}. \\
& \qquad [\![P]\!]_D^{\vec{s}} = [\![Q\{T \mapsto R\}]\!]_D^{f(\vec{s})} \\
& \Leftrightarrow \\
& \quad \exists T \in \mathcal{T}(\Sigma, I \cup R).\ T \textit{ R-bijective}, P^D = Q\{T \mapsto R\}^D \\
& \Leftrightarrow (E \textit{ is sound and faithfull}) \\
& \quad \exists T \in \mathcal{T}(\Sigma, I \cup R).\ T \textit{ R-bijective}, P =_E Q\{T \mapsto R\}
\end{aligned}
$$

■

Example 9.1. Consider again $uv + vw + uw \approx_{\mathbb{F}_2} u$ from Example 7.8. A valid witness of this equivalence is

$$t(u, v, w) = (uv + wu + vw, u + v, u + w)$$

To show that $t = (t_1, t_2, t_3)$ is indeed a valid witness, we show that t is R-bijective, which we do by exhibiting the inverse. We have $t_2 t_3 = uv + uw + wv + u = t_1 + u$. Thus, we have $t_2 t_3 + t_1 = u$. Then, $t_2 + t_2 t_3 + t_1 = v$ and $t_3 + t_2 t_3 + t_1 = w$. Finally, if we set $g(x_1, x_2, x_3) := (x_2 x_3 + x_1, x_2 + x_2 x_3 + x_1, x_3 + x_2 x_3 + x_1)$, we find that $g(t(u, v, w)) = (u, v, w))$, i.e. t is a bijection[1].

9.3 Symbolic Methods for Probabilistic Programs

Section Summary

Using the previous syntactic characterization of equivalence, we now leverage several classical symbolic cryptography techniques to reason about our relational problems:

- deduction is used to check R-bijectivity, and thus uniformity;
- deduction constraints are used to decide equivalence;
- static equivalence provides a negative criterion for equivalence.

Remark that most links are established for linear programs, but recall that in the case of finite fields, conditionals can be encoded using field operations, and in general conditionals can always be seen as part of the syntax of terms, rather than the syntax of programs.

9.3.1 Using Deduction to Check Uniformity

We show that, on effective algebras, deduction can be used to decide uniformity. When the equational theory is sound (which is generally straightforward), but not necessarily complete, deduction can still be used as a proof technique, as in that case our encoding still implies uniformity.

Example 9.2. Consider $\Sigma_\oplus = \{0, +\}$ and the equational theory $E_\oplus$ defined as the subset of $E_{\mathbb{F}_q}$ with $q = 2$ defined over $\Sigma_\oplus$. We have that

$$u + v + w, v + w \vdash_{E_\oplus} u$$

[1] Actually, we show that t has a left inverse g, which implies that t is injective. For a function over a finite set, injective implies bijective, so we conclude that t is a bijection.

witnessed by the term $R = x_1 + x_2$. However,

$$u + v + w, v + w \not\vdash_{E_\oplus} u + w.$$

The intuition behind the following Proposition is given through Corollary 7.1 that links uniformity and bijectivity. As a bijective function is a function which given its outputs allows to recompute its inputs, it can be checked if a function is bijective using deduction.

Proposition 9.4. *Let $(\underline{D}, \Sigma, E)$ be an effective algebra and $P \in \overline{\mathcal{P}}_\Sigma(I, R)$ with $R = \{r_1, \ldots, r_k\}$. We have that*

$$P \approx_D r_1, \ldots, r_k \quad \Leftrightarrow \quad \forall r_i \in R.\ P, I \vdash_E r_i$$

Moreover, if E is sound, but not faithful, the implication from right to left ($\Leftarrow$) still holds.

Proof. We have:

$$\begin{aligned}
P \approx_D r_1, \ldots, r_k \quad &\Leftrightarrow^1 \exists t \in \mathcal{T}(\Sigma, I \cup R).t\ R\text{-bijective} \wedge P =_E t \\
&\Leftrightarrow^2 \exists t \in \mathcal{T}(\Sigma, I \cup R).t\ R\text{-bijective} \wedge P^D = t^D \\
&\Leftrightarrow^3 \exists c : D^{|I|+|R|} \mapsto D^{|R|}.\ \forall \vec{i}.\ \vec{r} \mapsto c([\![I]\!]_D^{\vec{i}}, [\![R]\!]_D^{\vec{r}}) \in \mathsf{bij}^{\mathcal{D}^{|R|}} \wedge\ c([\![I, P]\!]_D^{\vec{i} \cup \vec{r}}) = [\![R]\!]_D^{\vec{r}} \\
&\Leftrightarrow^4 \forall r \in R.\ \exists c_r : D^{|I|+|R|} \mapsto D^{|R|}. \\
&\qquad \forall \vec{i}.(\vec{r} \mapsto (c_{r_1}([\![I]\!]_D^{\vec{i}}, [\![R]\!]_D^{\vec{r}}), \ldots, c_{r_k}([\![I]\!]_D^{\vec{i}}, [\![R]\!]_D^{\vec{r}}))) \in \mathsf{bij}^{\mathcal{D}^{|R|}} \wedge\ c_r([\![I, P]\!]_D^{\vec{i} \cup \vec{r}}) = [\![r]\!]_D^{\vec{r}} \\
&\Leftrightarrow^5 \forall r \in R.\ \exists c_r \in \mathcal{T}(\Sigma, I \cup R).(c_{r_1}, .., c_{r_k})\ R\text{-bijective} \wedge c_r(I, P(I, R)) =_E r \\
&\Leftrightarrow^6 \forall r \in R.\ \exists c_r \in \mathcal{T}(\Sigma, I \cup R).\ c_r(I, P(I, R)) =_E r \\
&\Leftrightarrow^7 \forall r \in R.\ P(I, R), I \vdash_E r
\end{aligned}$$

We detail below each equivalence.
$\Leftrightarrow^1$ By Lemma 9.1.
$\Leftrightarrow^2$ By the soundness and faithfulness of E.
$\Leftrightarrow^3$ t is R-bijective, so for any $\vec{i} \in \mathcal{D}^{|I|}$ and $z \in D^{|R|}$, we may take $c_{\vec{i}}(z) = t^{-1}([\![I]\!]_D^{\vec{i}}, z)$ and we may then define $c([\![I]\!]_D^{\vec{i}}, z) = c_{\vec{i}}(z)$ which is a bijection for any $\vec{i}$. We are basically taking the inverse of t with respect to R.
$\Leftrightarrow^4$ By splitting c over the k dimensions.
$\Leftrightarrow^5$ By primality, soundness and faithfulness.
$\Leftrightarrow^6$ We detail the difficult part:

$$\begin{aligned}
&\forall r \in R, \exists c_r \in \mathcal{T}(\Sigma, I \cup R), c_r(I, P(I, R)) =_E r \\
&\qquad \Rightarrow (c_{r_1}, .., c_{r_k})\ R\text{-bijective}
\end{aligned}$$

For any $\vec{i}$, $\vec{r} \mapsto [\![P]\!]_D^{\vec{i} \cup \vec{r}}$ has a left inverse which is $\vec{r} \mapsto [\![C]\!]_D^{\vec{i} \cup \vec{r}}$. $\vec{r} \mapsto [\![P]\!]_D^{\vec{i} \cup \vec{r}}$ is a function over a finite set, so if it has a left inverse, it is injective, which in a finite settings means that it is bijective (pigeon hole principle). Moreover, if a bijective function has both a right inverse and a left inverse, they are equals. Indeed, if f as for left inverse g ($g \circ f = id$) and for right inverse h ($f \circ h = id$), then $g(x) = g(f \circ h(x)) = g \circ f(h(x)) = h(x)$. Thus, $\forall \vec{i}$, $\vec{r} \mapsto [\![C]\!]_D^{\vec{i} \cup \vec{r}}$ is the inverse of $\vec{r} \mapsto [\![P]\!]_D^{\vec{i} \cup \vec{r}}$, and is in particular a bijection.

■

9.3.2 Deduction Constraints and Unification for Program Equivalence

In this Section we show how deduction constraint as used in symbolic cryptography [MS01] and (equational) unification can be used to verify program equivalence. Deduction constraints gener-

alize deduction from ground terms to terms that contain variables that have to be instantiated by the attacker.

In the following, we denote by $vars(t)$ the set of variables of a term t, and by $\mathsf{dom}(\sigma)$ the domain of a substitution, i.e., the set of variables that are replaced by the substitution.

Definition 9.4. Let Σ be a signature equipped with E, and $\mathcal{X}$ a set of variables. A deduction constraint is an expression $T \vdash_E^? u$ where $T \subseteq \mathcal{T}(\Sigma, \mathcal{X})$ is a set of terms and $u \in \mathcal{T}(\Sigma, \mathcal{X})$ a term.

A deduction constraint system is either $\bot$ or a conjunction of deduction constraints of the form:

$$T_1 \vdash_E^? u_1 \wedge \ldots T_n \vdash_E^? u_n$$

where $T_1, \ldots, T_n$ are finite set of terms, $u_1, \ldots, u_n$ are terms.

A substitution σ with $\mathsf{dom}(\sigma) = \mathcal{X}$ is a solution over variables $\mathcal{X}$ of a deduction constraint system if and only if $\forall i.\ T_i\sigma \vdash_E u_i\sigma$.

A deduction constraint system may satisfy additional properties:

- monotonicity: $\emptyset \subseteq T_1 \subseteq \ldots \subseteq T_n$
- origination: $\forall i.\ vars(T_i) \subseteq vars(u_1, \ldots, u_{i-1})$
- one-turn: $\forall i, j.\ T_i = T_j \wedge vars(u_i) = \emptyset$

Monotonicity and origination are classical notions that are naturally satisfied in the context of security protocols and exploited in decision procedures: monotonicity ensures that the attacker knowledge (the T_is) only grows, and origination ensures that any variable appearing in the attacker knowledge has been instantiated in a previous constraint.

The one-turn property is novel: it requires that the attacker knowledge is invariant and that all variables actually appear in the attacker knowledge. We show in Section 9.4 that extending the signature with a homomorphic function symbol allows to transform a one-turn constraint system into a constraint system that satisfies origination and monotonicity while preserving solutions.

Unification [Kni89] is the problem that, given two terms, asks to find a substitution which makes the terms equal in the equational theory. For any terms u, v, we denote by $\mathtt{mgu}_{\Sigma,E,\mathcal{X}}(u, v)$ the set of most general unifiers of u and v over Σ, equational theory E and variables $\mathcal{X}$. A set of unifiers is a most general set of unifiers if for any unifier σ, there exists a most general unifier μ, such that σ is an instance of μ, i.e., there exists a substitution θ such that $\sigma =_E \mu\theta$.

We now reduce program equivalence to unification and solving of one-turn deducibility constraint systems. For any set of variables $\mathcal{X}$, we denote by $\mathcal{X}^0$ a set of corresponding function symbols of arity 0, one for each element of $\mathcal{X}$.

Lemma 9.2. *Let $(\underline{D}, \Sigma, E)$ be an effective algebra and $P \in \overline{\mathcal{P}}(I, R)$. Let R' be a set of variables such that $|R'| = |R|$ and $R' \cap R = \emptyset$, and let $Q \in \overline{\mathcal{P}}(I, R')$. We have that*

$$P \approx_D Q \Leftrightarrow \begin{array}{c} \exists \sigma \in \mathtt{mgu}_{\Sigma \cup I^0 \cup R^0, E, R'}(P, Q). \\ (\bigwedge_{r \in R}(I, R')\sigma \vdash^? r) \text{ has a solution over } R' \end{array}$$

Moreover, if E is only sound, but not faithful, the implication from right to left ($\Leftarrow$) still holds.

Proof. $\Leftarrow$ We have a unifier of the form $\sigma := \{r'_i \mapsto t_i(I, R, R')\}$ and a solution to the deduction constraints $\sigma_{\vec{r}} := \{r'_i \mapsto u_i(I, R)\}$, we set $T = R'\sigma\sigma_{\vec{r}}$. First, $\sigma\sigma_{\vec{r}}$ is an instance of the mgu, thus we

have $P =_E Q\{R' \mapsto T\}$. Secondly, we have that $\forall r \in R, I, T \vdash r$, thus thanks to Proposition 9.4 and Corollary 7.1, we have that T is R-bijective. We conclude thanks to Lemma 9.1.

$\Rightarrow$ Using Lemma 9.1, we have T R-bijective such that $p =_E Q\{R' \mapsto T\}$. We have $T = (t_1, ..., t_k)$ (with $k = |R|$), and then $\phi := \{r'_i \mapsto t_i\}$ is a valid unifier.

Thus the most general unifier exists ([Nip90] shows that unification in a primal algebra is unitary, so the existence of a unifier implies the existence of the mgu). Let us call the mgu σ. We then have $\sigma_{\vec{r}}$ such that $\phi = \sigma \circ \sigma_{\vec{r}}$. By Proposition 9.4, we have $\forall r \in R, T, I \models r$, and as $T = R\sigma\sigma_{\vec{r}}$, this means that $\sigma_{\vec{r}}$ is a solution to our deduction constraints.

■

Note that in some cases, the most general unifier may not contain any fresh variables. Then the constraint solving problem is simply a deduction problem. For instance, by restricting equivalence to uniformity, the unifier becomes trivial and we obtain the following corollary:

Corollary 9.1. *Let $(\underline{D}, \Sigma, E)$ be an effective algebra and $P_1, \ldots, P_m \in \overline{\mathcal{P}}(I, R)$ all of arity 1, where $R = \{r_1, \ldots, r_n\}$ and $m < n$. Let R' be a disjoint set of variables such that $|R'| = n$. We have that*

$$\begin{array}{c} P_1, \ldots, P_m \\ \approx_D \\ r_1, \ldots, r_m \end{array} \quad \Leftrightarrow \quad \begin{array}{c} \exists P_{m+1}, \ldots, P_n \in \overline{\mathcal{P}}(I, R) \text{ of arity } 1. \\ \forall r \in R.\ (P_1, \ldots, P_n, I) \vdash r \end{array}$$

Moreover, if E is sound, but not faithful, then the implication from right to left ($\Leftarrow$) still holds.

Technical Details

We remark that our notions are closely related to permutation polynomials, defined as follows [MP13]:

Definition 9.5. A polynomial $f \in \mathbb{F}[x]$ is a permutation polynomial if the function $f : c \mapsto f(c)$ induces a permutation over $\mathbb{F}$.

Thanks to Corollary 7.1 we have a direct link between permutation polynomials and programs $P \in \overline{\mathcal{P}}(\emptyset, \{r\})$ over only one random variable r. We can then solve uniformity for those programs in polynomial time with [Kay05], deciding whether a univariate polynomial is a permutation polynomial in PTIME .

However, the one variable case is very limited for our applications and we need to consider multivariate permutation polynomials.

Definition 9.6. A polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ is a permutation polynomial in n variables over $\mathbb{F}$ if the equation $f(x_1, \ldots, x_n) = \alpha$ has exactly q^{n-1} solutions in $\mathbb{F}$ for each $\alpha \in \mathbb{F}$.

This directly corresponds to the fact that f is uniform when sampling $x_1, \ldots, x_n$ at random, as any point α has the same probability of being reached.

It is extended to the multiple polynomial case with the definition of an orthogonal system:

Definition 9.7. A set of polynomials $f_i \in \mathbb{F}[x_1, \ldots, x_n]$, $1 \leq i \leq r$, forms an orthogonal system in n variables over $\mathbb{F}$ if the system of equations $f_i(x_1, \ldots, x_n) = \alpha_i$,$1 \leq i \leq r$, has exactly q^{n-r} solutions in $\mathbb{F}$ for each $(\alpha_1, \ldots, \alpha_r) \in \mathbb{F}^r$.

Recall that in the case of finite fields, we mentioned in Section 8.3.1 that to the notion of exceptional polynomials, polynomials that are permutation polynomials over infinitely many extensions of a base finite field, is linked to universal equivalence.

In the finite case, Corollary 9.1 is actually the reformulation of a well known mathematical result.

Theorem 9.1 ([Nie71])**.** *For every orthogonal system $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$, $1 \leq m \leq n$, over $\mathbb{F}$ and every r, $1 \leq r \leq n - m$, there exist $f_{m+1}, \ldots, f_{m+r} \in \mathbb{F}[x_1, \ldots, x_n]$ so that $f_1, \ldots, f_{m+r}$ forms an orthogonal system in m variables over $\mathbb{F}$.*

We formalize the link between uniformity and orthogonal systems with the following Theorem.

Theorem 9.2. *Program equivalence over finite fields reduces in polynomial time to deciding if a set of polynomials is an orthogonal system.*

Proof. Using Lemma 7.1, we reduce program equivalence over a finite field to program equivalence without input variables, which we reduce to uniformity without input variables using Lemma 7.5, which we reduce to bijection testing of polynomials over random variables using Corollary 7.1, which directly reduces to deciding if a set of polynomials is an orthogonal system. ■

However, deciding efficiently if a set of polynomials is an orthogonal system, is to the best of our knowledge, still an open problem. This is a question of great interest for our problems. Remark that we have derived in the previous Chapter a non trivial class for this question.

9.3.3 Static Equivalence and Non Equivalence

The notion of static equivalence was introduced in [AF01] and its decidability has been first studied in [AC06]. Static equivalence expresses the inability of an adversary to distinguish two sequences of messages. We formally defined it in Definition 2.5 in order to define symbolic indistinguishability.

Example 9.3. Consider again the signature $\Sigma_\oplus$ and the equational theory $E_\oplus$ corresponding to linear boolean expressions introduced in Example 9.2. We have that

$$u \oplus v, v \oplus w, u \oplus w \not\sim_{E_\oplus} u, v, w$$

as the relation $x_1 + x_2 = x_3$ holds on the left hand side but not on the right hand side. However,

$$u \oplus v, v \oplus w, w' \sim_{E_\oplus} u, v, w$$

This notion has similarities to program equivalence. We show that indeed program equivalence implies static equivalence, and hence static non-equivalence may be used to show that two programs are not equivalent.

	$ACUNh$	F_{q^n}	$\bigcup_\Delta$	$\bigcup_T$	$\bigcup$	$\mathcal{R}$
σ	P[GNW00]	EXP[Nip90]	✓[KR94]	?	✓[KR94]	✗$^{Hilbert\ 10th}$
$\vdash$	P[Del06]	EXP	✓[ACD07]	P	?	EXP
$\vdash^?$	P[DKP12]	?	✓[CR10]	?	?	?
$\sim$	P[DKP12]	EXP	✓[ACD07]	?	?	EXP

Legend:
- $ACUNh$ Associative Commutative Unity Nilpotent with an homomorphism. (xor and h())
- $\bigcup_\Delta$ decidability for the disjoint union of decidable theory
- $\bigcup$ decidability for the union of decidable theory
- $\bigcup_T$ decidability for the disjoint union of typed decidable theory
- $\mathcal{R}$ A commutative ring of infinite size.
- q must be an explicit value in the problem
- n may be a variable
- ✓ problem decidable (P,EXP complexity)
- new result

Figure 9.1: Survey of Symbolic Methods Decidability

Proposition 9.5. *Let $\underline{D}$ be a finite primal algebra over Σ with a sound equational theory E, and $P, Q \in \overline{\mathcal{P}}(I, R)$. We have that*

$$P \not\sim_E Q \quad \Rightarrow \quad P \not\approx_D Q$$

Proof. Without loss of generality (by symmetry between P and Q), we have that

$$\exists R_1, R_2. \quad \begin{array}{l} R_1(P) =_E R_2(P) \wedge \\ R_1(Q) \neq_E R_2(Q) \end{array}$$

Now let us assume by contradiction that P and Q have a unifier σ such that $P\sigma =_E Q\sigma$. From Lemma 9.2, we can choose that only P depends on variables R' and have σ that do not affect Q, and thus $Q = Q\sigma = P\sigma$. Then, combined with $R_1(P) =_E R_2(P)$, we obtain that $R_1(P\sigma) =_E R_2(P\sigma)$ and thus $R_1(Q) =_E R_2(Q)$, which contradicts $R_1(Q) \neq_E R_2(Q)$. ■

Example 9.4. The converse does not hold. Consider the boolean algebra $\underline{\mathbb{F}_2}$ and let u and v be random variables. We have that $uv \sim_{E_{\mathbb{F}_2}} u$, but $uv \not\approx_{\mathbb{F}_2} u$, as uv and v do not follow the same distribution.

9.4 Extending Symbolic Results

Section Summary

We presented previously how several symbolic methods could be used to reason about probabilistic programs. We provide a few useful extensions to existing symbolic results, and a summary of the relevant state of the art is given in Figure 9.1.

As a main extension, we provide decision procedures for deducibility for the theory of Diffie-Hellman exponentiation, its extension to bilinear groups, and for the theory of fields. The decision procedures for Diffie-Hellman exponentiation are based on techniques from Gröbner bases. While its purpose is geared toward our previous links, it is an independent contribution on its own. As a side contribution, we show how to encode one-turn deduction constraints into more classical deduction constraints.

9.4.1 Deciding Deducibility for Diffie-Hellman Theories

Diffie-Hellman exponentiation is a standard theory that is used in key exchange protocols based on group assumptions. It is also used, in its bilinear and multilinear version, in the AutoG&P tool for proving security of pairing-based cryptography. In this setting, the adversary (also often called attacker in the symbolic setting) can multiply groups elements between them, i.e., perform addition in the field, and can elevate a group element to any power they can deduce in the field.

The standard form of deducibility problems that arises in this context is defined as follows: let Y be a set of names sampled in $\mathbb{F}_q$, g some group generator, E the equational theory capturing field and groups operations, some set $X \subset Y$, $f_1, ...f_k, h \in \mathbb{F}_q[Y]$ be a set of polynomials over the names, and Γ be a coherent set of axioms of the forms $f \neq 0$ for some polynomial f. The deducibility problem is then:

$$\Gamma \models X, g^{f_1}, ..., g^{f_k} \vdash_E g^h$$

Proposition 9.6. *Deducibility for Diffie-Hellman exponentiation is decidable.*

The algorithm that supports the proof of the proposition proceeds by reducing an input deducibility problem to an equivalent membership problem of the saturation of some $\mathbb{F}_q[X]$-module in $\mathbb{F}_q[Y]$, and by using an extension for modules [Eis13] of Buchberger's algorithm [Buc76] to solve the membership problem.

The reduction to the membership problem proceeds as follows: first, we reduce deducibility to solving a system of polynomial equations. We then use the notion of saturation for submodules and prove that solving the system of polynomial equations corresponding to the deducibility problem is equivalent to checking whether the polynomial h is a member of the saturation of some submodule M. The latter problem can be checked using Gröbner basis computations.

9.4.2 Fields and Commutative Rings

Another problem of interest is deducibility in a field rather than a group. It arises if we want for instance to prove the uniformity of a program over a finite field through Proposition 9.4. The deducibility problem can then be defined as follows: let Y be a set of names sampled in $\mathbb{F}_q$, where q is not explicitly known and the field is thus seen as a commutative ring, E the equational theory capturing field operations, $f_1, ...f_k, h \in \mathbb{F}_q[Y]$ be a set of polynomials over the names, and Γ be a coherent set of axioms. The deducibility problem is then:

$$\Gamma \models f_1, ..., f_k \vdash_E h$$

We emphasize that this problem is in fact not an instance of the problem for Diffie-Hellman exponentiation. In the previous problem, if we look at field elements, the adversary could compute any polynomial in $\mathbb{F}_q[X]$ but they may now compute any polynomial in $\mathbb{F}_q[f_1, ..., f_k]$, the subalgebra generated by the known polynomials.

Decidability is obtained thanks to [SS88], where they solve the subalgebra membership problem using methods based on classical Gröbner basis.

Proposition 9.7. *Deducibility for commutative rings is decidable.*

As the size of the field is often abstracted in the security proofs, we can soundly consider that we have a finite field of infinite size that we may abstract as a commutative ring. We provide a slight extension to make it complete when q, the size of the field, is explicitly known. We would like to capture the fact that in $\mathbb{F}$, we have $x^q = x$ for all the elements. We use techniques coming from boolean Gröbner Basis to solve our problem, which we simply extend to any q.

Theorem 9.3. *Deduction in $\mathbb{F}_q$ for a given q is decidable.*

Proof. With $x_1, \ldots, x_n$ the set of variables in the deduction problem, we use the previous reduction to compute the Gröbner basis GB of the module corresponding to the attacker knowledge in the abstract commutative ring $Z[x_1, .., x_n]$. Then, we may compute the module in $Z_q[x_1, .., x_n]$, by computing the Gröbner basis of $GB \cup (x_i^q - x_i)_{1 \leq i \leq n}$, and then removing the $(x_i^q - x_i)_{1 \leq i \leq n}$ ■

After a slight technical generalization, we use the combination result of [ACD07] to combine the result for deducibility $\mathbb{F}_q$ with the theory of tuples and projections of length m.

Technical Details

Deducibility for union of typed equational theory [ACD07] provides the results of combination for classical equational theories. We show how we can soundly encode a typed equational theory into an untyped one to use their result.

Definition 9.8. We consider a set of types T. A T-typed signature Σ consists of a finite set of functions symbols each with an arity and a type : $f/n : t_1 \times ... \times t_n \to s$. $\mathcal{X}$ is an infinite set of variables, containing infinitely many variable of each type. For any $t \in \mathcal{T}(\Sigma, \mathcal{X})$, we define in the classical sense that t is well-typed, denoted by $t_1 WT$, and we can then have $\tau : \mathcal{T}(\Sigma, \mathcal{X}) \to T$ which gives the type of a term. $(\Sigma, \mathcal{X}, E)$ is a T-typed equational theory if the equations in E are built over $\mathcal{T}(\Sigma, \mathcal{X})$ and are well typed: $\forall (t_1, t_2) \in E, t_1 WT \wedge t_2 WT \wedge \tau(t_1) = \tau(t_2)$

In this setting, equality is only defined on well-typed instances.

Definition 9.9. Given $(\Sigma, \mathcal{X}, E)$ a T-typed equational theory, we define its untyped instance (Σ_t, E_t, ϕ) with :

- $\Sigma_t = \Sigma \cup \{t/1 | t \in T\}$
- $\forall f/n \in \Sigma, h(f(x_1, ..., x_n)) := s(f(t_1(h(x_1)), ..., t_n(h(x_n)))) \; if \; \tau(f) = t_1 \times ... \times t_n \to s$
- $\forall x \in \mathcal{X}, h(x) := \tau(x)(h(x))$
- $\forall x, g(x) := x[t^n \leftarrow t, \forall t \in T]$
- $\phi := g \circ h$
- $E_t := \{(t_1, t_2)\phi | (t_1, t_2) \in E\}$

In the following, we use the deducibility based on inference system (classically known to be equivalent to our previous definition), as shown bellow:

$$[\text{AX}] \; \frac{}{M_1, \ldots, M_n \vdash_E M} \; M \in \{M_1, \ldots, M_n\}$$

$$[\text{FA}] \; \frac{\psi \vdash_E M_1 \quad \ldots \quad \psi \vdash_E M_l}{\psi \vdash_E f(M_1, ..., M_l)} \; f \in \Sigma \qquad [\text{EQ}] \; \frac{\psi \vdash_E M}{\psi \vdash_E M'} \; M =_E M'$$

Lemma 9.3. *Let $(\Sigma, \mathcal{X}, E)$ be a T-typed equational theory and its untyped instance (Σ_t, E_t, ϕ). For any σ WT substitution on $\mathcal{X}$ and $t \in \mathcal{T}(\Sigma, \mathcal{X})$ WT :*

$$t\sigma\phi =_{E_t} t\phi\sigma\phi$$

Proof. We have that $t[x_1, ..., x_n]\sigma\phi = t[x_1\sigma, ..., x_n\sigma]\phi = t\phi[x_1\sigma\phi, ..., x_n\sigma\phi]$ and $t\phi\sigma\phi = t\phi[\tau(x_1)(x_1), ..., \tau(x_n)(x_n)]\sigma\phi = t\phi[\tau(x_1)(x_1)\sigma\phi, ..., \tau(x_n)(x_n)\sigma\phi]$. We conclude by proving that $\tau(x_i)(x_i)\sigma\phi = x_i\sigma\phi$. As σ is well-typed, $\tau(x_i\sigma) = \tau(x_i)$, and we have that $x_i\sigma = f(y_1, ..., y_k)$ with $f : t_1 \times ... \times t_n \to \tau(x_i)) \in \Sigma$. Thus $x_i\sigma\phi = \tau(x_i)(f(t_1(y_1\phi), ..., t_k(y_k\phi))) = g(\tau(x_i)(\tau(x_i)(f(t_1(y_1\phi), ..., t_k(y_k\phi))))) = \tau(x_i)(x_i\sigma\phi) = \tau(x_i)(x_i)\sigma\phi$.

Thus $x_i\sigma\phi = \tau(x_i)(x_i\sigma\phi)$. ■

Lemma 9.4. *$(\Sigma, \mathcal{X}, E)$ a T-typed equational theory and its untyped instance (Σ_t, E_t, ϕ), we have for all $t, u \in \mathcal{T}(\Sigma, \mathcal{X})$ WT:*

$$t =_E u \Leftrightarrow t\phi =_{E_t} u\phi$$

Proof. We proceed by induction on the length of the reduction. $\Rightarrow$ We prove $t =_E u \Rightarrow t\phi =_{E_t} u\phi$
We have the first rule $t \to u' =_E u$ of the form $(t_1, t_2) \in E$. So there exists a position p in t and a substitution σ s.t. $t_p = t_1\sigma$ and $u' = t[t_2\sigma]_p$. We can apply ϕ to those equalities, which yields p' s.t, $(t\phi)_{p'} = t_1\sigma\phi$ and $u'\phi = t\phi[t_2\sigma\phi]_{p'}$. $(t\phi)_{p'} = t_1\phi\sigma\phi$ and $u'\phi = t\phi[t_2\phi\sigma\phi]_{p'}$ i.e., $t\phi =_{E_t} u'\phi$. We conclude by induction.

$\Leftarrow$ We prove $t\phi =_{E_t} u\phi \Rightarrow t =_E u$.
We have $(t_1\phi, t_2\phi) \in E_t$,$p$ position in $t\phi$ and σ substitution s.t $t\phi_p = t_1\phi\sigma$ and $u' = t\phi[t_2\phi\sigma]_p$. $t\phi_p = t_1\phi\sigma$, so $\sigma = \sigma'\phi$, and this substitution yields $t =_E u'$, where we can conclude by induction. ■

We finally have the following results.

Proposition 9.8. *Let $(\Sigma, \mathcal{X}, E)$ be a T-typed equational theory and its untyped instance (Σ_t, E_t, ϕ). Let $\psi = M_1, \ldots, M_n$ be a WT frame and t a WT term.*

$$\psi \vdash_E t \Leftrightarrow \psi\phi \vdash_{E_t} t\phi$$

Proof. We prove by induction on the size of the proofs that $\psi \vdash_E t \Leftrightarrow \psi\phi \vdash_{E_t} t\phi$. If the proof is of size one, t belongs to the frame ([AX]), and this is stable by substitution.
Let us assume that the result is true for any proofs smaller than some $n > 1$.
$\Rightarrow$ We consider the last rule of the proof of $\psi \vdash_E t$.

- [FA] $\dfrac{\psi \vdash_E M_1 \quad \ldots \quad \psi \vdash_E M_l}{\psi \vdash_E f(M_1, ..., M_l)}\ f \in \Sigma$ with $t = f(M_1, ..., M_l)$. By induction hypothesis, we have proofs of $\psi\phi \vdash_{E_t} M_i\phi$, and because $t\phi = s(f(t_1(M_1\phi), ..., t_n(M_n\phi))$, we conclude by multiple [FA] application.
- [EQ] $\dfrac{\psi \vdash_E M}{\psi \vdash_E M'}\ M =_E M'$, we have by induction a proof of $M\phi$, and we do have $M\phi =_{E_t} M'\phi$ by Lemma 9.4.

$\Leftarrow$ We consider the last rule of the proof of $\psi\phi \vdash_{E_t} t\phi$.

- [EQ] $\dfrac{\psi\phi \vdash_{E_t} M}{\psi\phi \vdash_{E_t} t\phi}\ M =_{E_t} t\phi$. We can find t' such that $M = t'\phi$, and we can then conclude by induction.

▶ [FA] $\dfrac{\phi\psi \vdash_{E_t} M_1}{\phi\psi \vdash_{E_t} s(M_1)}\ f \in \Sigma_t$ with $t\psi = s(M_1)$ and $s \in T$ (if the proof ends with a [FA] application, it must be with a typing function). Here, M_1 cannot be written as some $t'\psi$, so we cannot conclude by induction, and we must consider the last rule of the proof of $\phi\psi \vdash_{E_t} M_1$. Its premises will be writable under the form $t_i\psi$, and we can then conclude by induction hypothesis.

■

This Theorem allows us to transform a typed theory into a classical theory, and then use the result of [ACD07] to combine it with other theories.

9.4.3 From One-Step Deduction Constraints to Originated and Monotone Constraints.

The one-step property we introduced has not been studied previously. We thus want to reduce it to more classical deduction constraints in order to leverage existing results. We note that a decision procedure was developed for deduction constraints without origination or monotonicity [ACR+17], but they cover encryption theories, rather than the algebraic theory we are interested in.

Definition 9.10. Given an equational theory E, a symbol function h is homomorphic if

$$\forall n \geq 1.\ \forall f \in \Sigma^n.\ \forall x_1, \ldots, x_n \in \mathcal{X}.\\ h(f(x_1, \ldots, x_n)) = f(h(x_1), \ldots, h(x_n))$$

Definition 9.11. Signatures may contain private function symbol. Then, deducibility under private function symbols $\mathcal{P}$ is defined as, given an equational theory E over Σ, $t_1, \ldots, t_k \in \mathcal{T}(\Sigma, \mathcal{X})$ and $(x_1, \ldots, x_k)$ disjoint variables, $t_1, \ldots, t_k \vdash_E s$ if and only if:

$$\exists C(x_1, \ldots, x_k) \in \mathcal{T}(\Sigma \setminus \mathcal{P}, (x_1, \ldots, x_k)).\ C(t_1, \ldots, t_n) =_E s$$

Lemma 9.5. *We are given a theory (E, Σ), and a one-step deduction constraint which can be written as $T \vdash^? u_1, \ldots, T \vdash^? u_n$, with $T = (t_1, \ldots, t_k) \in \mathcal{T}(\Sigma, \mathcal{X})$ and $u_1, \ldots, u_n \in T(\Sigma)$. If we extend (E, Σ) with a disjoint private homomorphic symbol function $h/1$, yielding (E', Σ'), then:*

$$T \vdash^? u_1, \ldots, T \vdash^? u_n \text{ has a solution}\\ \Leftrightarrow V \vdash^? t_1, \ldots, V \vdash^? t_k, (V, h(t_1), \ldots, h(t_k)) \vdash^? h(u_1),\\ \ldots, (V, h(t_1), \ldots, h(t_k)) \vdash^? h(u_n) \text{ has a solution}$$

Moreover, the new deduction constraint system is monotonic and originated.

Proof. $\Rightarrow$ If we have an assignment σ and, for some term u, a context C such that $C(t_1, \ldots, t_k)\sigma =_{E'} u$, we have a context such that $C(h(t_1), \ldots, h(t_n))\sigma =_{E'} h(u)$ by homomorphism. Moreover, $V \vdash t_1\sigma$ is always satisfied for any σ.
$\Leftarrow$ By the stability of the reductions in a rewriting system by any substitution applied to the names (not appearing in the rewriting system), we have that for some set of constants $C \subset \Sigma^0$ not appearing in E, given $u(\Sigma^0 \subset C, C) \in \mathcal{T}((\Sigma^{\geq 1}), V)$ such that there exists $u' \in T(((\Sigma) \setminus C)$, $u \to_E^* u'$, then we have for any set of terms T of the same size as C, $u(\Sigma^0 \subset C, T)) \to_E u'$, i.e we can replace constants not appearing in the final term by anything.

We have an assignment σ and for some term u, a context $C \in \mathcal{T}(\Sigma, ((x_v^i)_{i \in \mathbb{N}}, x_1, \ldots, x_k))$ such that $C(V, h(t_1), \ldots, h(t_k))\sigma =_{E'} h(u)$. C does not contain the symbol h, as it is private. By applying

the rule that pushes down the h as much as possible both in C and $h(u)$, we get $C'(V, h(V)) \to_E u(h(V)))$, indeed, pushing down as much as possible the h does not block the application of any rule in E on C, and we can push h back at the top at end of the reduction. By using, the previous result, we can then have for any set of terms T known by the attacker $C'(T, h(V)) \to_E u(h(V)))$, and then by going back to C, we get $C(T, h(t_1), \ldots, h(t_k))\sigma =_{E'} h(u)$ which concludes the proofs.

■

9.5 Deriving Heuristics

Section Summary

We can use our framework to derive multiple ways to solve a specific problem, providing principled algorithms that are sound and/or complete. In this Section, we illustrate how such algorithms might be obtained either for general algebras or more precisely for boolean algebras. We focus mostly on the case of boolean algebras, which is of particular interest for cryptography. Nevertheless, procedures for more general settings can be obtained, such as the bilinear setting.

Regarding boolean algebras, a first interesting subcase is uniformity in the linear case, where only the xor is used. It was previously explored and shown useful in [BDK+10]. Using our work we are able to derive more general results, going beyond linearity and uniformity.

In the general case of boolean algebras (xor and conjunction), we develop several heuristics. As deciding equivalence for a given finite field $\mathbb{F}_q$ is at least $\mathsf{coNP}^{\mathsf{C=P}}$-hard (most likely strictly above NP and coNP, unless the polynomial hierarchy collapses; we refer the reader to Section 8.2 for our results on computational complexity) developing more efficient heuristics is particularly important.

Our techniques rely on well established symbolic methods, and novel extensions of these techniques (detailed previously in Section 9.4). A summary of related, existing results is given in Figure 9.1.

9.5.1 Soundness and Completeness

A first important consideration is that when given an algebra and an instance of a problem, we might be unable to solve it using a sound and faithful equational theory, i.e., a theory that matches exactly the algebra. However, our previous results from Section 9.3 allow us to either add equations and maintain completeness, or remove equations maintaining soundness.

Example 9.5. Let r be a random variable and x an input variable.

- $u + x$ is uniformly distributed for any $\underline{\mathbb{F}_q}$, as we can prove that it is uniformly distributed in a ring theory, which is a sound theory for any $\underline{\mathbb{F}_q}$;
- $u \times x$ is never uniform for any $\underline{\mathbb{F}_{2^n}}$, as we can prove that it is not uniform in $E_{\mathbb{F}_2}$, which is a faithful theory for $\underline{\mathbb{F}_{2^n}}$;
- $uv + vw + wu$ is uniform over $\underline{\mathbb{F}_2}$ but not $\underline{\mathbb{F}_4}$, while $E_{\mathbb{F}_2}$ is faithful for $\underline{\mathbb{F}_4}$;
- $u \times u$ is not uniform over a ring, but is over $\underline{\mathbb{F}_2}$, while a ring theory is sound (but not faithful) for $\underline{\mathbb{F}_2}$.

9.5.2 Boolean Algebras: the Linear Case

Consider programs only built on booleans and the xor operator, i.e., without conjunction, which corresponds to an Associative Commutative Unit Nilpotent (ACUN) theory. Uniformity and independence can be decided by program equivalence (see Figure 7.2). Program equivalence can be decided using unification and solving one-turn deduction constraint systems thanks to Lemma 9.2. Unification is solvable in polynomial time for ACUN theories [GNW00]. Solving one-turn deduction constraints for the ACUN theory can be reduced to solving classical deduction constraints for the ACUNh theory where h is a private symbol (Lemma 9.5 of Section 9.4).

Solving deduction constraint in ACUNh is decidable in polynomial time [DKP12]. Note that [DKP12] does not claim to handle private symbols, but they compute a basis of all possible solutions, and there exists a solution with h being private if only if we can find one in the basis that does not use h. The general decision procedures consist of two steps: first a unification, and then the resolution of a linear system over a polynomial ring using Gaussian elimination.

The authors of [BDK+10] solved uniformity in the linear case using Gaussian elimination. Our decision procedures are essentially the same as for uniformity. However we extend the procedures to independence and program equivalence by adding the unification step, and also add support for the non linear case.

9.5.3 Boolean Algebras: the General Case

Consider programs over a general boolean algebra, i.e., including xor and conjunction operators. This setting is unlikely to admit an efficient procedure for solving program equivalence. We therefore use our framework to derive several heuristics.

Example 9.6. Going back to Example 7.8, let $P = uv + vw + vu$. We cannot directly show that $P \approx_{\mathbb{F}_2} u$. However, if we extend P into a program of size 3, we may decide that

$$(uv + vw + uv, u + w, u + v) \approx_{\mathbb{F}_2} (u, v, w)$$

using deduction in finite fields (cf Section 9.4).

Example 9.7. Let u, v, w be three random variables. We can show using deduction in a commutative ring [BFG+18] that:

$$(u, v + uw, w - uw - v) \approx_{\mathbb{F}_2} u, v, w$$

Indeed, $u, v + uw, w - uw - v \vdash u, v, w$ by computing:

$$r, s, t \mapsto r, s - rt - rs, s + t$$

We now consider several heuristics that may be used in this context.

Unification Lemma 9.2 may yield a deduction constraint system with a trivial solution $R' \mapsto R$ that can be checked with deduction. This provides an efficient heuristic for program equivalence.

Example 9.8. Let u be a random variable, x, s, s' three input variables, $P = x(u + s)$ and $Q = x(u + s')$. We have that $x(u + s) \approx_{\mathbb{F}_2} x(u + s')$, by Lemma 9.2 with the unifier $u' \mapsto u + s' + s$ on $x(u + s) =_E x(u' + s')$ which is trivially bijective when seen as a function on u.

Derandomization Given a program, all of its randomness may not be needed to satisfy a given property (this is a trivial consequence of Proposition 7.1). We may for instance prove that a program is uniform even when we replace some of its random variables by input variables. This reduces the domain of the bijection needed to prove uniformity, and may simplify the proof. The same derandomization technique can be applied to equivalence or independence.

Example 9.9. Let u, v be random and x_v, s be input variables. We may solve $\bot_{\mathbb{F}_2}(P, v(u+s))s$ by replacing v with the input variable x_v and prove $\bot_{\mathbb{F}_2}(s, x_v(u+s))$, which follows directly from Example 9.8 and Lemma 7.3.

Solving uniformity through independence Lemma 7.4 may be used to prove uniformity through independence. This may be useful as it may simplify the deducibility constraints obtained by Lemma 9.2.

Example 9.10. Consider again $P = uv + vw + vu$. Applying Lemma 9.2 directly we obtain the deducibility constraint

$$uv + vw + vu, v', w' \vdash^{?} u, v, w$$

which admits no obvious solution. Let s, s' be two input variables. Applying Lemma 7.4, uniformity of P is equivalent to $\bot_{\mathbb{F}_2}(s, uv + vw + vu + s)$, which in turn (Lemma 7.3) holds if and only if

$$uv + vw + vu + s \approx_{\mathbb{F}_2} u'v' + v'w' + v'u' + s'$$

By setting s to $s + s'$, this is directly equivalent to

$$uv + vw + vu + s \approx_{\mathbb{F}_2} u'v' + v'w' + v'u'$$

On this equivalence, Lemma 9.2 provides a unifier $u', v', w' \mapsto u + s, v + s, w + s$ for which the deducibility constraint has a trivial solution.

Brute forcing the witnesses space We know that we can reduce each of the problems we study to testing uniformity of fully random programs, using the encoding given in Figure 7.2. Let $P \in \overline{\mathcal{P}}(I, R)$ with $|P| = m$ where $R = \{r_1, \ldots, r_n\}$. Using Proposition 9.4, uniformity is equivalent to the existence of a program $\alpha \in \overline{\mathcal{P}}(I, R)$ with $|\alpha| = n - m$ such that P, α is uniform, which can be verified using deducibility by Proposition 9.4. Such an α exists if and only if p is uniform.

This yields a sound and complete procedure. However, the procedure has an exponential running time, as we need to search over all possible polynomials. This technique is nevertheless of interest when combined with approximations of the algebra, because the witness found might be valid only using a subset of the equations over the algebra.

9.5.4 Extension to More Complex Algebras

In [ACD07] the question of deducibility or static equivalence for the union of disjoint theories is reduced to the deducibility or the static equivalence in each theory. We extend their result for deducibility to typed equational theory in Section 9.4. By Proposition 9.4, we reduce uniformity to deducibility. Thus, we may use our results on algebras that are the union of disjoint algebras.

We may for instance consider programs over both boolean linear expressions and group exponentiation with linear maps. Indeed, deducibility for boolean linear expressions is decidable in polynomial time [Del06], and deducibility in the bilinear setting was studied in [BFG+18] (which we extend to the case where the finite field is of explicit size in Section 9.4).

We may also consider a combination of any theory extended with free function symbols. The free function symbols might for instance represent arbitrary code, e.g., attacker actions in cryptographic games.

9.5.5 Interference Witnesses

Non-static equivalence allows to find witnesses of non equivalence, and thus non independence, and finally of non interference. The idea is that given $P = (P_1, \ldots, P_k) \in \overline{\mathcal{P}}(I, R)$, every relation of the form $C_1[P_1, \ldots, P_n] = C_2[I']$ where $I' \subset I$ and C_1, C_2 are contexts is a witness that $\not\perp \{P, I'\}$.

The techniques for deciding static equivalence based on Gröbner Basis (Section 9.4.1) provide an algorithm that computes the set of relations satisfied between multiple polynomials. Based on this work, we develop an algorithm that, given a program, returns a set of variables that do not verify non interference, and witnesses to verify the leakage. This is done by computing the set of all relations over the program and the secrets, keeping those that are actual witnesses, and simply outputting the set of all secrets leaked according to the witnesses.

9.5.6 Sampling from Multiple Distributions

We considered that programs were only performing random sampling over a single distribution, the uniform distribution over some algebra. In practice, we often sample random variables over multiple distributions or domains. Our results, and notably Proposition 7.1, can be extended to this case, for instance by considering programs built over $\overline{\mathcal{P}}(I, R_1, \ldots, R_n)$ and exhibiting bijections over each R_i to prove program equivalence.

9.6 Applications

Section Summary

We provide in this Section applications of our techniques, describing how we developed a library based on some of our results and integrated it into two cryptographic tools, EasyCrypt [BGH+11; BDG+13], and MaskVerif [BBD+15]. The use of EasyCrypt is slightly enhanced by simplifying a tactic, and MaskVerif is now able to provide an interference witness in some cases where it fails to prove the non-interference. The implementations are available online [Seq; Ecs; Mvs].

9.6.1 Implementation of a Library

We implemented parts of our framework as an OCaml library [Seq]. Given that our main focus is automation of cryptographic proofs, the library provides procedures to handle programs over finite fields. We implemented Gröbner basis techniques developed previously for deciding deducibility in rings (Section 9.4.1). This allows for rings of both characteristic 0 and 2 to:

- compute the inverse of a function over multiple variables,
- compute the set of relations between elements.

With those building blocks and based on the practical considerations of the previous Section, we thus provide algorithms that can either be sound or complete (Section 9.5.1) to:

- decide uniformity through deducibility (Proposition 9.4);
- heuristically prove uniformity through derandomization (Section 9.5.3);
- provide a witness of interference through static non-equivalence (Section 9.5.5);
- provide a witness of non-interference through uniformity (Lemma 7.4).

In particular our building blocks for rings of characteristic 0 and 2, yield a sound heuristic to show uniformity through derandomization and deducibility for any finite field $\mathbb{F}_{2^n}$. We may also use static non-equivalence to provide a witness of interference which is valid for any finite field $\mathbb{F}_{2^n}$.

9.6.2 Integration in MaskVerif

MaskVerif is a tool developed to formally verify masking schemes, i.e., counter measure against differential power analysis attacks.

MaskVerif allows to check security properties like n-probing security, non-interference (NI) and strong non-interference (SNI). Those notions are strongly related to probabilistic non-interference. All properties require that for any n-tuple of sub-expressions P used in the program the following base property is satisfied: there exists a subset I' of the input I, such that for all $\vec{i}, \vec{i'} \in \mathbb{F}_2^{|I|}$ where $\vec{i}$ and $\vec{i'}$ coincides over variables in I', we have that $[P]_D^{\vec{i}} = [P]_D^{\vec{i'}}$. Furthermore, there is a cardinality constraint on I but this is independent of the property and not relevant here. If we have an algorithm to check the base property for a single tuple, there exists an algorithm that verifies the three properties.

MaskVerif provides a very efficient procedure to check the base property. However it is incomplete and, in case of failure, does not provide a witness of interference. We propose here a new method that limits incompleteness and, importantly, provides a witness of interference.

We use two functionalities of our library. A witness of interference can be obtained when checking static non-equivalence and a witness of non-interference can be obtained by showing uniformity which allows to prove that a given tuple verifies the base property. Those two procedures are sound but not complete, yielding results for all finite fields $\mathbb{F}_{2^n}$.

Combining both procedures, we obtain an efficient algorithm. We first compute a set of secrets that are leaked from the tuple. If this set of secrets is already larger than the expected size of I, we have a proof that the tuple depends on too many inputs. Otherwise, we try to prove that the remainder of the tuple, the part for which we did not find any obvious secret leakage, is actually independent of the remaining secrets by proving that it is uniform. This last step is still incomplete.

The modification of MaskVerif is minor: we first try the original heuristic and if it fails we call the new heuristic based on our library. This change does not affect efficiency when the original heuristic succeeds but allows to prove new examples, such as proving that the masked multiplication proposed in [CS18] is NI and SNI. The key point for this example is to prove independence of tuples of the form:

$$x_1y_1 + (x_1(y_0 + r) + (x_1 + 1)r)$$

where r_0 and r_1 are the random variables.

A major advantage is that we can now provide witnesses of interference, ensuring that a proof failure is not a false negative. For instance, when analysing a masked implementation of an AND gate, among the 3,784 tuples to check, there is a tuple (t_1, t_2, t_3) of the form

$$((a_1b_1 + r_1) + a_1b_0 + a_0b_1 + r_0, r_0, r_1)$$

A simple witness of interference that we obtain is then the equation $t_1+t_2+t_3 = a_1b_1+a_1b_0+a_0b_1$.

Our procedure does output a witness demonstrating that the masked implementation of the Verilog implementation of the AES Sbox as designed in [GMK17] is not NI at order 1.

9.6.3 Integration in EasyCrypt

EasyCrypt [BGZ09; BGH+11] is a mechanized prover that allows to perform proofs of cryptographic protocols and primitives in the computational model. It is based on the code-based game-playing technique [BR06], and notably provides a set of tactics to reason about probabilistic distributions.

The rnd rule In cryptographic games, one may replace an expression by another expression as long as both expressions range over the same distribution. E.g., if an expression is uniformly distributed, one may replace it by a variable that is sampled at random.

In EasyCrypt, this is done via the proof tactic `rnd`. It allows to replace an expression of the form $f(x)$ by x if f is invertible - i.e., if one can give an effective expression for f^{-1}. For instance, we may replace $x \oplus y$ by x because $\oplus$ is an involution, i.e., $(x \oplus y) \oplus y = x$.

Listing 9.1 presents actual examples of the `rnd` tactic, where the bijection inverse is specified by hand.

```
> examples/elgamal.ec
rnd (fun z, z + log (if b then m1 else m0){2})
    (fun z, z - log (if b then m1 else m0){2}).

> examples/cramer-shoup/cramer_shoup.ec
rnd (fun x2 ⇒ (x2 + G2.v * G1.y2)
         * (G1.w * (G1.u' - G1.u))
         + G1.u * (G1.x + G2.v * G1.y)){2}
    (fun r ⇒ (r - G1.u * (G1.x + G2.v * G1.y))
         / (G1.w * (G1.u' - G1.u))
         - G2.v * G1.y2){2}.

> examples/incomplete/oaep/OAEP.eca
rnd (fun x ⇒ x + pad (if b then m0 else m1){2})
    (fun x ⇒ x - pad (if b then m0 else m1){2}).
```

Listing 9.1: Examples of the EasyCrypt rnd tactic

Using our library, it has been possible to enhance the `rnd` tactic by making the bijection inverse expressions optional. In that case, EasyCrypt tries to automatically compute the inverse. Our library is powerful enough to remove all the explicit inverse expressions in every occurrence of the `rnd` tactic in the EasyCrypt standard libraries and examples.

Simultaneous rnd rules Based on the fact that we can actually compute inverses for tuples, we developed an EasyCrypt tactic which allows to reduce the distance between cryptographic pen and paper proofs and EasyCrypt proofs. It appears that in this field of applications, the hypothesis of Proposition 9.4 about the number of outputs of the program equal to the number of its random variables is not a restriction.

As an example, let us consider the Cramer-Shoup encryption scheme of Figure 9.2. We consider a goal that appears in the EasyCrypt proof of the Cramer-Shoup encryption scheme, at the point where one has to apply the DDH assumption. We present in Figure 9.3 the pseudo-code of the goal, i.e., the two games and the (simplified) post-condition for these games. For readability, we

$$x, x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{F}_q;$$
$$sk := (g^x, x_1, x_2, y_1, y_2, z_1, z_2);$$
$$pk := (g^x, g^{x_1+xx_2}, g^{y_1+xy_2}, g^{z_1+xz_2});$$
$$\textsf{return}\ \ sk$$

Figure 9.2: Key Generation in Cramer-Shoup Encryption (abstracted and simplified)

Left game:

$x \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$
$y, z \xleftarrow{\$} \mathbb{F}_q$
$gx, gy, gz \leftarrow g^x, g^y, g^z$
$x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{F}_q$
$g_, a, a_ \leftarrow gx, gy, gz$
$k \xleftarrow{\$} dk$
$e \leftarrow g^{x1} * g_^{x2}$
$f \leftarrow g^{y1} * g_^{y2}$
$h \leftarrow g^{z1} * g_^{z2}$
$pk \leftarrow (k, g, g_, e, f, h)$
$sk \leftarrow (k, g, g_, x_1, x_2, y_1, y_2, z_1, z_2)$

Right game:

$w \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$
$u, x_0 \xleftarrow{\$} \mathbb{F}_q$
$g_ \leftarrow= g^w$
$k \xleftarrow{\$} dk$
$x, x_2 \xleftarrow{\$} \mathbb{F}_q$
$x_1, e \leftarrow x - w * x_2, g^x$
$y, y_2 \xleftarrow{\$} \mathbb{F}_q$
$y_1, f \leftarrow y - w * y_2, g^y$
$z, z_2 \xleftarrow{\$} \mathbb{F}_q$
$z_1 \leftarrow z - w * z_2$

Post-condition:

$(k, g, g_, x_1, x_2, y_1, y_2, z_1, z_2) \simeq (k, g, g_, x_1, x_2, y_1, y_2, z_1, z_2))$

Figure 9.3: (Abstracted) EasyCrypt Goal for Cramer-Shoup

omit the variable prefixes and suffixes used by EasyCrypt, and simply write w for $G1.w2$. We also omit some variables assignments that do not affect the post condition.

This goal amounts to prove that the secret keys provided by the actual game or the simulator are the same. We directly used our previous notations to capture this goal. If we expend some variable bindings, we obtain the following goal:

$$(k, g, g^x, x_1, x_2, y_1, y_2, z_1, z_2)$$
$$\simeq$$
$$(k, g, g^w, x - w * x_2, x_2, y - w * y_2, y_2, z - w * z_2, z_2)$$

We can then conclude by proving that the following map:

$$(k, g, x, x_1, x_2, y_1, y_2, z_1, z_2) \mapsto$$
$$(k, g, w, x - w * x_2, x_2, y - w * y_2, y_2, z - w * z_2, z_2)$$

is a bijection - for example, the inverse of $x_1 \mapsto x - w * x2$ being $r \mapsto r + w * x2$.

Currently, performing this part of the proof in EasyCrypt requires a mixture of code motion, code inlining and multiple applications of the `rnd` rules, as shown in Listing 9.2.

```
swap{1} 16 -9;  wp; swap -1; swap -1.
rnd (fun z ⇒  z + G1.w{2} * G1.z2{2})
    (fun z ⇒  z - G1.w{2} * G1.z2{2}).
rnd.
```

```
wp; swap -1.
rnd (fun z ⇒ z + G1.w{2} * G1.y2{2})
    (fun z ⇒ z - G1.w{2} * G1.y2{2}).
rnd.
wp; swap -1.
rnd (fun z ⇒ z + G1.w{2} * G1.x2{2})
    (fun z ⇒ z - G1.w{2} * G1.x2{2}).
rnd; wp; rnd; wp.
```

Listing 9.2: EasyCrypt Proof Script

Using our techniques, we were able to replace it with a single line tactic, using the new tactic `rndmatch`.

```
rndmatch (k, g, x, x_1, x_2, y_1, y_2, z_1, z_2)
  (G.k, G.g, G.w, G.x - G.w * G.x2, G.x2, G.y - G.w * G.y2, G.y2, G.z - G.w * G, z2, G.z2))
```

The underlying idea is that a user should only specify the variables on the left which they wish to map to expressions on the right. The tactic handles the necessary code motions and inlinings into the game until it produces a tuple at the end, after which the tactic automatically solves the equivalence using the enhanced `rnd` tactic:

```
rnd (fun (v1, v2, v3, v4, v5, v6, v7, v8, v9) ⇒
       (v1, v3, v3, v4 - v3 * v5, v5, v6 - v3 * v7, v7, v8 - v2 * v9, v9).
```

Part IV

A New Hope

In which we introduce the Squirrel *prover*

10 An Interactive Prover for Indistinguishability Proofs

> Girls are capable of doing everything men are capable of doing. Sometimes they have more imagination than men.
>
> *(Katherine Johnson)*

10.1 Introduction

We have shown how to simplify computational proofs by decomposing them into smaller proofs (Part II), and how to improve automation of low-level proof steps (Part III). Yet, high-level reasoning on protocols is still difficult in the computational model. To ease this kind of reasoning and provide a mechanized prover, we consider several points:

- It is often easier to use symbolic reasoning, based on simple logical deduction rules, rather than complex cryptographic reductions.
- We consider that backward search is often a reasoning more intuitive for users: to prove that a bad state may never be reached, we assume that we are in this bad state, look at what must have happened previously and derive from this a contradiction.
- Many properties do not require to look explicitly at all execution traces of the protocol: the security proof for an execution trace often subsumes the proof for many other traces.
- Indistinguishability proofs often require to prove that a bad state is never reached. This kind of proofs can be simplified if it is proved that the bad state cannot occur in a dedicated reachability prover, and if this reachability property is then leveraged in an indistinguishability prover to simplify the final proof.

The BC logic is a promising approach, as it allows to derive computational guarantees through purely symbolic reasoning. Furthermore, it is modular in term of axioms: to add a new primitive, it is only required to prove a new axiom independently from the others. It also allows for simpler reasoning by abstracting unnecessary details. In the symbolic model, reasoning about the xor implies to define complex equationnal theories and use heavy term rewriting techniques. In the BC logic, to add the xor operator, we can simply add a function symbol of arity two, without the full equational theory, and only add the axioms required for the security proofs. Typically, it holds that $n \oplus t$ is indistinguishable from n for any name n that does not occur in the term t. This axiom is often the only one required to prove the security of xor-based protocols.

Keeping in mind the previous points, we design, based on the BC logic, a meta-logic that allows for abstract reasoning about the traces. We provide the implementation of a tool, the Squirrel Prover (sources available at [Squ]), used to perform multiple case-studies, some among them relying on our composition framework. In this Thesis, we do not formally define the complete theory of the meta-logic, but rather try to give a flavour of the theory (the complete theory is also provided at [Squ]). We also highlight how the tool was easily adapted to support our composition result, and provide some examples in the user syntax of Squirrel.

Chapter Summary

In this Chapter, we develop a framework and an interactive prover allowing to perform security proofs at the symbolic level while obtaining guarantees at the computational level. To achieve this, we develop a meta-logic as well as a proof system for deriving security properties. We implement our approach within a new interactive prover, the Squirrel Prover, taking as input protocols specified in the applied pi-calculus.

We perform a number of case studies covering a variety of primitives (hashes, encryption, signatures, Diffie-Hellman exponentiation) and security properties (authentication, strong secrecy, unlinkability). By using both the Squirrel prover and the composition framework of Part II, we provide the first security proofs of real life protocols based on the BC logic that are both mechanized and valid for an unbounded number of sessions of a protocol.

10.1.1 Our Contributions

We use the BC logic as a building block to design a meta-logic that allows to reason about security properties by manipulating abstract traces and performing backward reasoning. Notably, it allows to reason by induction over the number of sessions of a protocol in a completely abstract way, and for instance prove indistinguishability by only considering abstract traces that terminate by each possible atomic action of the protocol. The meta-logic encompasses both a reachability and an indistinguishability sequent calculus, where reachability properties can be leveraged in indistinguishability proofs.

We implemented this approach in the Squirrel prover. The tool supports a variety of cryptographic primitives, among which hash, encryption, signature, Diffie-Hellman exponentiation and xor. It was used to prove authentication, strong secrecy and unlinkability for a variety of RFID protocols. Those case studies are performed for an arbitrary but fixed number of sessions, meaning that for each number of session there exists a security proofs, but the number of session does not depend on the security parameter.

We also perform the proof of SSH for an unbounded number of sessions, that can depend on the security parameter, by performing the sub-proofs obtained by the application of our composition framework (Part II). Remark that the tool was easily adapted to support the composition result, which argues in favour of our approach in Parts II and IV.

Limitations

We do not provide concrete security, where an explicit bound is given on the attacker's advantage. Also, we do not support proofs for an unbounded number of sessions without using the composition result.

10.1.2 Related Work

There exists a wide variety of provers that provide computational guarantees, most, if not all of them based on game-hopping techniques. We refer the interested reader to [BBB+19] for a detailed comparison of existing tools, and only highlight here two of the most successful ones.

EasyCrypt [BGH+11; BDG+13] is an interactive prover supporting game-hopping techniques through a probabilistic relational Hoare logic. It is a high-order logic that requires expertise

in program verification, while we consider a first-order logic dedicated to protocol verification. CryptoVerif [Bla07] is both an automated and interactive prover that provides proofs as game sequences. It can complete many proofs completely automatically, which is out of reach of our tool for the time being.

The various approaches can be compared on several criteria; we mention a few to highlight differences between our tool and existing ones.

Like CryptoVerif, our protocol specifications are given in the applied pi-calculus, which is well suited for this purpose. Yet, unlike CryptoVerif and EasyCrypt, we only provide asymptotic security bounds. However, our approach hides from the user all quantitative aspects such as probabilities and the security parameter and, on the surface, our tool is as simple as symbolic verification frameworks.

As we shall see, our proof methodology differs significantly from the game-hopping technique used e.g. in CryptoVerif. Game transformations, e.g. replacing all calls to some oracle by calls to another oracle, are *global* while our approach is more local: for example, our unforgeability rule states that if, at some instant T of the protocol, the attacker returns a hash of a message with a key that he does not know, then this hash must originate from an honest message sent at an instant $T' \leq T$. We argue that our local approach is often simpler, as it allows to reason about a protocol "message by message", and not as a single block.

Furthermore, we rely on a simple first-order logic formalism. This enable us to use techniques from proof theory, such as rewriting proof techniques or cut eliminations. It is also amenable to automatization by leveraging widely used first-order automatization, and a non trivial fragment of the BC logic has already been proven decidable [Kou19a].

Finally, rather than comparing to all other tools, we now discuss how we fit in the taxonomy of [BBB+19], that provides a detailed presentation of computer-aided cryptography. Their taxonomy captures all the tools that provide computational guarantees. They consider several criteria, divided in four categories: accuracy (A) of modeling/analysis, scope (S) of modeling/analysis, trust (T), and usability (U). The criterions and how we fit in them are as follows.

- Automation (U) - we do not provide standalone automation, but the reasoning about message equality is automated;
- Composition (U) - we support the composition framework of Part III;
- Concrete Security (A) - we only provide asymptotic security;
- Game hopping (U) - this category is difficult to consider in our case, as we are not per-say in the game-based model. Remark though that the logic supports some of the classical game hopping techniques;
- Unary Reasoning (U) - we support this kind of reasoning in the reachability prover;
- Link to implementation (T) - we do not provide any link with an implementation;
- Trusted computing base (T) - self code base, in OCaml;
- Specification Language (U) - a pi-calculus.

? Future Work

The BC logic, and thus our tool, could be extended to provide concrete security bounds. It would be interesting to improve the automation. For instance, the automated reasoning for equality over messages is currently handled by our own algorithms, while it could be integrated using SMT-based techniques. Finally, the BC logic and the tool could be extended to perform proofs by induction that are directly valid for an unbounded number of sessions, without going through the composition result.

10.2 Overview

Section Summary

We first give an overview of our framework and tool. The SQUIRREL prover and our case studies can be found in [Squ].

Let us consider a toy example, where an initiator I authenticates a Diffie-Hellman share to a responder R:

$$I \to R : \langle g^a, \mathsf{sign}(g^a, sk)\rangle$$

This protocol can be written in the SQUIRREL syntax as depicted in Listing 10.1.

```
signature sign,verify,pk

abstract ok : message
abstract error : message
name sk : message
name a : index→ message

channel c

process I(i:index) =
  out(c, ⟨ g^a[i], sign(g^a[i], sk)⟩)

process R(i:index) =
  in(c,x);
  if verify(snd(x), pk(sk)) = fst(x) then
     out(c,ok)
  else
     out(c,error)

system !_i R(i) | !_i I(i).
```

Listing 10.1: DH share authentication in SQUIRREL (user syntax)

The `signature` keyword is a built-in used to declare three functions which model a signature scheme that satisfies EUF-CMA[1]. `a` is a name indexed to instantiate a unique share for each session, $g^{a[i]}$ is used to denote Diffie-Hellman exponentiation, and `fst,snd` denote the first and second projection of the pair.

We now describe informally how to instantiate our framework to analyze this protocol. In our framework, we see protocols as set of actions, each action consisting of an execution condition and an output, which may depend on the input of the action and inputs of previous actions. The condition and output of an action A are terms, built notably over the *macro* input@A which is used to refer to the input given to the action by the attacker. In practice, our tool performs this instantiation automatically from the applied pi-calculus specification. The previous protocol is given by three actions.

- I[i] is the (unique) action performed by the i^{th} session of the initiator process I. Its execution condition is true, and its output is $\langle g^{a[i]}, \mathtt{sign}(g^{a[i]}, sk)\rangle$.
- R1[i] represents the action of a responder session i, the **then** branch of the process R, that received as input a pair whose second projection is a valid signature of its first projection. Its condition is
$$\mathsf{verify}(\mathsf{snd}(\mathsf{input@R1}[i]), \mathtt{pk}(sk)) = \mathsf{fst}(\mathsf{input@R1}[i])$$
and its output is the constant ok.

[1] Remark that we use here a variant of the signature scheme, where instead of a `checksign` function symbol, there is a verify function symbol such that $\mathsf{verify}(m, \mathtt{pk}(sk)) = m'$ is true if and only if m is equal to $\mathtt{sign}(m', sk)$.

- R2[i] represents the action of a responder session i, the **else** branch of the process R, when there is no valid signature. Its execution condition is the negation of the one of R1[i], and its output is the constant error.

A well authentication for the previous protocol is expressed in the user syntax Listing 10.2.

```
goal authentication :
forall (i:index), cond@R1[i] ⇒ (exists (j:index),  I(j) < R1(i)
                                                   && fst(input@R1(i)) = fst(output@I(j))
                                                   && snd(input@R1(i)) = snd(output@I(j))).
```

Listing 10.2: Simple property in SQUIRREL (user syntax)

Here cond@R1[i] is a *macro* which stands for the executability condition of action R1[i], where the responder checks that there is a valid signature.

Our authentication goal expresses that, whenever this condition holds, there must be some session j of the initiator that has been executed before R1[i]. Moreover, the output of the initiator's action coincides with the input of the responder's action.

This authentication goal can be proved in our tool using a succession of four *tactics*:

```
simpl. expand cond@R1(i). euf M0. exists i1.
```

Each tactic can be explained at a high level as follows:

- `simpl` introduce the variables i and the assumption cond@R1[i]. So that we can refer to this assumption, it is automatically given the label `M0`, as it is the first hypothesis over Messages.
- `expand cond@R1(i)` expands this macro into its meaning, i.e.

$$\mathsf{verify}(\mathsf{snd}(\mathsf{input@R1}[i]), \mathsf{pk}(sk)) = \mathsf{fst}(\mathsf{input@R}[i])$$

- `euf M0` applies the EUF-CMA assumption: the condition states that snd(input@R1[i]) is a valid signature of fst(input@R1[i]), thus the term fst(input@R1[i]) must be equal to a message that has previously been honestly signed. Therefore we deduce that there exists an initiator's session i_1 occurring before the action R1[i], such that its output was forwarded to R1[i].
- `exists i1` instantiates the existential quantification over j by i_1, which concludes the proof.

10.3 A Meta-Logic for Reachability and Equivalence

Section Summary

We design a meta-logic that contains terms referring to an execution of a protocol, and for instance refers to the value of the output performed by the protocol at some point in time. To this end, we see protocols as set actions (an action is given by an execution condition and an output) that can be triggered by an attacker to produce execution traces. Then, we introduce macros, which for an abstract trace allows to talk about the output, the input or the condition corresponding to a point in the trace.

We finally introduce two sequent calculi in this meta-logic, one for reachability and one for indistinguishability. BC axioms are naturally translated in the calculi, and we obtain the rules corresponding to a variety of cryptographic axioms.

10.3.1 The Meta-Logic

Our meta-logic is an extension of the BC logic, meaning that any formula and proof of the BC logic could be expressed in our meta-logic. The meta-logic does not increase the expressivity of

Timestamps

T	::=	τ	timestamp variable
	|	$\mathsf{a}[\vec{i}]$	indexed action name
	|	init	minimal element of a trace
	|	$\mathsf{pred}(T)$	predecessor

Meta-terms

t	::=	n	name
	|	$n[\vec{i}]$	indexed name
	|	x	term variable
	|	$f(t_1, \ldots, t_n)$	function of arity n
	|	$m@T$	macro with $m \in \{\mathsf{input}, \mathsf{output}, \mathsf{frame}\}$
	|	$\mathbf{if}\ \phi\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2$	conditional branching

Atoms

A	:=	$t = t'$	atomic proposition over messages
	|	$T = T' \mid T \leq T' \mid \mathsf{cond}@T \mid \mathsf{exec}@T$	atomic proposition over timestamps
	|	$i = i'$	atomic proposition over indices

Meta-formulas

$$\phi \quad ::= \quad A \mid \mathsf{true} \mid \mathsf{false} \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \phi \Rightarrow \phi' \mid \neg\phi \mid \forall i.\phi \mid \exists i.\phi \mid \forall \tau.\phi \mid \exists \tau.\phi$$

Figure 10.1: Syntax of the Meta-Logic

the BC logic: we cannot for instance derive computational guarantees that the BC logic could not. Rather, our framework formalizes an abstract reasoning about the number of sessions or the interleavings of a protocol. Notably, it is possible to derive a proof in the BC logic for any number of sessions of a protocol from a single proof in the meta-logic.

Syntax We extend the terms so that they can refer to protocol executions. Recall that we used terms to represent the bitstrings manipulated and communicated by the protocols, and used index variables to instantiate multiple copies of the same name. We now build a meta-logic over

- *timestamps*, that represent time points in an execution trace of a protocol;
- *meta-terms*, terms extended with macros;
- *meta-formulas* to express conditions over the meta-terms.

As meta-terms contain a conditional branching over meta-formulas, meta-terms and meta-formulas are mutually inductive. We still use indices to instantiate multiple copies. The complete syntax is given in Figure 10.1.

To refer to point in a trace, we use timestamps. We consider that we have a set of action names $\mathcal{A}$, that each refer to a possible atomic action of a protocol, and each action name can also be indexed. Timestamps are then either an explicit action, which will refer to the time point at which this action occurs, a timestamp variable, or the init constant, which is used to refer to the first time point of a trace. We also define a predecessor function pred over the timestamps, to refer to the previous action in a trace.

With those timestamps, denoted by T, we are then able to refer to elements of the actions performed at some point. The message macros $\mathsf{input}@T$ and $\mathsf{output}@T$ refer to the input and output messages of the action executed at time T and the boolean macro $\mathsf{cond}@T$ encodes the execution condition of the action at T. Finally, the boolean macro $\mathsf{exec}@T$ encodes the conjunction of all execution conditions up until time T, $\mathsf{exec}@T$ intuitively corresponds to $\mathsf{exec}@\mathsf{pred}(T) \wedge \mathsf{cond}@T$, where $\mathsf{exec}@\mathsf{init} = \mathsf{true}$.

The message macro $\mathsf{frame}@T$ refers to the attacker's knowledge at time T, where they can learn if an action is executable, and if it is indeed executable, they learn the corresponding output. Then, by reusing the ternary function symbol **if** _ **then** _ **else** _ (not to be confused with the construct of the protocols) to encode conditionals in terms, we may see $\mathsf{frame}@T$ as the triple $\langle \mathsf{frame}@\mathsf{pred}(T), \mathsf{exec}@T, \textbf{if } \mathsf{exec}@T \textbf{ then } \mathsf{output}@T \textbf{ else } \mathbf{0}\rangle$, where $\mathbf{0}$ is a default value that we set for the **else** branch. By defining the frame in this way, for each action triggered by the attacker, they get the value of the condition and the value of the output if the action can be executed, else they get the value $\mathbf{0}$.

The meta-formulas are quantified over variables of type index and timestamp, and contain basic logical combinations. This syntax allows to express a variety of first-order formulas for reachability properties.

Example 10.1. Let us assume that we have two action names $\mathsf{a}[i]$ and $\mathsf{b}[i]$. The following formula is used to specify that whenever the condition of an $\mathsf{a}[i]$ is true, it means that some $\mathsf{b}[j]$ occurred previously, and the output produced by $\mathsf{b}[j]$ was forwarded as input to $\mathsf{a}[i]$:

$$\forall i : \mathsf{index}.\ \mathsf{cond}@\mathsf{a}[i] \Leftrightarrow \exists j : \mathsf{index}.\ \mathsf{b}[j] < \mathsf{a}[i] \wedge \mathsf{input}@\mathsf{a}[i] = \mathsf{output}@\mathsf{b}[j]$$

This formula typically express a (non injective) well-authentication property, and could be verified by a protocol that uses a signature.

The meta-logic will allow us to prove that a meta formula is true with overwhelming probability for all possible execution traces of a protocol. Remark that if the meta-formula of Example 10.1, holds, it means that the two meta-formulas that are equivalent have exactly the same probability distributions for all execution traces of a protocol. One could then replace one by another without changing the protocol.

The meta-logic will also support boolean formulas built over indistinguishability atoms of the form $\overline{u} \sim \overline{v}$ for sequences of meta-terms or meta-formulas, where macros on the two sides will be interpreted with two distinct protocols. Intuitively, the formula $\mathsf{frame}@\tau \sim \mathsf{frame}@\tau$ will be true if two protocols, the one used to interpret the frame on the left and the one for the frame on the right, are in fact indistinguishable.

Technical Details

This notion of frame is meant to capture the sequence of message ϕ_P^τ of Definition 2.14, and thus match our previous notion of equivalence. Remark though that to exactly match the definition of ϕ_P^τ, we would have to define $\mathsf{frame}@T$ as $\langle \mathsf{frame}@\mathsf{pred}(T), \textbf{if } \mathsf{exec}@T \textbf{ then } \mathsf{output}@T \textbf{ else } \mathbf{0}\rangle$. Here, we add $\mathsf{exec}@T$ to the frame, because we want the attacker to know if an action can be executed or not.

Let us consider a simple example, to illustrate the issue. We simply consider two protocols,

- $P = \mathbf{in}(c, x).\mathbf{if}\ x <> \mathbf{0}\ \mathbf{then}\ \mathbf{out}(c, \mathbf{0})$
- $Q = \mathbf{in}(c, x).\mathbf{if}\ \mathsf{true}\ \mathbf{then}\ \mathbf{out}(c, \mathbf{0})$

Intuitively, we expect those two protocols to be distinguishable, as the action of P is not always executed, depending on the input, while the one of Q is always executed. If we define the frame

without exec, then for instance for P (where we denote its only action P), there are two possible frames:

- the frame with the constant $\mathbf{0}$, corresponding to the beginning of the protocol, before the attacker gave any input to P;
- the frame obtained after the attacker input, which is equal to

$$(\mathbf{if}\ \mathsf{exec}@P\ \mathbf{then}\ \mathsf{output}@P\ \mathbf{else}\ \mathbf{0})$$

As $\mathsf{output}@P = \mathbf{0}$, this last frame is actually equal to $\mathbf{0}$. The behaviour is similar for Q, and thus, if we do not put the execution condition in the frame, P and Q are indistinguishable. If we add the condition, the frame of P will contain the value of $\mathsf{input}@P <> \mathbf{0}$, while the frame of Q will contain true, and an attacker can then distinguish those two values.

Remark that this issue stems from the fact that we build the frame using $\mathbf{0}$ in the else branch, while $\mathbf{0}$ can also be produced by the protocol.

This issue was not raised in Definition 2.14 because we only considered simple protocols, where the conditional branchings appearing in ϕ_P^τ are part of the protocol specification (the user must then be careful about the modeling). In the case of the tool, as the conditional are built by the tool, we wanted to provide a precise notion of indistinguishability based on the user input.

Actions and Protocols An action is given by $\mathsf{a}[i_1, \ldots, i_k].(o, \phi)$, where a is the action name, $i_1, \ldots, i_k$ a sequence of index variables, o a meta-term corresponding to the output of the action and ϕ its executability condition.

A protocols is then a couple $(P_A, \leq_P)$, where P_A is a set of actions and $\leq_P$ a partial order $\leq_P$ over P_A (a protocol is then a poset). The partial orders indicates which actions must be executed before other ones. An action may in its condition and output refer to the inputs of previous actions (w.r.t. $\leq_P$)

Example 10.2. For illustration purposes, we consider the following protocol P made of two actions

1. $\mathsf{a}[i].(\mathsf{true}, \mathsf{ok})$; and
2. $\mathsf{b}[i].(\mathsf{true}, \langle \mathsf{input}@\mathsf{a}[i], \mathsf{input}@\mathsf{b}[i] \rangle)$
3. $\mathsf{c}[j].(\mathsf{input}@\mathsf{c}[j] = \mathsf{ok}, \mathsf{ok})$

with $\mathsf{a}[i] \leq_P \mathsf{b}[i]$. Intuitively, this corresponds to a protocol that first inputs a message, emits ok; and then inputs another message before outputting the pair of the two messages it has received. Because of the ordering, the condition and the output of $\mathsf{c}[j]$ is not allowed to depend on a macro of a or b.

In the pi-calculus of Chapter 2, this protocol would be written as

$$\|^i(\mathbf{in}(c, x); \mathbf{out}(x, \mathsf{ok}); \mathbf{in}(c, y); \mathbf{out}(c, \langle x, y \rangle)) \|^j (\mathbf{in}(d, x); \mathbf{if}\ x = \mathsf{ok}\ \mathbf{then}\ \mathbf{out}(d, \mathsf{ok}))$$

We may note that $\mathsf{b}(i)$ is allowed to refer to $\mathsf{input}@\mathsf{a}[i]$ since we have specified that $\mathsf{a}[i] \leq_P \mathsf{b}[i]$.

A trace of a protocol P is any sequence of actions $\mathsf{a}[l_1, \ldots, l_k] \in P_A$, where the index variables have been instantiated by concrete values over the integers, which is valid w.r.t. the ordering $\leq_P$ and where an instance of an action does not occur twice.

Example 10.3. With the protocol of Example 10.2, for a single session of i numbered by index 1, and no session for j, there are two possible traces: the trace with a single action $\mathsf{a}[1]$, and the trace with the two actions $\mathsf{a}[1].\mathsf{b}[1]$. As soon as we consider multiple sessions, many interleavings become possible, as long as they comply with the ordering.

Remark that in this Part, we have changed the definition of protocols, in order to build a logic over them. The translation is straightforward and implemented in the tool. We do not provide its details here for concision.

Semantics Given a protocol P, we now define the interpretation of meta-terms and meta-formulas. Intuitively, in addition to the computational models $\mathcal{M}$ of the BC logic (Section 2.4), models of our meta-logic are also built over a trace model $\mathcal{T}$. It consists of a (finite) trace of the protocol, and thus provides an interpretation domain $\mathcal{D}_\mathcal{T}$ for the timestamp variables, which is the set of actions appearing in the trace, and an interpretation domain $\mathcal{D}_\mathcal{I}$ for the index variables, which is the set of indexes appearing in the actions of the trace.

Given a protocol P and one of its trace models $\mathcal{T}$, we can interpret all meta-terms and meta-formulas as classical terms of the BC logic (recall that we have in the terms of the BC logic boolean connectives $\dot{\wedge}, \dot{\vee}, \ldots$ as function symbols with a fixed interpretation). We denote this interpretation by $(\phi)^\mathcal{T}_P$, but only provide here an idea of the translation, where for instance:

$$(\forall \tau : \mathsf{timestamp}.\ \phi)^\mathcal{T}_P := \dot{\bigwedge}_{T \in \mathcal{D}_\mathcal{T}} (\phi)^{\mathcal{T}\{\tau \mapsto T\}}_P$$

Intuitively, ϕ holds for all timestamps of the trace model $\mathcal{T}$ if the conjunction of the interpretations of ϕ where $\mathcal{T}$ interprets τ by all possible values is true. The translation is similar for other quantifiers, ($\exists$ translates to $\bigvee$), and other Boolean operations are translated with the corresponding connective. Input macros are interpreted using a dedicated attacker function symbol g from the BC logic:

$$(\mathsf{input@}T)^\mathcal{T}_P := g((\mathsf{frame@pred}(T))^\mathcal{T}_P)$$

Other macros simply correspond to the interpretation of the meta-term corresponding to the action at the given timestamp.

Given a protocol P, a computational model $\mathcal{M}$ and a trace model $\mathcal{T}$, we say that ϕ holds w.r.t. $\mathcal{M}, \mathcal{T}$, denoted by $\mathcal{M}, \mathcal{T} \models_P \phi$, if

$$\mathcal{M} \models (\phi)^\mathcal{T}_P \sim \mathsf{true}$$

in the BC logic. A formula is P-valid if it holds for all computational models and all trace models of P. If a formula is P-valid, it means that it is true with overwhelming probability on all traces of P.

We also define the semantics of indistinguishability formulas. Given two protocols P_1, P_2 that have the same set of trace models, a computational model $\mathcal{M}$ and a trace model $\mathcal{T}$ of the protocols, and two sequences of meta-terms $\overline{u}, \overline{v}$, we say that $\overline{u} \sim \overline{v}$ holds w.r.t. $\mathcal{M}, \mathcal{T}$, denoted by $\mathcal{M}, \mathcal{T} \models_{P_1,P_2} \overline{u} \sim \overline{v}$, if

$$\mathcal{M} \models (\overline{u})^\mathcal{T}_{P_1} \sim (\overline{v})^\mathcal{T}_{P_2}$$

$\overline{u} \sim \overline{v}$ is then (P_1, P_2)-valid if it holds for all trace models of P_1, P_2.

Example 10.4. If $\mathsf{frame@}\tau \sim \mathsf{frame@}\tau$ is (P_1, P_2) valid, it implies that for all their traces (the two protocols must have the same set of traces, but can differ in their common actions), and for all points in the trace, the sequence of messages up to this point produced by P_1 is indistinguishable to the one of P_2. In other terms, it implies that P_1 and P_2 are computationally indistinguishable.

Technical Details

Expecting that the two protocols have the same set of traces can be a restriction to verify indistinguishability. Indeed, consider the two following protocols, where bool is a function which extracts the first bit of a bitstring:

- $\mathbf{in}(c, x).\mathbf{if}\ \mathsf{bool}(x)\ \mathbf{then}\ \mathbf{out}(c, n_1)\ \mathbf{else}\ \mathbf{out}(c, n_2)$
- $\mathbf{in}(c, x).\mathbf{out}(c, n_1)$

Those two protocols are indistinguishable, as both of them always return a fresh random name, independently from their input. Yet, with our translation, the first one is expressed with two actions, and the second one with a single action: they cannot then be indistinguishable in our framework. However, and in the spirit of simple protocols of Section 2.4, the first protocol could be rewritten by pushing the conditional in the output:

$$\mathbf{in}(c, x).\mathbf{out}(c, \mathbf{if}\ \mathsf{bool}(x)\ \mathbf{then}\ n_1\ \mathbf{else}\ n_2)$$

Then, in our framework, the two protocols do become indistinguishable.

10.3.2 Reachability Rules

With a fixed protocol P, we now provide a sequent calculus that allows to prove the validity of formulas.

Definition 10.1. Let P be a protocol. A sequent $\Gamma \vdash_P \phi$ is composed of a set of meta-formulas Γ and a meta-formula ϕ.

The sequent $\Gamma \vdash_P \phi$ is valid if and only if $\Gamma \Rightarrow \phi$ is P-valid.

Basic rules For concision, we do not provide all the rules of the sequent calculus. We notably designed the calculus so that all the rules of the standard first-order sequent calculus are valid. We provide in Figure 10.2 some of the rules dedicated to the security proofs of protocol.

The first three rules allow to conclude by deriving a contradiction from the set of hypothesis. With overwhelming probability, two names are not equal. Thus, if in a sequent we have as hypothesis that two distinct names are equal, we can derive false with rule NameIndep. A variant of the rule would specify that if $\mathsf{n}[\vec{i}] = \mathsf{n}[\vec{j}]$, then we can conclude that $\vec{i} = \vec{j}$.

If we have as hypothesis a trace constraint $b[\vec{i}] \leq a[\vec{j}]$ that contradicts the ordering $\leq_P$ of the protocol, we can conclude with ActDep. ActEq states that two distinct actions cannot occur at the same timestamp. Finally, Exec states that if $\mathsf{exec}@\tau$ is an hypothesis for some timestamp, we have the condition execution of all the smaller timestamps.

These four rules are sound, which means that they can safely be used to derive true statements. All the rules of our sequent calculus, along with their soundness proofs, are available in the technical report [Squ].

Advanced rules The rules presented previously are unconditionally sound: they hold without any computational hardness assumption. For each computational assumption, we can derive a dedicated rule. Let us recall the axiom scheme EUF-CMA$_{sk}$ (Definition 2.17) which, for any term

$$\textsc{NameIndep}\ \frac{\mathsf{n} \neq \mathsf{m}}{\Gamma, \mathsf{n}[\vec{i}] = \mathsf{m}[\vec{j}] \vdash_P \phi} \qquad \textsc{ActDep}\ \frac{a[\vec{j}] <_P b[\vec{i}]}{\Gamma, b[\vec{i}] \leq a[\vec{j}] \vdash_P \phi} \qquad \textsc{ActEq}\ \frac{a \neq b}{\Gamma, a[\vec{i}] = b[\vec{j}] \vdash_P \phi} \qquad \textsc{Exec}\ \frac{\Gamma, \forall \tau' \leq \tau.\mathsf{cond}@\tau' \vdash_P \phi}{\Gamma, \mathsf{exec}@\tau \vdash_P \phi}$$

Figure 10.2: Some Rules of our Sequent Calculus for Reachability

t such that sk is only in key position, corresponds to:

$$\begin{array}{l}\textbf{if } (\mathtt{checksign}(t, \mathtt{pk}(sk))) \textbf{ then} \\ \quad \dot{\bigvee}_{\mathtt{sign}(x,sk)\in \mathtt{St}(t)} (t \doteq \mathtt{sign}(x, sk)) \quad \sim \top \\ \textbf{else } \top\end{array}$$

This naturally translates into a rule of the form:

$$\frac{\Gamma, \bigvee_{\mathtt{sign}(x,sk)\in \mathsf{S}_{sk(t)}} t = \mathtt{sign}(x, sk) \vdash_P \phi}{\Gamma, \mathtt{checksign}(t, \mathtt{pk}(sk)) = \mathsf{true} \vdash_P \phi} \text{ when } \mathsf{SSC}_{sk}(t)$$

Where we must define $\mathsf{SSC}_{sk}(t)$ and $\mathsf{S}_{sk}(t)$ such that,

- if the Syntactic Side-Condition $\mathsf{SSC}_{sk}(t)$ holds, then all translations of t satisfy the side-condition of the BC axiom, i.e., sk only appears in key position;
- the translation in any trace model of the Set of meta-terms $\mathsf{S}_{sk}(t)$ contains all signatures over sk of the translation of t.

While we do not provide the cumbersome details, it is straightforward to define $\mathsf{SSC}_{sk}(t)$ and $\mathsf{S}_{sk}(t)$ by iterating over the terms and the possible values of the macros in a term, by exploring the possible corresponding actions. For instance, if t contains a macro $\mathsf{input}@\tau$, we must check that all possible outputs of the protocol satisfy the side condition, as they might have occurred before τ and be used by the attacker to build a new term that could contradict the syntactic side condition.

Our sequent calculus also contains rules derived, for instance from the collision resistance of a hash function, or from the EUF-CMA axiom corresponding to an hash function.

Extension for the composition result As our composition result only relies on new axioms for the BC logic, we can easily integrate them in our sequent calculus by deriving a rule from a new axiom. Recall that the EUF-CMA$_{T,sk}$ axiom (Definition 6.4), is expressed for any boolean function T and term t where sk is only used in key position, as:

$$\begin{array}{l}\textbf{if } (\mathtt{checksign}(t, \mathtt{pk}(sk))) \\ \quad \textbf{then } T(\mathtt{getmess}(t)) \\ \qquad \dot{\bigvee}_{\mathtt{sign}(x,sk)\in \mathtt{St}(t)} (t \doteq \mathtt{sign}(x, sk)) \quad \sim \top \\ \quad \textbf{else } \top\end{array}$$

As EUF-CMA$_{T,sk}$ is very close to the classical EUF-CMA axiom, it is adapted similarly in our sequent calculus, with a rule of the form:

$$\frac{\Gamma, \bigvee_{\mathtt{sign}(x,sk)\in \mathsf{S}_{sk(t)}} t = \mathtt{sign}(x, sk) \vee T(\mathtt{getmess}(t)) \vdash_P \phi}{\Gamma, \mathtt{checksign}(t, \mathtt{pk}(sk)) = \mathsf{true} \vdash_P \phi} \text{ when } \mathsf{SSC}_{sk}(t)$$

It is interesting to note that this is the only extension required to our tool in order to be able to perform proofs based on our composition framework Part II.

10.3.3 Indistinguishability Rules

We now define a sequent calculus dedicated to proving indistinguishability properties.

Definition 10.2. Let P_1, P_2 be protocols with the same set of traces. An equivalence sequent $\Gamma \vdash_{P_1,P_2} \vec{u} \sim \vec{v}$ comprises a set of hypotheses Γ and a goal $\vec{u} \sim \vec{v}$, where Γ and $\vec{u} \sim \vec{v}$ are all equivalence formulas of the meta-logic.

The sequent $\Gamma \vdash_{P_1,P_2} \vec{u} \sim \vec{v}$ is valid if for all $\mathcal{T}$, $\mathcal{M}$, σ such that $\mathcal{T}, \mathcal{M}, \sigma \models_{P_1,P_2} \psi$ for all $\psi \in \Gamma$, we have that $\mathcal{T}, \mathcal{M}, \sigma \models_{P_1,P_2} \vec{u} \sim \vec{v}$.

The main technique used to reason in this calculus is to prove indistinguishability by induction, as for two protocols P_1, P_2 with the same set of traces, they are indistinguishable, in the sense that $\mathsf{frame@}\tau \sim \mathsf{frame@}\tau$ is (P_1, P_2)-valid, if and only if

$$\mathsf{frame@pred}(\tau) \sim \mathsf{frame@pred}(\tau) \vdash_{P_1,P_2} \mathsf{frame@}\tau \sim \mathsf{frame@}\tau$$

Remark that we can omit the base case, as it is always trivial ($\mathsf{frame@init} = \mathbf{0}$).

To reason about such sequents, once again, all classical sequent calculus rules apply, and we do not present them. Other rules are instantaneous consequences of the properties of $\sim$, based on transitivity, reflexivity and symmetry:

$$\textsc{Sym}\ \frac{\Gamma \vdash_{P_2,P_1} \vec{u} \sim \vec{v}}{\Gamma \vdash_{P_1,P_2} \vec{v} \sim \vec{u}} \qquad \textsc{Refl}\ \frac{}{\Gamma \vdash_{P_1,P_2} \vec{t} \sim \vec{t}}\ \vec{t} \text{ is macro-free}$$

$$\textsc{Trans}\ \frac{\Gamma \vdash_{P_1,P_2} \vec{u} \sim \vec{t} \qquad \Gamma \vdash_{P_2,P_3} \vec{t} \sim \vec{v}}{\Gamma \vdash_{P_1,P_3} \vec{u} \sim \vec{v}}\ P_1, P_2, P_3 \text{ have the same set of traces}$$

Some of the most powerful rules are the ones that combine both sequent calculi:

$$\textsc{Equiv}\ \frac{\Gamma \vdash_{P_1} t_1 = t_2 \quad \Gamma \vdash_{P_2} t_1 = t_2 \quad \Gamma \vdash_{P_1,P_2} \vec{u}[t_1/t_2] \sim \vec{v}[t_1/t_2]}{\Gamma \vdash_{P_1,P_2} \vec{u} \sim \vec{v}}$$

$$\textsc{Equiv'}\ \frac{\phi \vdash_{P_1} \phi \Leftrightarrow \psi \quad \phi \vdash_{P_2} \phi \Leftrightarrow \psi \quad \Gamma \vdash_{P_1,P_2} \vec{u}[\phi/\psi] \sim \vec{v}[\phi/\psi]}{\Gamma \vdash_{P_1,P_2} \vec{u} \sim \vec{v}} \qquad \textsc{If-reach}\ \frac{\phi \vdash_{P_1} \mathsf{false} \quad \phi \vdash_{P_2} \mathsf{false}}{\Gamma \vdash_{P_1,P_2} \mathbf{if}\ \phi\ \mathbf{then}\ u \sim \mathbf{if}\ \phi\ \mathbf{then}\ v}$$

The rule Equiv allows to replace in an indistinguishability goal a term by another one, as long as we can prove in the reachability calculus that the two terms are equal with overwhelming probability. Equiv' is the corresponding rule but for formulas. Equiv' can typically be used to

replace an execution condition of an action by the well-authentication property that is equivalent to it (cf. Example 10.1).

Finally, more advanced rules can be defined by adapting the rules from the BC logic. Our sequent sequent calculus notably contains rules for encryption from the IND-CCA$_1$ and ENC$_{KP}$ axioms (adapted from [BC14a]), rules for hash function with the PRF axiom and for xor operations ([Kou19c]) and finally for DDH ([BCE+19]).

10.4 Implementation and Case-Studies

Section Summary

We implemented the meta-logic in a tool, the SQUIRREL prover, available at [Squ]. The tool is a computer-aided protocol prover. Its input language is a dialect of the applied pi-calculus, as depicted in Listing 10.1, and allows to prove formulas of the meta-logic for a specified protocol. We used it to perform a number of case studies, proving different kind of properties (authentication, secrecy, anonymity, unlinkability) under multiple cryptographic assumptions (IND-CCA, EUF-CMA, PRF, DDH), for a variety of protocols (RFID based, SSH, private authentication). We only outline here some details about its usage, and the integration of the composition result.

10.4.1 The Tool

The core of the prover is an implementation of the two sequent calculi for reachability and indistinguishability. Protocols are specified in a pi-calculus and a straight-forward algorithm allows to obtain from this specification the list of corresponding actions.

Once the protocol is specified, the user has to write a reachability or indistinguishability goal expressing the desired security property. This goal is the starting step from which rules of our sequent calculus (tactics in the tool) will successively be applied until completing the proof. Thus, proving a security property for a protocol using our tool consists in writing the script describing the order in which tactics are applied.

A simplification tactic, which is the result of a combination of tactics, is applied at each step of the proofs. The reasoning over equalities and disequalities of terms is automated, allowing to automatically derive contradictions.

The tool supports user-defined axioms, and previous proofs can be re-used. Notably, the reachability calculus can be used in an indistinguishability proof. For instance, if we prove that the condition of a conditional branching is always false, we can remove the corresponding branch. From our experience, these interactions between the two calculi ease the proof process.

All axioms and assumptions used in a proof must be explicitly provided. For instance, it may be necessary to specify that a pair cannot be confused with a name. Our tool then allows to derive the exact assumptions that the implementation should verify. Only axioms that are part of the first order logic formula without $\sim$ can be defined in the user syntax.

A strength of the tool is that it is symbolic and still computationally sound. In particular, to support the XOR theory, which is challenging in the symbolic model, we simply add a tactic that captures the cryptographic assumptions of the XOR function which are necessary to prove the security of protocols. Notably, there is no need to provide a full support of the XOR equational

Protocol	LoC	Assumptions	Security properties
Basic Hash [BCH10]	100	PRF, EUF-CMA	Authentication & Unlinkability
Hash Lock [JW09]	130	PRF, EUF-CMA	Authentication & Unlinkability
LAK (with pairs) [HBD19]	250	PRF, EUF-CMA	Authentication & Unlinkability
MW [MW04]	300	PRF, EUF-CMA, XOR	Authentication & Unlinkability
Feldhofer [FDW04]	250	IND-CCA$_1$, EUF-CMA	Authentication & Unlinkability
Private Authentication [BC14a]	60	IND-CCA$_1$, EUF-CMA, ENC-KP	Anonymity
Signed DDH [Iso, ISO 9798-3]	150	EUF-CMA, DDH	Authentication & Strong Secrecy
Additional case studies, using the composition framework from Part II			
Signed DDH [Iso, ISO 9798-3]	200	EUF-CMA, DDH	Authentication & Strong Secrecy
SSH (with forwarding agent) [YL]	750	EUF-CMA, DDH	Authentication & Strong Secrecy

Table 10.1: Case Studies

theory, or to be able to perform equational unification. Another strength of the tool is its modularity: extending the tool with new cryptographic primitives does not impact the core of the tool. It only requires to add new tactics and prove their soundness. Currently, the tool supports PRF,EUF-CMA (both for signatures and hashes), IND-CCA$_1$,ENC-KP,DDH and XOR.

10.4.2 Case-Studies

To illustrate the variety of examples, all the case studies that can be found in the Squirrel repository [Squ] are summarized in Table 10.1.

For each protocol, we provide the number of Lines of Codes (LoC) required for the specification and the proofs, the cryptographic assumptions used, and the security properties studied. Interestingly, most proofs follow the intuition of the pen-and-paper proofs, while some low-level reasoning is successfully abstracted away or automated. To our knowledge, those protocols have not been previously proved secure using a computationally sound mechanized prover.

We only discuss here the application to the DDH based protocols, that were already discussed in Part II. The other case studies are presented in more details in the repository [Squ].

DDH based protocols We first performed a proof of strong secrecy of the shared key for the signed DDH protocol [Iso, ISO 9798-3]. This proof was performed for an arbitrary but fixed number of sessions, where the number of sessions does not depend on the security parameter. We first performed a proof of the authentication property, which is a direct consequence of the EUF-CMA axiom. Then, when we look at the strong-secrecy of the key, where in the honest case we can easily use the DDH axiom to conclude, and we can prove that the event that leaks the key cannot happen thanks to the authentication property.

We also present two additional case studies for the signed DDH protocol [Iso, ISO 9798-3] and the SSH protocol [YL], where proofs are performed through the use of the composition framework of Part II. We outlined how to decompose a proof for those protocols into single session proofs, which consist in slightly modifying the hash function by giving more capacities to the attacker by providing them access to oracles.

For instance, we allow to define a signature that follows the EUF-CMA$_{T1,sk1}$ axiom, the EUF-CMA$_{T2,sk2}$ axiom for two specific keys $sk1, sk2$ and the EUF-CMA$_{sk}$ axiom for all other keys, with the syntax described in Listing 10.3.

```
signature sign,checksign,pk with oracle
   forall (m:message,k:message)
          (k = sk1 ⇒ T1(m)) && (k = sk2 ⇒ T2(m))
```

Listing 10.3: Declaration of a signature following the EUF-CMA$_{T,sk}$ axiom

The tagged EUF-CMA axioms defined in Section 5.7 to prove the security of the ISO 9798-3 protocol can then be declared in SQUIRREL as:

```
signature sign,checksign,pk with oracle
     forall (m:message,k:message)
       (k = skI ⇒  exists (i:index, x1:message, x2:message) m=<x1,g^a[i]⟩,x2⟩)
          &&
       (k = skR ⇒  exists (i:index, x1:message, x2:message) m=<x1,g^b[i]⟩,x2⟩)
```

Listing 10.4: Declaration of the signature for the ISO 9798-3 protocol

With our tool and the previous axioms, we were able to mechanize those proofs. Compared to the other case studies, those two hold for an unbounded number of sessions that may depend on the security parameter, and not just for an arbitrary but fixed number of sessions. Those are the first security proofs through the BC logic that are mechanized and valid for an unbounded number of sessions.

Conclusion and Future Work

Un dernier calcul et on s'en va.

(Proverbe ancestral)

In this work, by leveraging symbolic methods in various ways, we strived to help derive formal guarantees about protocols while considering attackers as powerful as possible. To this end, we

1. defined a methodology for the extensive analysis in the symbolic model of Multi-Factor authentication protocols, a complex application scenario, and we used the Proverif tool to carry out a case study over widely deployed protocols;
2. developed a composition framework both for the computational and the BC logic; it notably adds to the BC logic the support for security proofs of protocols with an unbounded number of sessions;
3. studied the automation of low-level proof steps about probabilistic distributions taking a foundational perspective, providing decidability and complexity results, and showing how to use symbolic methods to derive efficient heuristics;
4. implemented an interactive prover built over the BC logic, supporting both reachability and indistinguishability properties.

We believe that after pushing the symbolic model to its limits in (1), considering over 6000 scenarios to illustrate how the symbolic model can handle powerful attackers, we have made it easier in (2,3,4) to derive computational guarantees, stronger than the symbolic ones, by reasoning in a logical framework. We helped perform such proofs by simplifying both the probabilistic (3) and the logical reasoning (2,4) about protocols. In doing so, some important limitations of the BC logic were lifted, first by providing a composition framework (2) that reduces the size of the proofs and allow for proofs valid for an unbounded number of sessions, and second by building over it an interactive prover (4) that allows to reason in an abstract way about all the possible executions of a protocol.

Future Work

As we already gave for each Chapter some dedicated future work, we only consider here a high level point of view. We have made progress regarding symbolic analysis of protocols with computational guarantees, trying to consider attackers as strong as possible. Our long term goal is to be able to perform such analysis for:

1. more complex protocols, e.g., e-voting protocols, or advanced cryptographic primitives, e.g., Multi-Party Computation;
2. more complex security properties, e.g., post-compromise security or forward secrecy for key exchange protocols, or privacy-related properties such as unlinkability.

Modular The framework that we designed is general. We applied it to the specific case of key exchanges, but it may be applied in a variety of other contexts, and extended if need be. It would be interesting to see if our framework can be used in the case where it is not the number of sessions but the number of parties that can be unbounded. This is notably the case for e-voting protocols.

Further, the framework could be leveraged to simplify the proof of a variety of other properties, such as the ones mentioned previously.

Automation The Squirrel prover only provides a very basic level of automation. While it is already sufficient to perform real life case-studies, it may become capital to improve its automation to tackle more complex case studies. As mentioned previously, proof steps can be either seen as high-level reasoning, the logical level where probabilities have been abstracted, or as low-level reasoning about message lengths or probability distributions. The techniques required for automating those two levels are very different:

1. *high-level reasoning* - [Kou19a] provides a decidability result for a fragment of the BC logic. Building on this and classical first-order logic automation techniques, it would be interesting to try to automate the application of the logical rules of our tool.
2. *low-level reasoning* - as for instance advanced cryptographic primitives often rely on probabilistic reasoning, it could be interesting to integrate our results from Part III in the Squirrel prover.

Collaboration between tools We believe that the Squirrel prover bears great promises for the analysis of security protocols. However, we do not claim that our tool is or will ever be the best to handle everything. Notably, we cannot hope to achieve the level of efficiency or automation of symbolic tools, such as Proverif [BCA+10], Tamarin [MSC+13] or DeepSec [CKR18b]. Further, we do not aim to compete with EasyCrypt [BGH+11; BDG+13] when it comes to the analysis of cryptographic primitives, nor will we compete with MaskVerif [BBD+15] for the automatic verification of non interference properties.

Those are only some examples among many others. Each tool has its particular strengths and limitations, and each tool should be used for what it does best. In this spirit, we argue that it is important to build bridges between the tools. Notably, we aim to develop a framework that would enable from a single input file, to export protocol models for multiple tools, then using distinct tools for distinct proofs of security over the same protocol model. It would also be interesting to develop stronger interactions, where in the Squirrel prover, it would be possible to discharge a proof goal to EasyCrypt, or even to a symbolic tool when it is to verify a positive reachability property, e.g., an attack or an executability witness.

Finally, the variety of tools highlights the need for a foundational approach to security. Problems should not be studied with a single tool in mind: they should be studied from a general perspective, so that a single result can be used to improve multiple tools. We have followed this idea in Part II, with a composition framework that is used in Squirrel but could also be used in EasyCrypt. Conversely, Part III yields results that are used in EasyCrypt but could be integrated in Squirrel. Further, as Part III builds on symbolic methods, it could also yield interesting byproducts for symbolic tools. In the future, we shall then

- continue to leverage symbolic methods to tackle security related questions with a foundational perspective, thus allowing to improve multiple tools at once;
- in parallel build bridges between the multiple tools, so that the analysis of security protocols can benefit from the strengths of each tool.

Beyond formal guarantees We strongly believe that the right to privacy is instrumental in shaping a free society. We have made some small steps in this direction, trying to help derive guarantees about privacy. Of course, we only looked at a small part of the chain, and much more work is required before we can derive global formal guarantees about the hardware, the implementation, the protocol design,...

But, looking even further away, and despite how it may appear in this work, we would like to emphasize that formal guarantees are not the only important point. We have seldom talked about one of the main link in the chain: human beings. It is not enough that protocols with strong guarantees exist: they must also be used in practice, by citizens, companies or states. If people don't believe that the right to privacy is essential, this won't happen. Security experts have a role to play in this, notably by informing people about privacy risks. Among other examples, this role has been highlighted with the recent debate about COVID tracking apps and popularization articles written by security experts (see e.g., [BCV+20] for an example in French). Also, many computer scientists have advocated for a long time that teaching computer sciences at a young age is important, notably w.r.t. privacy notions. They have for instance submitted reports to governing bodies, or more concretely set up popularization workshops in schools. We will participate in such endeavours in the future.

To conclude, we believe that our future work is two fold. First, aim to provide the best formal guarantees possible about the systems used in our everyday lives. Second, and maybe even more importantly, defend the right to privacy in our society.

Bibliography

[ABF17] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: mobile values, new names, and secure communication. *Journal of the ACM*, 65(1):1:1–1:41, October 2017.

[AC06] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.*, 367(1-2):2–32, 2006.

[ACD07] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. Combining algorithms for deciding knowledge in security protocols. In Franck Wolter, editor, *Proceedings of the 6th International Symposium on Frontiers of Combining Systems (FroCoS'07)*, volume 4720 of *Lecture Notes in Artificial Intelligence*, pages 103–117, Liverpool, UK. Springer, September 2007. DOI: 10.1007/978-3-540-74621-8_7.

[ACD12] M. Arapinis, V. Cheval, and S. Delaune. Verifying Privacy-Type Properties in a Modular Way. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 95–109, June 2012. DOI: 10.1109/CSF.2012.16.

[ACR+17] Tigran Avanesov, Yannick Chevalier, Michael Rusinowitch, and Mathieu Turuani. Satisfiability of general intruder constraints with and without a set constructor. *Journal of Symbolic Computation*, 80:27–61, 2017.

[ACZ13] Alessandro Armando, Roberto Carbone, and Luca Zanetti. *Formal modeling and automatic security analysis of two-factor and two-channel authentication protocols*. In *Network and System Security: 7th International Conference, NSS 2013, Madrid, Spain, June 3-4, 2013. Proceedings*. Javier Lopez, Xinyi Huang, and Ravi Sandhu, editors. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pages 728–734. ISBN: 978-3-642-38631-2. DOI: 10.1007/978-3-642-38631-2_63. URL: https://doi.org/10.1007/978-3-642-38631-2_63.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2001*, pages 104–115, New York. ACM, 2001.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In *Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics*, TCS '00, pages 3–22, Berlin, Heidelberg. Springer-Verlag, 2000. ISBN: 978-3-540-67823-6. URL: http://dl.acm.org/citation.cfm?id=647318.723498 (visited on 03/03/2019).

[AR02] M. Abadi and P. Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). *J. Cryptology*, 15(2):103–127, 2002.

[Ax68] James Ax. The elementary theory of finite fields. *Annals of Mathematics*, 88(2):239–271, 1968.

[BBB+19] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. Sok: computer-aided cryptography. 2019. URL: https://eprint.iacr.org/2019/1393.pdf.

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015. DOI: 10.1007/978-3-662-46800-5_18. URL: http://dx.doi.org/10.1007/978-3-662-46800-5_18.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016. DOI: 10.1145/2976749.2978427. URL: https://doi.org/10.1145/2976749.2978427.

[BC12] Gergei Bana and Hubert Comon-Lundh. Towards unconditional soundness: computationally complete symbolic attacker. In Pierpaolo Degano and Joshua D. Guttman, editors, *Proceedings of the 1st International Conference on Principles of Security and Trust (POST'12)*, volume 7215 of *Lecture Notes in Computer Science*, pages 189–208, Tallinn, Estonia. Springer, March 2012. DOI: 10.1007/978-3-642-28641-4_11. URL: http://www.lsv.fr/Publis/PAPERS/PDF/BC-post12.pdf.

[BC14a] Gergei Bana and Hubert Comon-Lundh. A computationally complete symbolic attacker for equivalence properties. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS'14)*, pages 609–620, Scottsdale, Arizona, USA. ACM Press, November 2014. DOI: 10.1145/2660267.2660276. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/BC-ccs14.pdf.

[BC14b] David Basin and Cas Cremers. Know your enemy. *ACM Transactions on Information and System Security*, 17(2), 2014.

[BCA+10] Bruno Blanchet, Vincent Cheval, Xavier Allamigeon, and Ben Smyth. Proverif: cryptographic protocol verifier in the formal model. *URL http://prosecco. gforge. inria. fr/personal/bblanche/proverif*, 17, 2010.

[BCE18] Gergei Bana, Rohit Chadha, and Ajay Kumar Eeralla. Formal Analysis of Vote Privacy Using Computationally Complete Symbolic Attacker. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, pages 350–372, 2018. DOI: 10.1007/978-3-319-98989-1_18. URL: https://doi.org/10.1007/978-3-319-98989-1_18.

[BCE+19] Gergei Bana, Rohit Chadha, Ajay Kumar Eeralla, and Mitsuhiro Okada. Verification methods for the computationally complete symbolic attacker based on indistinguishability. *ACM Transactions on Computational Logic (TOCL)*, 21(1):1–44, 2019.

[BCG+13] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella Béguelin. Fully automated analysis of padding-based encryption in the computational model. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 1247–1260. ACM, 2013. DOI: 10.1145/2508859.2516663. URL: http://doi.acm.org/10.1145/2508859.2516663.

[BCH10] Mayla Brusò, Konstantinos Chatzikokolakis, and Jerry den Hartog. Formal verification of privacy for RFID systems. In *CSF*, pages 75–88. IEEE Computer Society, 2010.

[BCJ+19] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. Automated methods for checking differential privacy. *CoRR*, abs/1910.04137, 2019. arXiv: 1910.04137. URL: http://arxiv.org/abs/1910.04137.

[BCV+20] Xavier Bonnetain, Anne Canteaut, VéroniqueCortier, Pierrick Gaudry, Lucca Hirschi, Steve Kremer Stéephanie Lacour, Matthieu Lequesne, Gaetan Leurent Léo Perrin, André Schrottenloher, Emmanuel Thomé, Serge Vaudenay, and Christophe Vuillot. 2020. URL: https://risques-tracage.fr/docs/risques-tracage.pdf.

[BDF+18] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. In *ASIACRYPT (3)*, volume 11274 of *Lecture Notes in Computer Science*, pages 222–249. Springer, 2018.

[BDG+13] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier López, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013. ISBN: 978-3-319-10081-4. DOI: 10.1007/978-3-319-10082-1_6. URL: https://doi.org/10.1007/978-3-319-10082-1_6.

[BDH15] David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In Luca Aceto and David de Frutos-Escrig, editors, *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 497–510, Madrid, Spain. Leibniz-Zentrum für Informatik, September 2015.

[BDH+08] Michael Backes, Markus Dürmuth, Dennis Hofheinz, and Ralf Küsters. Conditional reactive simulatability. *Int. J. Inf. Sec.*, 7(2):155–169, 2008.

[BDK+10] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 46–63. Springer, 2010. DOI: 10.1007/978-3-642-17511-4_4. URL: https://doi.org/10.1007/978-3-642-17511-4_4.

[BDK+20] David Baelde, Stéphanie Delaune, Adrien Koutsos, Charlie Jacomme, and Moreau Solène. An interactive prover for protocol verification in the computational model, 2020. URL: https://github.com/squirrel-submission-sp21/squirrel-prover. Under submission.

[BFG+18] Gilles Barthe, Xiong Fan, Joshua Gancher, Benjamin Grégoire, Charlie Jacomme, and Elaine Shi. Symbolic proofs for lattice-based cryptography. In Michael Backes and XiaoFeng Wang, editors, *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS'18)*, pages 538–555, Toronto, Canada. ACM Press, October 2018. URL: https://dl.acm.org/citation.cfm?doid=3243734.3243825.

[BFS+13] C. Brzuska, M. Fischlin, N. P. Smart, B. Warinschi, and S. C. Williams. Less is more: relaxed yet composable security notions for key exchange. en. *International Journal of Information Security*, 12(4):267–297, August 2013. ISSN: 1615-5270. DOI: 10.1007/s10207-013-0192-y. URL: https://doi.org/10.1007/s10207-013-0192-y (visited on 05/17/2019).

[BFW+11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-rogaway Key Exchange Protocols. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 51–62, New York, NY, USA. ACM, 2011. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046716. URL: http://doi.acm.org/10.1145/2046707.2046716 (visited on 01/30/2019).

[BGH+11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Heidelberg. Springer, 2011.

[BGJ+19] Gilles Barthe, Benjamin Grégoire, Charlie Jacomme, Steve Kremer, and Pierre-Yves Strub. Symbolic methods in computational cryptography proofs. In Stéphanie Delaune and Limin Jia, editors, *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'19)*, pages 136–151, Hoboken, NJ, USA. IEEE Computer Society Press, July 2019. DOI: 10.1109/CSF.2019.00017. URL: https://hal.inria.fr/hal-02117794.

[BGS15] Gilles Barthe, Benjamin Grégoire, and Benedikt Schmidt. Automated proofs of pairing-based cryptography. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1156–1168. ACM, 2015. DOI: 10.1145/2810103.2813697. URL: http://doi.acm.org/10.1145/2810103.2813697.

[BGV18a] Benjamin Bichsel, Timon Gehr, and Martin Vechev. Fine-grained semantics for probabilistic programs. In Amal Ahmed, editor, *Programming Languages and Systems*, pages 145–185, Cham. Springer International Publishing, 2018. ISBN: 978-3-319-89884-1.

[BGV18b] Benjamin Bichsel, Timon Gehr, and Martin Vechev. Fine-grained semantics for probabilistic programs. In *27th European Symposium on Programming (ESOP'18)*, volume 10801 of *Lecture Notes in Computer Science*, pages 145–185. Springer, 2018.

[BGZ09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101, New York. ACM, 2009.

[BHO+12] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 553–567. IEEE Computer Society, 2012. ISBN: 978-0-7695-4681-0. DOI: 10.1109/SP.2012.44. URL: http://dx.doi.org/10.1109/SP.2012.44.

[BJK20] Gilles Barthe, Charlie Jacomme, and Steve Kremer. Universal equivalence and majority on probabilistic programs over finite fields. In Naoki Kobayashi, editor, *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'20)*, Saarbrücken. ACM, July 2020. To appear.

[Bla06] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *27th IEEE Symposium on Security and Privacy, S&P 2006*, pages 140–154. IEEE Computer Society, 2006.

[Bla07] Bruno Blanchet. CryptoVerif: a computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied"*, October 2007.

[Bla09] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[Bla16] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends® in Privacy and Security*, 1(1-2):1–135, 2016. ISSN: 2474-1558. DOI: 10.1561/3300000004. URL: http://dx.doi.org/10.1561/3300000004.

[Bla18] Bruno Blanchet. Composition Theorems for CryptoVerif and Application to TLS 1.3. In *31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 16–30, Oxford, UK. IEEE Computer Society, July 2018.

[BLVGB+17] Vijay Bharadwaj, Hubert Le Van Gong, Dirk Balfanz, Alexei Czeskis, Arnar Birgisson, Jeff Hodges, Michael B. Jones, Rolf Lindemann, and J.C. Jones. Web authentication: an api for accessing public key credentials, December 2017. URL: https://www.w3.org/TR/2017/WD-webauthn-20171205/.

[Bom66] Enrico Bombieri. On exponential sums in finite fields. *American Journal of Mathematics*, 88(1):71–105, 1966.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The Reactive Simulatability (RSIM) Framework for Asynchronous Systems. *Inf. Comput.*, 205(12):1685–1720, December 2007. ISSN: 0890-5401. DOI: 10.1016/j.ic.2007.05.002. URL: http://dx.doi.org/10.1016/j.ic.2007.05.002.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, Heidelberg. Springer, 2006.

[Bra06] Mark Braverman. Termination of integer linear programs. In *International Conference on Computer Aided Verification*, pages 372–385. Springer, 2006.

[BRS15] David A. Basin, Sasa Radomirovic, and Michael Schläpfer. A complete characterization of secure human-server communication. In *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 199–213, 2015. DOI: 10.1109/CSF.2015.21. URL: https://doi.org/10.1109/CSF.2015.21.

[BRS16] D. Basin, S. Radomirovic, and L. Schmid. Modeling human errors in security protocols. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 325–340, 2016. DOI: 10.1109/CSF.2016.30.

[Buc76] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19–29, August 1976. ISSN: 0163-5824. DOI: 10.1145/1088216.1088219. URL: http://doi.acm.org/10.1145/1088216.1088219.

[Can00] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. 2000. URL: http://eprint.iacr.org/2000/067.

[CDD+17] Véronique Cortier, Constantin Cătălin Drăgan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 993–1008. IEEE, 2017.

[CGH+85] Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem of t-resilient functions (preliminary version). In *26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 396–407. IEEE Computer Society, 1985.

[CGT18] Véronique Cortier, David Galindo, and Mathieu Turuani. A formal analysis of the neuchâtel e-voting protocol. In *3rd IEEE European Symposium on Security and Privacy (EuroSP'18)*, pages 430–442, London, UK, 2018. DOI: 10.1109/EuroSP.2018.00037.

[CJS20] Hubert Comon, Charlie Jacomme, and Guillaume Scerri. Oracle simulation: a technique for protocol composition with long term shared secrets. In Jonathan Katz and Giovanni Vigna, editors, *Proceedings of the 27st ACM Conference on Computer and Communications Security (CCS'20)*, Orlando, USA. ACM Press, November 2020. To appear.

[CK17] Hubert Comon and Adrien Koutsos. Formal Computational Unlinkability Proofs of RFID Protocols. In Boris Köpf and Steve Chong, editors, *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 100–114, Santa Barbara, California, USA. IEEE Computer Society Press, August 2017. DOI: 10.1109/CSF.2017.9. URL: http://ieeexplore.ieee.org/document/8049714/.

[CKK+19] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. iUC: Flexible Universal Composability Made Simple. Technical report, 2019. URL: https://eprint.iacr.org/2019/1073.pdf.

[CKR18a] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. DEEPSEC: Deciding Equivalence Properties in Security Protocols - Theory and Practice. In *Proceedings of the 39th IEEE Symposium on Security and Privacy (S&P'18)*, pages 525–542, San Francisco, CA, USA. IEEE Computer Society Press, May 2018. DOI: 10.1109/SP.2018.00033.

[CKR18b] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Deepsec: deciding equivalence properties in security protocols theory and practice. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 529–546. IEEE, 2018.

[CKR+03] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with diffie-hellman exponentiation and products in exponents. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS 2003: Foundations of Software Technology and Theoretical Computer Science: 23rd Conference, Mumbai, India, December 15-17, 2003. Proceedings*, pages 124–135, Berlin, Heidelberg. Springer Berlin Heidelberg, 2003. ISBN: 978-3-540-24597-1. DOI: 10.1007/978-3-540-24597-1_11. URL: http://dx.doi.org/10.1007/978-3-540-24597-1_11.

[CKW10] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems. *Journal of Automated Reasoning*, 46(3-4):225–259, April 2010. DOI: 10.1007/s10817-010-9187-9. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CKW-jar10.pdf.

[CL06] Antoine Chambert-Loir. Compter (rapidement) le nombre de solutions d'équations dans les corps finis. 2006. URL: https://arxiv.org/abs/math/0611584.

[CMP19] Dmitry Chistikov, Andrzej S Murawski, and David Purser. Asymmetric distances for approximate differential privacy. In *30th International Conference on Concurrency Theory (CONCUR 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[CR03] Ran Canetti and Tal Rabin. Universal Composition with Joint State. en. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, Lecture Notes in Computer Science, pages 265–281. Springer Berlin Heidelberg, 2003. ISBN: 978-3-540-45146-4.

[CR10] Yannick Chevalier and Michael Rusinowitch. Symbolic protocol analysis in the union of disjoint intruder theories: combining decision procedures. *Theoretical Computer Science*, 411(10):1261–1282, 2010.

[CR16] Brent Carmer and Mike Rosulek. Linicrypt: a model for practical cryptography. In *36th Annual International Cryptology Conference (CRYPTO'16)*, volume 9816 of *Lecture Notes in Computer Science*, pages 416–445. Springer, 2016.

[Cre08] Cas Cremers. On the Protocol Composition Logic PCL. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '08, pages 66–76, New York, NY, USA. ACM, 2008. ISBN: 978-1-59593-979-1. DOI: 10.1145/1368310.1368324. URL: http://doi.acm.org/10.1145/1368310.1368324 (visited on 05/17/2019). event-place: Tokyo, Japan.

[CS18] Gaetan Cassiers and François-Xavier Standaert. Improved bitslice masking: from optimized non-interference to probe isolation. *IACR Cryptology ePrint Archive*, 2018:438, 2018. URL: https://eprint.iacr.org/2018/438.

[CSV19] Ran Canetti, Alley Stoughton, and Mayank Varia. Easyuc: using easycrypt to mechanize proofs of universally composable security. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 167–16716. IEEE, 2019.

[DDM+05] Anupam Datta, Ante Derek, John C. Mitchell, Vitaly Shmatikov, and Mathieu Turuani. Probabilistic Polynomial-Time Semantics for a Protocol Security Logic. en. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 16–29. Springer Berlin Heidelberg, 2005. ISBN: 978-3-540-31691-6.

[Del06] Stéphanie Delaune. Easy intruder deduction problems with homomorphisms. *Information Processing Letters*, 97(6):213 –218, 2006. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2005.11.008. URL: http://www.sciencedirect.com/science/article/pii/S0020019005003248.

[Del92] Gilles Deleuze. Postscript on the societies of control. *October*, 59:3–7, 1992.

[DG14] Daniel J. Dougherty and Joshua D. Guttman. Decidability for lightweight diffie-hellman protocols. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 217–231, 2014. DOI: 10.1109/CSF.2014.23. URL: http://dx.doi.org/10.1109/CSF.2014.23.

[DKP12] Stéphanie Delaune, Steve Kremer, and Daniel Pasaila. Security protocols, constraint systems, and group theories. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR'12)*, volume 7364 of *Lecture Notes in Artificial Intelligence*, pages 164–178, Manchester, UK. Springer, June 2012. DOI: 10.1007/978-3-642-31365-3_15. URL: https://members.loria.fr/skremer/files/Papers/CKP-ijcar12.pdf.

[DMP03] Nancy Durgin, John Mitchell, and Dusko Pavlovic. A Compositional Logic for Proving Security Properties of Protocols. *J. Comput. Secur.*, 11(4):677–721, July 2003. ISSN: 0926-227X. URL: http://dl.acm.org/citation.cfm?id=959088.959095 (visited on 05/28/2019).

[Dwo60] Bernard Dwork. On the rationality of the zeta function of an algebraic variety. *American Journal of Mathematics*, 82(3):631–648, 1960.

[DY81] D. Dolev and A. Chi-Chih Yao. On the security of public key protocols. In *22nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 1981*, pages 350–357. IEEE Computer Society, 1981.

[Ecs] Easycyrpt github repository. 2019. URL: `https://github.com/EasyCrypt/easycrypt/tree/deploy-solveeq`.

[Eis13] David Eisenbud. *Commutative Algebra: with a view toward algebraic geometry*, volume 150. Springer Science & Business Media, 2013.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer, 2004.

[FG14] Marc Fischlin and Felix Günther. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1193–1204, New York, NY, USA. ACM, 2014. ISBN: 978-1-4503-2957-6. DOI: `10.1145/2660267.2660308`. URL: `http://doi.acm.org/10.1145/2660267.2660308` (visited on 05/28/2019). event-place: Scottsdale, Arizona, USA.

[Fid] FIDO Yubikey. Web site, 2018. URL: `https://www.yubico.com/solutions/fido-u2f/`. Accessed in January 2018.

[FID18] FIDO. Universal 2nd Factor (U2F), 2018. URL: `https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/FIDO-U2F-COMPLETE-v1.2-ps-20170411.pdf`.

[FJ14] Matthew Fredrikson and Somesh Jha. Satisfiability modulo counting: a new approach for analyzing privacy properties. In *Joint Meeting of the 23rd Annual Conference on Computer Science Logic (CSL) and the 29th ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10. ACM, 2014.

[FKS14] Daniel Fett, Ralf Küsters, and Guido Schmitz. An Expressive Model for the Web Infrastructure: Definition and Application to the BrowserID SSO System. In *35th IEEE Symposium on Security and Privacy (S&P 2014)*, pages 673–688. IEEE Computer Society, 2014. URL: `https://publ.sec.uni-stuttgart.de/fettkuestersschmitz-sp-2014.pdf`.

[Fou75] Michel Foucault. Surveiller et punir. *Paris*, 1:192–211, 1975.

[G2s] Google 2 Step Verification. Web site, 2018. URL: `https://www.google.com/landing/2step/`. Accessed in January 2018.

[GFN+17] Paul A. Grassi, James L. Fenton, Elaine M. Newton, Ray A. Perlner, Andrew R. Regenscheid, William E. Burr, Justin P. Richer, Naomi B. Lefkovitz, Jamie M. Danker, Kristen K. Choong Yee-Yin Greene, and Mary F. Theofanos. Nist special publication 800-63b – digital identity guidelines – authentication and lifecycle management, June 2017. Available at `https://doi.org/10.6028/NIST.SP.800-63b`.

[GGF17] Paul A. Grassi, Michael E. Garcia, and James L. Fenton. Nist special publication 800-63-3 – digital identity guidelines, June 2017. Available at `https://doi.org/10.6028/NIST.SP.800-63-3`.

[GHS+20] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David Basin. A spectral analysis of noise: a comprehensive, automated, formal analysis of diffie-hellman protocols, 2020.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017. ISBN: 978-3-319-52152-7. DOI: 10.1007/978-3-319-52153-4_6. URL: https://doi.org/10.1007/978-3-319-52153-4_6.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GNP19] Marco Gaboardi, Kobbi Nissim, and David Purser. The complexity of verifying circuits as differentially private. *CoRR*, abs/1911.03272, 2019. arXiv: 1911.03272. URL: http://arxiv.org/abs/1911.03272.

[GNW00] Qing Guo, Paliath Narendran, and D.A. Wolfram. Complexity of nilpotent unification and matching problems. *Information and Computation*, 162(1):3–23, 2000. ISSN: 0890-5401. DOI: https://doi.org/10.1006/inco.1999.2849. URL: http://www.sciencedirect.com/science/article/pii/S0890540199928493.

[Gol05] Oded Goldreich. *Foundations of cryptography: a primer*, volume 1. Now Publishers Inc, 2005.

[Hal05] S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005.

[HBD19] Lucca Hirschi, David Baelde, and Stéphanie Delaune. A method for unbounded verification of privacy-type properties. *J. Comput. Secur.*, 27(3):277–342, 2019.

[HS15] Dennis Hofheinz and Victor Shoup. GNUC: A New Universal Composability Framework. *Journal of Cryptology*, 28(3):423–508, July 2015. ISSN: 1432-1378. DOI: 10.1007/s00145-013-9160-y. URL: https://doi.org/10.1007/s00145-013-9160-y.

[Iso] ISO/IEC 9798-3:2019, IT Security techniques – Entity authentication – Part 3: Mechanisms using digital signature techniques. en. URL: https://www.iso.org/standard/67115.html (visited on 05/28/2019).

[JCC+19] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. Seems legit: automated analysis of subtle attacks on protocols that use signatures. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2165–2180. ACM, 2019. DOI: 10.1145/3319535.3339813. URL: https://doi.org/10.1145/3319535.3339813.

[JK18] Charlie Jacomme and Steve Kremer. An extensive formal analysis of multi-factor authentication protocols. In Steve Chong and Stéphanie Delaune, editors, *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 1–15, Oxford, UK. IEEE Computer Society Press, July 2018. DOI: 10.1109/CSF.2018.00008. URL: https://ieeexplore.ieee.org/document/8429292/.

[Joh16] William Andrew Johnson. *Fun with fields*. PhD thesis, UC Berkeley, 2016.

[JR10] Charanjit S. Jutla and Arnab Roy. A completeness theorem for pseudo-linear functions with applications to uc security. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:92, 2010.

[JW09] Ari Juels and Stephen A. Weis. Defining strong privacy for RFID. *ACM Trans. Inf. Syst. Secur.*, 13(1):7:1–7:23, 2009.

[Kay05] Neeraj Kayal. Recognizing permutation functions in polynomial time. *ECCC, TR05-008*, 173:185–311, 2005.

[KGG+18] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: exploiting speculative execution. *meltdownattack.com*, 2018. URL: https://spectreattack.com/spectre.pdf.

[Kie76] Catarina Kiefe. Sets definable over finite fields: their zeta-functions. *Transactions of the American Mathematical Society*, 223:45–59, 1976.

[KK16] Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. *Journal of Computer Security*, 24(5):583–616, 2016. DOI: 10.3233/JCS-160556. URL: https://hal.inria.fr/hal-01351388.

[KMT12] Steve Kremer, Antoine Mercier, and Ralf Treinen. Reducing Equational Theories for the Decision of Static Equivalence. *Journal of Automated Reasoning*, 48(2):197–217, 2012. DOI: 10.1007/s10817-010-9203-0. URL: https://hal.inria.fr/inria-00636797.

[Kni89] Kevin Knight. Unification: a multidisciplinary survey. *ACM Comput. Surv.*, 21(1):93–124, March 1989. ISSN: 0360-0300. DOI: 10.1145/62029.62030. URL: http://doi.acm.org.ins2i.bib.cnrs.fr/10.1145/62029.62030.

[Kou19a] Adrien Koutsos. Decidability of a sound set of inference rules for computational indistinguishability. In Stéphanie Delaune and Limin Jia, editors, *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'19)*, Hoboken, NJ, USA. IEEE Computer Society Press, July 2019. To appear.

[Kou19b] Adrien Koutsos. *Symbolic Proofs of Computational Indistinguishability*. PhD thesis, 2019.

[Kou19c] Adrien Koutsos. The 5g-aka authentication protocol privacy. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 464–479. IEEE, 2019. DOI: 10.1109/EuroSP.2019.00041. URL: https://doi.org/10.1109/EuroSP.2019.00041.

[KR17] Ralf Küsters and Daniel Rausch. A Framework for Universally Composable Diffie-Hellman Key Exchange. In *IEEE 38th Symposium on Security and Privacy (S&P 2017)*, pages 881–900. IEEE Computer Society, 2017.

[KR19] Yael Tauman Kalai and Leonid Reyzin. A survey of leakage-resilient cryptography. 2019.

[KR94] Hélène Kirchner and Christophe Ringeissen. Combining symbolic constraint solvers on algebraic domains. *J. Symb. Comput.*, 18(2):113–155, 1994. DOI: 10.1006/jsco.1994.1040. URL: https://doi.org/10.1006/jsco.1994.1040.

[KS13] Robert Künnemann and Graham Steel. *Yubisecure? formal security analysis results for the yubikey and yubihsm.* In *Security and Trust Management: 8th International Workshop, STM 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*. Audun Jøsang, Pierangela Samarati, and Marinella Petrocchi, editors. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pages 257–272. ISBN: 978-3-642-38004-4. DOI: 10.1007/978-3-642-38004-4_17. URL: https://doi.org/10.1007/978-3-642-38004-4_17.

[KT11] Ralf Küsters and Max Tuengerthal. Composition Theorems Without Pre-established Session Identifiers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 41–50, New York, NY, USA. ACM, 2011. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046715. URL: http://doi.acm.org/10.1145/2046707.2046715 (visited on 05/17/2019). event-place: Chicago, Illinois, USA.

[Len85] A.K. Lenstra. Factoring multivariate polynomials over finite fields. *Journal of Computer and System Sciences*, 30(2):235 –248, 1985. ISSN: 0022-0000. DOI: http://dx.doi.org/10.1016/0022-0000(85)90016-9. URL: http://www.sciencedirect.com/science/article/pii/0022000085900169.

[LGM98] Michael L Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

[LMO+08] Axel Legay, Andrzej S Murawski, Joël Ouaknine, and James Worrell. On automated verification of probabilistic programs. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 2008.

[LN83] Rudolf Lidl and Harald Niederreiter. *Finite fields*. Addison-Wesley, 1983.

[LSB+19] Andreas Lochbihler, S Reza Sefidgar, David Basin, and Ueli Maurer. Formalizing constructive cryptography using crypthol. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 152–15214. IEEE, 2019.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *meltdownattack.com*, 2018. URL: https://meltdownattack.com/meltdown.pdf.

[LW06] Alan GB Lauder and Daqing Wan. Counting points on varieties over finite fields of small characteristic. *arXiv preprint math/0612147*, 2006.

[Mau01] Stefan Maubach. The automorphism group over finite fields, 2001.

[Mau11] Ueli Maurer. Constructive cryptography - A new paradigm for security definitions and proofs. In *TOSCA*, volume 6993 of *Lecture Notes in Computer Science*, pages 33–56. Springer, 2011.

[Mfa] Proverif source files. 2018. URL: https://gitlab.inria.fr/cjacomme/multi-factor-authentication-proverif-examples.

[Mil99] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[MO05] Andrzej S. Murawski and Joël Ouaknine. On probabilistic program equivalence and refinement. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2005. DOI: 10.1007/11539452_15. URL: https://doi.org/10.1007/11539452_15.

[MP13] Gary L Mullen and Daniel Panario. *Handbook of finite fields*. Chapman and Hall/CRC, 2013.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001, Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 166–175. ACM, 2001. URL: https://doi.org/10.1145/501983.502007.

[MSC+13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 696–701. Springer, 2013.

[MST84] Maurice Mignotte, Tarlok Nath Shorey, and Robert Tijdeman. The distance between terms of an algebraic recurrence sequence. *Journal für die reine und angewandte Mathematik*, (349):63–76, 1984.

[MT79] Robert Morris and Ken Thompson. Password security: a case history. *Communications of the ACM*, 22(11):594–597, 1979.

[Mvs] Maskverif source files. 2019. URL: https://sites.google.com/site/symbolicforcrypto/.

[MW04] David Molnar and David A. Wagner. Privacy and security in library RFID: issues, practices, and architectures. In *ACM Conference on Computer and Communications Security*, pages 210–219. ACM, 2004.

[MW12] Scott McCallum and Volker Weispfenning. Deciding polynomial-transcendental problems. *Journal of Symbolic Computation*, 47(1):16–31, 2012.

[Nie71] H Niederreiter. Orthogonal systems of polynomials in finite fields. *Proceedings of the American Mathematical Society*, 28(2):415–422, 1971.

[Nip90] Tobias Nipkow. Unification in primal algebras, their powers and their varieties. *J. ACM*, 37(4):742–776, October 1990. ISSN: 0004-5411.

[OW12] Joël Ouaknine and James Worrell. Decision problems for linear recurrence sequences. In *6th International Workshop on Reachability Problems (RP'12)*, volume 7550 of *Lecture Notes in Computer Science*, pages 21–28. Springer, 2012.

[OW14a] Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In *25th ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 366–379. Society for Industrial and Applied Mathematics, 2014.

[OW14b] Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 330–341. Springer, 2014.

[PNB+18] Andrey Popov, Magnus Nystrom, Dirk Balfanz, Adam Langley, Nick Harper, and Jeff Hodges. Token Binding over HTTP. Internet-Draft draft-ietf-tokbind-https-12, Internet Engineering Task Force, January 2018. 24 pages. URL: `https://datatracker.ietf.org/doc/html/draft-ietf-tokbind-https-12`. Work in Progress.

[PRW17] Olivier Pereira, Florentin Rochet, and Cyrille Wiedling. Formal Analysis of the Fido 1.x Protocol. In *The 10th International Symposium on Foundations & Practice of Security*, Lecture Notes in Computer Science (LNCS). Springer, October 2017.

[PSS+11] Kenneth G. Paterson, Jacob C. N. Schuldt, Martijn Stam, and Susan Thomson. On the Joint Security of Encryption and Signature, Revisited. en. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, Lecture Notes in Computer Science, pages 161–178. Springer Berlin Heidelberg, 2011. ISBN: 978-3-642-25385-0.

[Sce15] Guillaume Scerri. *Proofs of security protocols revisited*. PhD thesis, 2015.

[SCR+18] Giada Sciarretta, Roberto Carbone, Silvio Ranise, and Luca Viganò. Design, formal specification and analysis of multi-factor authentication solutions with a single sign-on experience. In *International Conference on Principles of Security and Trust*, pages 188–213. Springer, Cham, 2018.

[Seq] Solveq github repository, 2019. URL: `https://github.com/EasyCrypt/solveq`.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.

[SMC+12] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 78–94, June 2012. DOI: `10.1109/CSF.2012.25`.

[Squ] Squirrel source files and case-studies. 2020. URL: `https://github.com/squirrel-submission-sp21/squirrel-prover`.

[SR16] Guillaume Scerri and Stanley-Oakes Ryan. Analysis of Key Wrapping APIs: Generic Policies, Computational Security. en. In pages 281–295. IEEE Computer Society, June 2016. DOI: `10.1109/CSF.2016.27`. URL: `https://hal.inria.fr/hal-01417123` (visited on 02/27/2019).

[SS88] David Shannon and Moss Sweedler. Using gröbner bases to determine algebra membership, split surjective algebra homomorphisms determine birational equivalence. *J. Symb. Comput.*, 6(2-3):267–273, December 1988. ISSN: 0747-7171. DOI: `10.1016/S0747-7171(88)80047-6`. URL: `http://dx.doi.org/10.1016/S0747-7171(88)80047-6`.

[tea17] G Suite team. G suite updates. Blog entry, February 2017. URL: `https://gsuiteupdates.googleblog.com/2017/02/improved-phone-prompts-for-2-step.html`.

[Tor88] Jacobo Toràn. An oracle characterization of the counting hierarchy. In *3rd Annual Structure in Complexity Theory Conference*, pages 213–223, 1988.

[Tor91] Jacobo Torán. Complexity classes defined by counting quantifiers. *Journal of the ACM*, 38:753–774, 1991.

[Ver85] NK Vereshchagin. The problem of appearance of a zero in a linear recurrence sequence. *Mat. Zametki*, 38(2):609–615, 1985.

[Wei49] André Weil. Numbers of solutions of equations in finite fields. *Bulletin of the AMS*, 1949.

[YL] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Transport Layer Protocol. en. URL: `https://tools.ietf.org/html/rfc4253` (visited on 05/17/2019).

A Appendix of Part I

A.1 Global Results for MFA

We summarize in Table A.1, A.2, an A.3 all the results we computed using the automated generation of scenarios, the captions being given in Figure A.1. The results were obtained in 8 minutes of computing on a server with 12 Intel(R) Xeon(R) CPU X5650 @ 2.67GHz and 50Gb of RAM. During the computation, 6172 calls to ProVerif were made. As ProVerif may not terminate we set a timeout at 3 seconds: only two scenarios exceeded the timeout limit. For readability, we only display the minimal or maximal interesting scenarios, and results which are implied by an other scenario are greyed. The table was completely generated by an automated script, to avoid transcription mistakes.

Threat Scenarios		g2V	g2VFPR	g2VD	g2ST	g2STFPR	g2STD	g2DTFPR	g2DTD	g2DTDE	U2F	TB-U2F	TB-g2DTD
		✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓
		✗	✓	✓/✓	✗	✓	✓	✓	✓/✓	✓	✓	✓	✓/✓
PH		✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	✓	✓
	NC	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	✓	✓
	FS	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	✓	✓
PH	NC	✗	✗	✗	✗	✗	✗	✗	✗	✓/✓	✓	✓/✓	✗
PH	FS	✗	✗	✗/✓	✗	✗	✗	✗	✗/✓	✓	✓	✓	✗/✓
PH	FS	✗	✗	✗/✗	✗	✗	✗	✗	✗/✗	✓/-	✓/✗	✓/✗	✗/✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗/-	✓/✗	✓/✗	✓	✗/✓
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗/✓	✓	✓	✓	✗/✓
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✓/✗	✓/✗	✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✓/✗	✓/✓	✗
PH	FS	✗	✗	✗/✗	✗	✗	✗	✗	✗	✓/✗	✓/✗	✓/✗	✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗/-	✓/✗	✓/✗	✓	✗/✗
PH	FS	✗	✗	✓/✓	✗	✗	✗	✗	✗	✓/✗	✓	✓	✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗/✗	✓/-	✓/✗	✓/✗	✗/✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✗	✗	✗/-
PH	FS	✗	✗	✗/✗	✗	✗	✗	✗	✗/✗	✓/✗	✓/✗	✓/✗	✗/✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✗	✗	✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✓/✓	✓/✓	✓/✓
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✗	✗	✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✓/✗	✓/✗	✗/✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✗	✗	✗
PH	FS	✗	✗	✗	✗	✗	✗	✗	✗	✓/✗	✓/✗	✓/

	Threat Scenarios	g2V	g2VFPR	g2VD	g2ST	g2STFPR	g2STD	g2DTFPR	g2DTD	g2DTDE	U2F	TB-U2F	TB-g2DTD
	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✗✗	✓✓✗	✖	✗✗	✓✓✗	✗✗	✓✓✗	✓✓✗	✖	✗✗	✓✓✗
NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✓✓✓	✓	✖
NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$ $\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✖	✖	✖
NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✓	✓✓✓	✖
NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✖	✖	✖
	$\mathcal{M}^{t-\mathcal{RO}}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖	✖	✓	✓	✖	✓	✖	✓✓✓	✓	✖	✖	✓✓✓
NC	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✓	✖	✖	✖
	$\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✓✓✗	✓✓✗	✓✓✗	✖	✖	✖	✓✓✗	✓✓✗	✓✓-	✓✓✗	✓✓✗	✓✓✗
FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✗✗	✖	✖	✖	✓✓-	✓✓✗	✓✓✗	✓✓✗	✓	✓
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✖	✗✗	✖	✖	✖	✓✓✓	✓	✓	✓	✓	✓
FS NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✓✓-	✓✓✗	✓✓✗	✓✓✗	✓✓	✓✓✓
FS	$\mathcal{M}^{t-mem}_{in:\mathcal{RW}}$	✓✓✗	✓✓✗	✓✓✗	✖	✖	✖	✓✓✗	✓✓✗	✓✓✗	✓✓✗	✓✓✗	✓✓✗
FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✗✗	✖	✖	✖	✓✓-	✓✓✗	✓✓-	✓✓✗	✓✓✗	✓✓✗
FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✓✓✓	✓✓✓	✓	✖	✖	✖	✖	✓✗✗	✓✗✗	✓	✓	✓✗✗
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✗✗	✖	✖	✖	✓✓✗	✓✓✗	✓✓-	✓✓✗	✓✓✗	✓✓✗
FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✗✗	✖	✖	✖	✗✗✗	✓✗✗	✓✗✗	✖	✗✗✗	✓✗-
FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✓✓-	✓✓-	✓✓✗	✓✓✗	✓✓✗	✓✓✗
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✓✓✗	✓✓✗	✓✓-	✓✓✗	✓✓✗	✓✓✗
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✓✗✗	✓	✓✓✗	✓	✓✗✗
FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{io:\mathcal{RW}}$	✓✓✗	✓✓✗	✓✓✗	✖	✖	✖	✖	✖	✗✗	✓✓✗	✓✓✗	✖
FS NC	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✗✗	✖	✖	✖	✖	✗✗	✓✓✗	✓✓✗	✓✓✗	✓✗✗
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✗✗	✖	✖	✖	✖	✗✗	✓✓✗	✓✓✗	✓✓✗	✓✗✗
FS NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✗✗	✖	✖	✖	✖	✗✗	✓✓✗	✓✓✗	✓✓✗	✓✗✗
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✓✓✗	✖
FS NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✓✓✗	✖
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✓✓✗	✖
FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✓✓✗	✖
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓	✖	✖	✖
FS	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✓✓✗	✖
FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓	✖	✖	✖
FS	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS NC	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-mem}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS NC	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖

Table A.2: Global Results for Malware on Trusted Platform - Part 2

	Threat	Scenarios	g2V	g2VFPR	g2VD	g2ST	g2STFPR	g2STD	g2DTFPR	g2DTD	g2DTDE	U2F	TB-U2F	TB-g2DTD
PH	NC	$\mathcal{M}^{\mathbf{u}-dis}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖
		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✖	✗/✓	✗/✓	✔	✔	✗/✓
	PH	FS	$\mathcal{M}^{\mathbf{u}-mem}_{in:\mathcal{RO}}$	✔	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✔
			$\mathcal{M}^{\mathbf{u}-dis}_{io:\mathcal{RO}}$	✔	✔	✔	✖	✔	✔	✔	✓/✓	✓/✓	✔	✔	✓/✓
			$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RO}}$	✖	✓/✓	✔	✖	✓/✓	✔	✔	✔	✔	✔	✔	✔
			$\mathcal{M}^{\mathbf{u}-usb}_{in:\mathcal{RO}}$	✖	✓/✓	✔	✖	✔	✔	✔	✔	✔	✔	✔	✔
	NC		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✔	✔	✔	✔	✔	✔	✔
	NC	NC	$\mathcal{M}^{\mathbf{u}-usb}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✔	✔	✔	✔	✔	✔	✔
			$\mathcal{M}^{\mathbf{u}-usb}_{in:\mathcal{RO}}$	✖	✖	✖	✓/✗	✓	✓/✗	✓/✓	✔	✖	✖	✓/✓	✓/✓
		NC	$\mathcal{M}^{\mathbf{u}-dis}_{io:\mathcal{RW}}$	✔	✔	✖	✖	✖	✖	✖	✖	✔	✔	✔	✖
			$\mathcal{M}^{\mathbf{u}-usb}_{in:\mathcal{RW}}$	✖	✔	✔	✓/✗	✓	✓/✗	✔	✔	✓/✗	✓/✗	✓/✗	✔
	NC		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
	NC		$\mathcal{M}^{\mathbf{u}-usb}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓/✓	✓/✗	✖	✖
	NC		$\mathcal{M}^{\mathbf{u}-dis}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
	NC		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✓/✗	✓	✓/✗	✓/✓	✔	✓/✗	✓/✗	✓/✗	✔
	NC		$\mathcal{M}^{\mathbf{u}-usb}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
	NC		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✓/✗	✓	✓/✗	✓/✓	✔	✓/✗	✓/✗	✓/✗	✔
	NC		$\mathcal{M}^{\mathbf{u}-usb}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
	NC		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✔	✓/✓	✔	✓/✗	✓/✗	✓/✗	✔
	NC		$\mathcal{M}^{\mathbf{u}-usb}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓/✗	✗/✗	✖	✖
	NC		$\mathcal{M}^{\mathbf{u}-usb}_{io:\mathcal{RW}}$	✖	✖	✖	✓/✗	✓	✓/✗	✓/✓	✔	✗/✗	✗/✗	✗/✗	✔
	FS		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RO}}$	✖	✖	✗/✓	✖	✖	✖	✔	✓/✓	✔	✔	✔	✓/✓
	FS		$\mathcal{M}^{\mathbf{u}-usb}_{io:\mathcal{RO}}$	✖	✖	✗/✓	✖	✖	✖	✔	✓/✓	✔	✔	✔	✓/✓
	FS		$\mathcal{M}^{\mathbf{u}-tls}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✗/✓	✓/✓	✔	✔	✗/✓
	FS		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✔	✔	✓/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-usb}_{io:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-dis}_{in:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-usb}_{in:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-dis}_{io:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-tls}_{io:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-usb}_{io:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗
	FS		$\mathcal{M}^{\mathbf{u}-dis}_{io:\mathcal{RW}}$	✖	✖	✗/✓	✖	✖	✖	✗/✗	✗/✗	✓/✓	✔	✔	✗/✗

Table A.3: Global Results for Malware on Untrusted Platform

Protocols

- G2V- *Google 2-step* with Verification code
- G2V^{FPR}- G2V with fingerprint display
- G2V^{DIS}- G2V^{FPR} with action display
- G2OT- *Google 2-step* One Tap
- G2OT^{FPR}- G2OT with fingerprint display
- G2OT^{DIS}- G2OT with action display
- G2DT^{FPR}- *Google 2-step* Double Tap (random to compare)
- G2DT^{DIS}- G2DT^{FPR} with action display
- U2F- *FIDO's U2F*
- U2F_{TB}- TokenBinding U2F
- $\text{G2DT}^{\text{DIS}}_{\text{TB}}$- TokenBinding G2DT^{DIS}

Scenarios:

- NC- No Compare, the human does not compare values
- FS- Fingerprint spoof, the attacker can copy the user IP address
- PH- The user might be victim of phishing only on trusted everyday connections or untrusted computers
- $\mathcal{M}^{interface}_{in:acc1,out:acc2}$- The interface inputs are given to the attacker with access level acc1, and acc2 for the outputs

Notations:

- ✓- Property satisfied (✔if all three)
- ✗- Attack found (✖if all three)
- ✗- Attack also present in a weaker scenario
- ✓- Property also satisfied in a stronger scenario
- - - Either scenario not pertinent, or failure to reconstruct attack trace

Figure A.1: Caption for Global Results

B Appendix of Part II

B.1 Formal Corollary for Key Exchange

We denote $\overline{p} = \{id^I, id^R\}$ and $\overline{s} = \{lsid_i^I, lsid_i^R\}_{i\in\mathbb{N}}$ the set of all the copies of the local session identifiers.

Formalizing the previous Section, to prove the security of a key exchange, we can use the following Corollary of Theorem 5.5.

Corollary B.1. *Let $\mathcal{O}_{ke}$, $\mathcal{O}$ be oracles and $KE_i[_1,_2] := I(lsid_i^I, id^I);_1 \| R(lsid_i^R, id^R);_2$ a key exchange protocol, such that I binds $x^I, x_{id}^I, x_{lsid}^I$, R binds $x^R, x_{id}^R, x_{lsid}^R$ and $\mathcal{N}_l(KE)$ is disjoint of the oracle support. Let id^I, id^R be names and $\overline{s}^I = \{lsid_i^I\}_{i\in\mathbb{N}}, \overline{s}^R = \{lsid_i^R\}_{i\in\mathbb{N}}$ sets of names :*

1. $\forall i \geq 1, (\nu lsid_i^I, id^I, lsid_i^R, id^R.$

$$KE_i[\textbf{out}(\langle x^I, lsid_i^I, x_{lsid}^I, x_{id}^I\rangle), \textbf{out}(\langle x^R, lsid_i^R, x_{lsid}^R, x_{id}^R\rangle)] \| \textbf{out}(\langle lsid_i^R, lsid_i^I\rangle)$$

   is $\mathcal{O}_{ke}$ simulatable)).
2. *$\overline{s}$ is disjoint of the support of $\mathcal{O}$.*

$$KE_0[\textbf{out}(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle), \textbf{out}(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle)] \cong_{\mathcal{O}_{ke},\mathcal{O}}$$

$$\begin{aligned} KE_0[\quad & \textbf{if } x_{lsid}^I = lsid_0^R \wedge x_{id}^I = id^R \textbf{ then} \\ & \quad \textbf{out}(\langle k, lsid_0^I, x_{lsid}, x_{id}\rangle) \\ & \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} \\ & \quad \perp \\ & \textbf{else } out(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle), \\ & \textbf{if } x_{lsid}^R = lsid_0^I \wedge x_{id}^R = id^I \textbf{ then} \\ & \quad \textbf{out}(\langle k, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle) \\ & \textbf{else if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} \\ & \quad \perp \\ & \textbf{else out}(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle)] \end{aligned}$$

Then, for any N which depends on the security parameter:

$$\begin{aligned} \|^{i\leq N} KE_i[\textbf{out}(x^I), \textbf{out}(x^R)] \cong_{\mathcal{O}} \\ \|^{i\leq N} \quad KE_i[\quad & \textbf{if } (x_{id}^I = id^R) \textbf{ then} \\ & \quad \underset{1\leq j\leq N}{\textbf{if}} x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id_R \textbf{ then} \\ & \quad\quad \textbf{out}(k_{i,j}) \\ & \textbf{else out}(x^I), \\ & \textbf{if } (x_{id}^R = id_I) \textbf{ then} \\ & \quad \underset{1\leq j\leq N}{\textbf{if}} x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id_I \textbf{ then} \\ & \quad\quad \textbf{out}(k_{j,i}) \\ & \textbf{else out}(x^R)] \end{aligned}$$

Then, building upon the previous Corollary and the sequential composition Theorems, the following Corollary shows the precise requirements to prove the security of a protocol which uses a key exchange, for an bounded number of session and with long term secrets shared between the key exchange and the protocol.

Corollary B.2. *Let $\mathcal{O}_T$, $\mathcal{O}_{ke}$, $\mathcal{O}_r$,$\mathcal{O}_{P,Q}$ be oracles and*

$KE_i[_1,_2] := I(lsid_i^I, id^I);_1 \| R(lsid_i^R, id^R);_2$ a key exchange protocol, such that I binds $x^I, x_{id}^I, x_{lsid}^I$, R binds $x^R, x_{id}^R, x_{lsid}^R$ and $\mathcal{N}_l(KE)$ is disjoint of the oracle support. Let id^I, id^R be names, $\overline{s}^I = \{lsid_i^I\}_{i\in\mathbb{N}}, \overline{s}^R = \{lsid_i^R\}_{i\in\mathbb{N}}$ and $\overline{s} = \overline{s}^I \cap \overline{s}^R$ sets of names.

Let $\overline{p} = \{id^I, id^R\}$, $P(x,\overline{y}) = P_1(x,\overline{y})\|P_2(x,\overline{y})$ *and* $Q(x,\overline{y},\overline{z}) = Q_1(x,\overline{y},\overline{z})\|Q_1(x,\overline{y},\overline{z})$ *be parameterized protocols, such that* $\mathcal{N}_l(P,Q)$ *is disjoint of the oracle support.*

I-1 $\forall i \geq 1, (\nu lsid_i^I, id^I, lsid_i^R, id^R.KE_i[\textbf{out}(x^I), \textbf{out}(x^R)]\|\textbf{out}(\langle lsid_i^R, lsid_i^I\rangle)$ *is* $\mathcal{O}_T$*-simulatable)).*

I-2 $\overline{s}$ *is disjoint of the support of* $\mathcal{O}_{P,Q}$.

$$KE_0[\textbf{out}(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle), \textbf{out}(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle) \cong_{\mathcal{O}_T,\mathcal{O}_{P,Q}}$$

I-3

$$\begin{aligned} KE_0 \quad & [\textbf{if } x_{lsid}^I = lsid_0^R \wedge x_{id}^I = id^R \textbf{ then} \\ & \quad \textbf{out}(\langle k, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle) \\ & \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} \\ & \quad \bot \\ & \textbf{else out}(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle), \\ & \textbf{if } x_{lsid}^R = lsid^I \wedge x_{id}^R = id^I \textbf{ then} \\ & \quad \textbf{out}(\langle k, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle) \\ & \textbf{else if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} \\ & \quad \bot \\ & \textbf{else out}(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle)] \end{aligned}$$

and

R-1 $\forall\ 1 \leq i,j \leq n, \nu\overline{p}, k_{i,j}.P_0(\overline{p}, k_{i,j})$ *is* $\mathcal{O}_r$*-simulatable.*

R-2 $\forall\ 1 \leq i \leq n, \nu\overline{p}, k_{i,j}.Q_0(\overline{p}, k_{i,j})$ *is* $\mathcal{O}_r$*-simulatable.*

R-3 $\overline{s}$ *is disjoint of the support of* $\mathcal{O}_k$.

R-4 $P_0(\overline{p}, k) \cong_{\mathcal{O}_r,\mathcal{O}_{ke}} Q_0(\overline{p}, k)$

and

C-1 $\nu\overline{p}.in(x_i^I).P_i^I(x_i^I)\|in(x_i^R).P_i^R(x_i^R)$ *is* $\mathcal{O}_{P,Q}$*-simulatable.*

1. $\nu\overline{p}.$

$$\begin{aligned} \|^{i\leq n} \quad KE_i[\quad & \textbf{if } (x_{id}^I = id^R) \textbf{ then} \\ & \underset{1\leq j\leq n}{\textit{if}} (x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id_R) \textbf{ then} \\ & \quad out(\langle i,j\rangle) \\ & \textbf{else } P_i^I(x_i^I), \\ & \textbf{if } (x_{id}^R = id_I) \textbf{ then} \\ & \underset{1\leq j\leq n}{\textit{if}} (x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id_I) \textbf{ then} \\ & \quad out(\langle i,j\langle) \\ & \textbf{else } P_i^R(x_i^R)] \end{aligned}$$

is $\mathcal{O}_{ke}$*-simulatable.*

Then, for any n *which may depend on the security parameter:*

$$\|^{i\leq n} KE_i[P_i^I(x_i^I), P_i^R(x_i^R)] \cong$$

$$\|^{i\leq n} KE_i[\textbf{if } x_{id}^I = id^R \textbf{ then } Q_i^I(x_i^I) \textbf{ else } P_i^I(x_i^I), \textbf{if } x_{id}^R = id^I \textbf{ then } Q_i^R(x_i^R) \textbf{ else } P_i^R(x_i^R)]$$

B.2 Formal Corollary for Key Confirmations

The Theorem for those key exchanges is very similar to Corollary B.2. The main difference is that now, instead of working on a key exchange $KE := I(lsid^I, id^I)|R(lsid^R, id^R)[$, we further split I and R, in $I = I^0; I^1$ and $R := R^0; R^1$, where I^0 and R^0 will corresponds to the key exchange up to but not including the first use of the secret key, and I^1 and R^1 as the remainder of the protocol.

Corollary B.3. *Let* $\mathcal{O}_{KE}$, $\mathcal{O}_r$,$\mathcal{O}_{P,Q}$ *be oracles and*

$$KE_i[__1, __2] := I_i(lsid_i^I, id^I); __1|R_i(lsid^R i, id^R); __2$$

a key exchange protocol with $I_i(lsid_i^I, id^I) := I_i^0(lsid_i^I, id^I); I_i^1(x^I)$ *and* $R_i(lsid_i^R, id^R) := R_i^0(lsid_i^R, id^R); R_i^1(x^R)$ *such that* I^0 *binds* x^I, x_{id}, x_{lsid}, R^0 *binds* x^R, x_{id}, x_{lsid} *and* $\mathcal{N}_l(KE)$ *is disjoint of the oracles support. Let* $\overline{p} = \{id^I, id^R\}$, $P_i(x, \overline{y}) = P_i^I(x, \overline{y}) \| P_i^R(x, \overline{y})$,$Q(x, \overline{y}, \overline{z}) = Q_i^I(x, \overline{y}, \overline{z}) \| Q_i^R(x, \overline{y}, \overline{z})$, $C_i(\overline{z})$ *and* $D_i(\overline{z})$ *be protocols, such that* $\mathcal{N}_l(P, Q, C, D)$ *is disjoint of the oracles support.*

Let id^I, id^R *be names,* $\overline{s}^I = \{lsid_i^I\}_{i\in\mathbb{N}}, \overline{s}^R = \{lsid_i^R\}_{i\in\mathbb{N}}$ *and* $\overline{s} = \overline{s}^I \cap \overline{s}^R$ *sets of names.*

A-1 $\forall i \in \mathbb{N}, (\nu lsid_i^I, id^I, lsid_i^R, id^R.C_i(\overline{p}) \| I_i^0(lsid_i^I, id^I); \textbf{out}(x^I) \| R_i^0(lsid_i^R, id^R); \textbf{out}(x^R)$ *is* $\mathcal{O}_{KE}$ *simulatable)).*

A-2 $\overline{s}$ *is disjoint of the support of* $\mathcal{O}_p$.

A-3

$$
\begin{array}{ll}
C_i(\overline{p}) \| I_0^0(lsid_0^I, id^I); & \textbf{if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} \\
& \quad I^1(x^I); \textbf{out}(x^I) \\
& \textbf{else out}(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I \rangle) \\
\| R^0(lsid_0^R, id^R); & \textbf{if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id} = id^I \textbf{ then} \\
& \quad R^1(x^R); \textbf{out}(x^R) \\
& \textbf{else out}(\langle x^R, lsid^R, x_{lsid}^R, x_{id}^R \rangle) \\
\cong_{\mathcal{O}_{KE}, \mathcal{O}_p} & \\
C_i(\overline{p}) \| I^0(lsid_0^I, id^I); & \textbf{if } x_{lsid}^I = lsid^R \wedge x_{id} = id^R \textbf{ then} \\
& \quad \textbf{out}(\langle k, lsid_0^I, x_{lsid}^I, x_{id}^I \rangle) \\
& \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} \\
& \quad I^1(x^R); \bot \\
& \textbf{else out}(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I \rangle) \\
\| R^0(lsid_0^R, id^R); & \textbf{if } x_{lsid}^R = lsid^I \wedge x_{id}^R = id^I \textbf{ then} \\
& \quad \textbf{out}(\langle k, lsid_0^R, x_{lsid}^R, x_{id}^R \rangle) \\
& \textbf{else if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} \\
& \quad I^1(x^R); \bot \\
& \textbf{else out}(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R \rangle)
\end{array}
$$

and for any N *which may depend on the security parameter:*

B-1 $\|^{i \le N^2} D_i(\overline{p}) \| I_i^1(k_i); P_i^I(\overline{p}, k_i) \| B_i^1(k_i); P_i^R(\overline{p}, k_i) \cong_{\mathcal{O}_r, \mathcal{O}_k} \|^{i \le n^2} D_i(\overline{p}) \| I_i^1(k_i); Q_i^I(\overline{p}, k_i) \| B_i^1(k_i); Q_i^R(\overline{p}, k_i)$

and

C-1 $\nu \overline{p}, lsid_i^I, lsid_i^R.D_i(\overline{p}) \| in(x).P_i(x) \ \| in(x).Q_i(x) \| in(x).I_i^1(x); P_i^I(x) \| in(x).R_i^1(x); P_i^R(x) \| in(x).I_i^1(x); Q_i^I(x) \| in(x).R_i^1(x); Q_i^R(x)$ *is* $\mathcal{O}_p$ *simulatable.*

C-2 $\nu \overline{p}.$

$$
\begin{array}{lll}
\|^{i \le N} & C_i(\overline{p}) \| I_i^0(lsid_i^I, id^I); & \textbf{if } \underset{1 \le j \le N}{} x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id^R \textbf{ then} \\
& & \quad \textbf{out}(\langle i, j \rangle) \\
& & \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} \\
& & \quad I_i^1(x^I); \bot \\
& & \textbf{else } I_i^1(x^I); P_i^I(x^I) \\
\| R_i^0(lsid_i^R, id^R)[& & \textbf{if } \underset{1 \le j \le N}{} x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id^I \textbf{ then} \\
& & \quad \textbf{out}(\langle i, j \rangle) \\
& & \textbf{else if } (x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} \\
& & \quad R_i^1(x^R); \bot \\
& & \textbf{else } R_i^1(x^R); P_i^R(x^R)
\end{array}
$$

is $\mathcal{O}_k$ *simulatable.*

Then, for any n*:*

$$\|^{i \le N} C_i(\overline{p}) \| D_i(\overline{p}) \| KE_i[P_i^I(x^I), P_i^R(x^R)] \cong$$

$$\|^{i \le N} C_i(\overline{p}) \| D_i(\overline{p}) \| KE_i[\textbf{if } x_{id}^I = id^R \textbf{ then } Q_i^I(x^I) \textbf{ else } P_i^I(x^I), \textbf{if } x_{id}^R = id^I \textbf{ then } Q_i^R(x^R) \textbf{ else } P_i^R(x^R)]$$

B.3 Proofs of Chapter 5

B.3.1 Oracle Simulation

We first show that $\mathcal{O}$-simulation, whose definition implies the identical distributions of two messages produced either by the simulator of by the oracle, implies the equality of distributions of message sequences produced by either the oracle or the simulator.

Lemma 5.2. *Given a functional model $\mathcal{M}^f$, a sequence of names $\overline{n}$, an oracle $\mathcal{O}$ with support $\overline{n}$ and a protocol P, that is $\mathcal{O}$-*simulatable *with $\mathcal{A}^{\mathcal{O}}$, we have, for every $\overline{x}, \overline{y}, c, r_2, r_{\mathcal{B}} \in \{0,1\}^\star$, every $\overline{v} \in D_{\overline{n}}^{\eta}$, for every $m \geq 1$, for every PTOM $\mathcal{B}^{\mathcal{O}}$ (using tags prefixed by 1):*

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta_m^1 = \overline{x}, \phi_m^1 = \overline{y} |\ [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\quad = \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta_m^2 = \overline{x}, \phi_m^2 = \overline{y} |\ [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\end{aligned}$$

where we split $\rho_{\mathcal{O}}$ into $\rho_{\mathcal{O}}^{\mathcal{A}} \uplus \rho_{\mathcal{O}}^{\mathcal{B}}$ such that $\mathcal{O}$ called by $\mathcal{B}$ only accesses $\rho_{\mathcal{O}}^{\mathcal{B}}$ and $\mathcal{O}$ called by $\mathcal{A}$ only accesses $\rho_{\mathcal{O}}^{\mathcal{A}}$ (which is possible thanks to the distinct prefixes).

Proof. We proceed by induction on m. Let us fix $\overline{x}, \overline{y}, c, r_2, r_{\mathcal{B}} \in \{0,1\}^\star$ and $\overline{v} \in D_{\overline{n}}^{\eta}$. We assume that:

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta_m^1 = \overline{x}, \phi_m^1 = \overline{y} |\ [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\quad = \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta_m^2 = \overline{x}, \phi_m^2 = \overline{y} |\ [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\end{aligned}$$

We define $v_{m+1}^i = \mathcal{B}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\mathcal{M}_f, \rho_{r_2}, \eta, \phi_m^i)$.
As the support of $\mathcal{O}$ is $\overline{n}$, we have that $\mathcal{O}(\rho_s, \rho_{\mathcal{O}}) = \mathcal{O}(\pi_{\overline{k}}(\rho_s, \eta), \rho_{\mathcal{O}})$.

Using conditional probabilities, we have that:

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta_{m+1}^1 = \overline{x}, \phi_m^1 = \overline{y} |\ [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\quad = \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{v_{m+1}^1 = x_{m+1} |\ \theta_m^1 = \overline{x}, \phi_m^2 = \overline{y}, [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\qquad \times \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta_m^1 = \overline{x}, \phi_m^1 = \overline{y} |\ [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\end{aligned}$$

Now, if we define $\mathcal{O}_{\overline{v},r_{\mathcal{B}}}$ such that $\mathcal{O}_{\overline{v},r_{\mathcal{B}}} = \mathcal{O}(\pi_{\overline{n}}(\rho_s, \eta), \rho_{\mathcal{O}}^{\mathcal{B}})$ when $[\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}$ and $\rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}$, we have that

$$\begin{aligned}&\mathbb{P}_{\rho_s,,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{v_{m+1}^1 = x_{m+1} |\ \theta_m^1 = \overline{x}, \phi_m^1 = \overline{y}, [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\quad =^1 \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O}(\pi_{\overline{n}}(\rho_s,\eta),\rho_{\mathcal{O}}^{\mathcal{B}})}(\mathcal{M}_f, \rho_{r_2}, \eta, \phi_m^1) = x_{m+1}\\ &\qquad\qquad |\ \theta_m^1 = \overline{x}, \phi_m^1 = \overline{y}, [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\quad =^2 \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O}_{\overline{v},r_{\mathcal{B}}}}(\mathcal{M}_f, r_2, \eta, \overline{y}) = x_{m+1}\\ &\qquad\qquad |\ \theta_m^1 = \overline{x}, \phi_m^1 = \overline{y}, [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\quad =^3 \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O}_{\overline{v},r_{\mathcal{B}}}}(\mathcal{M}_f, r_2, \eta, \overline{y}) = x_{m+1}\}\\ &\quad =^4 \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O}_{\overline{v},r_{\mathcal{B}}}}(\mathcal{M}_f, r_2, \eta, \overline{y}) = x_{m+1}\\ &\qquad\qquad |\ \theta_m^2 = \overline{x}, \phi_m^2 = \overline{y}, [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\quad =^5 \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{B}^{\mathcal{O}(\pi_{\overline{n}}(\rho_s,\eta),\rho_{\mathcal{O}}^{\mathcal{B}})}(\mathcal{M}_f, \rho_{r_2}, \eta, \phi_m^2) = x_{m+1}\\ &\qquad\qquad |\ \theta_m^2 = \overline{x}, \phi_m^2 = \overline{y}, [\![\overline{n}]\!]_{\rho_s}^{\eta} = \overline{v}, \rho_{\mathcal{O}}^{\mathcal{B}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\end{aligned}$$

Justified with:

1. because $\mathcal{O}(\rho_s, \rho_{\mathcal{O}}) = \mathcal{O}(\pi_{\overline{n}}(\rho_s, \eta), \rho_{\mathcal{O}}^{B})$;
2. $\mathcal{O}(\pi_{\overline{n}}(\rho_s, \eta), \rho_{\mathcal{O}}^{B}) = \mathcal{O}_{\overline{v},r_{\mathcal{B}}}$, and $\phi_m^1 = \overline{y}$;
3. the considered event does not depends on any of the conditional events removed;
4. the considered event does not depends on any of the conditional events added;

5. reversing the previous steps.

So we conclude that, as we also have the induction hypothesis:

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^1_{m+1}=\overline{x},\phi^1_m=\overline{y}|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\ =\quad&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^2_{m+1}=\overline{x},\phi^2_m=\overline{y}|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\quad(i)\end{aligned}$$

We now define:

$$u^1_{m+1}=\mathcal{A}^{\mathcal{O}(\pi_{\overline{n}}(\rho_s,\eta),\rho^{\mathcal{A}}_{\mathcal{O}})}(\mathcal{M}_f,\rho_{r_1},\theta^1_m\uplus v^1_{m+1},\eta)$$

$$u^2_{m+1}=\mathcal{O}_P(\rho_s,\theta^2_m\uplus v^2_{m+1})$$

We define the Turing machine $\underline{\mathcal{B}}$, such that:

$$\begin{aligned}&\underline{\mathcal{B}}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\mathcal{M}_f,\rho_{r_2},\eta,\phi^i_m):=\\&\quad\textbf{if}\ \forall j\leq m+1,\mathcal{B}^{\mathcal{O}(\overline{v},r_{\mathcal{B}})}(\mathcal{M}_f,r_2,\eta,\phi^i_j)=x_j\\&\qquad\wedge\phi^i_m=\overline{y}\\&\quad\textbf{then}\ \mathcal{B}^{\mathcal{O}(\overline{v},r_{\mathcal{B}})}(\mathcal{M}_f,r_2,\eta,\phi^i_m)\\&\quad\textbf{else}\ \bot\end{aligned}$$

We then define v'_m and θ'_m for $\underline{\mathcal{B}}$ similarly as v_m for $\mathcal{B}$.

We define $\mathcal{O}_{\overline{v},\rho^{\mathcal{A}}_{\mathcal{O}}}$ such that $\mathcal{O}_{\overline{v},\rho^{\mathcal{A}}_{\mathcal{O}}}=\mathcal{O}(\pi_{\overline{n}}(\rho_s,\eta),\rho^{\mathcal{A}}_{\mathcal{O}})$ when $[\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}$. We then have:

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{u^1_{m+1}=y_{m+1}|\ \theta^1_{m+1}=\overline{x},\phi^1_m=\overline{y},[\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\&=^1\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}_{\overline{v},\rho^{\mathcal{A}}_{\mathcal{O}}}}(\mathcal{M}_f,\rho_{r_1},\overline{x},\eta)=y_{m+1}|\ \theta^1_{m+1}=\overline{x},\phi^1_m=\overline{y},\\&\qquad[\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\&=^2\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}_{\overline{v},\rho^{\mathcal{A}}_{\mathcal{O}}}}(\mathcal{M}_f,\rho_{r_1},\theta^{1'}_{m+1},\eta)=y_{m+1}|\ \theta^1_{m+1}=\overline{x},\phi^1_m=\overline{y},\\&\qquad[\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\&=^3\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}_{\overline{v},\rho^{\mathcal{A}}_{\mathcal{O}}}}(\mathcal{M}_f,\rho_{r_1},\theta^{1'}_{m+1},\eta)=y_{m+1}|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}\}\\&\quad\times(\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^1_{m+1}=\overline{x},\phi^1_m=\overline{y}|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\&\qquad\times\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}\})^{-1}\\&=^4\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{O}_P(\rho_s,\theta^{2'}_{m+1})=y_{m+1}|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}\}\\&\quad\times(\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^1_{m+1}=\overline{x},\phi^1_m=\overline{y},|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\&\qquad\times\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}\})^{-1}\\&=^5\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{O}_P(\rho_s,\theta^{2'}_{m+1})=y_{m+1}|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}\}\\&\quad\times(\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^2_{m+1}=\overline{x},\phi^2_m=\overline{y},|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\&\qquad\times\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2|\ [\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v}\})^{-1}\\&=^6\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{O}_P(\rho_s,\theta^{2'}_{m+1})=y_{m+1}|\ \theta^2_{m+1}=\overline{x},\phi^2_m=\overline{y},[\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\\&\qquad\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\&=^7\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{O}_P(\rho_s,\overline{x})=y_{m+1}|\ \theta^2_{m+1}=\overline{x},\phi^2_m=\overline{y},[\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\\&\qquad\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\\&=^8\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{u^2_{m+1}=y_{m+1}|\ \theta^2_{m+1}=\overline{x},\phi^2_m=\overline{y},[\![\overline{n}]\!]^{\eta}_{\rho_s}=\overline{v},\\&\qquad\rho^{\mathcal{B}}_{\mathcal{O}}=r_{\mathcal{B}},\rho_{r_2}=r_2\}\end{aligned}$$

Justified with:

1. using the conditional probabilities;
2. by definition of $\underline{\mathcal{B}}$ which produces $\overline{x}$ under the conditional events;
3. using conditional probabilities, as $\theta_m\neq\overline{x}\vee\phi_m\neq\overline{y}\Rightarrow\underline{\mathcal{B}}=\bot$;
4. by $\mathcal{O}$ simulatability on $\underline{\mathcal{B}}$;
5. using (i);

6. using conditional probabilities, as $\theta_m \neq \overline{x} \vee \phi_m \neq \overline{y} \Rightarrow \underline{\mathcal{B}} = \bot$;
7. by definition of $\underline{\mathcal{B}}$ which produces $\overline{x}$ under the conditional events;
8. using the conditional probabilities.

Combining the previous equality with equation (i) finally yields through conditional probabilities:

$$\begin{aligned}&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^1_{m+1} = \overline{x}, \phi^1_{m+1} = \overline{y} |\ [\![\overline{n}]\!]^{\eta}_{\rho_s} = \overline{v}, \rho^{\mathcal{B}}_{\mathcal{O}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\\ &\quad = \quad \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\theta^2_{m+1} = \overline{x}, \phi^2_{m+1} = \overline{y} |\ [\![\overline{n}]\!]^{\eta}_{\rho_s} = \overline{v}, \rho^{\mathcal{B}}_{\mathcal{O}} = r_{\mathcal{B}}, \rho_{r_2} = r_2\}\end{aligned}$$

■

B.3.2 Autocomposition Results

Proposition 5.4. *Let $\mathcal{O}$ be an oracle, two parameterized processes $P(x), Q(x)$, a set of names $\overline{n} = \mathcal{N}_g(P,Q)$ and fresh names k_0, l. We assume that $\mathcal{N}_l(P,Q)$ is disjoint of the support of $\mathcal{O}$. If:*

- $\nu\overline{n}.\mathbf{in}(c_P, x); P(x) \| \mathbf{in}(c_Q, x); Q(x)$ *is $\mathcal{O}$-simulatable, and*
- $P(k_0); \mathbf{out}(c_P, x) \| Q(k_0); \mathbf{out}(c_Q, x) \cong_{\mathcal{O}} P(k_0); \mathbf{out}(c_P, l) \| Q(k_0); \mathbf{out}(c_Q, l)$

then, for any N,

$$\begin{aligned}&P(k_0); P(x)^{;N}; \mathbf{out}(c_P, x) \| Q(k_0); Q(x)^{;N}; \mathbf{out}(c_Q, x)\\ &\quad \cong_{\mathcal{O}} P(k_0); P(x)^{;N}; \mathbf{out}(c_P, l) \| Q(k_0); Q(x)^{;N}; \mathbf{out}(c_Q, l)\end{aligned}$$

Proof. We proceed by induction on N. The result is exactly the first hypothesis for $N = 0$.

Given some $N > 1$, we assume that

$$P(k_i)^{;N-1}; out(k) \| Q(k_i)^{;N-1}; out(k) \cong_{\mathcal{O}} P(k_i)^{;N-1}; out(l) | Q(k_i)^{;N-1}; out(l) \quad (i)$$

In the following, we will write $P(k_i)^{;N-1}$ for $P(k_i); P(k)^{;N-2}$ and we will omit to mention the α-renaming made over the local names in $\mathcal{N}_l(P,Q)$ between the different copies of P and Q. The renaming is however essential so that we may for instance have $\mathcal{N}_l(P^{N-1}(k)) \cap \mathcal{N}_l(P) = \emptyset$ when we wish to apply Theorem 5.4. This silent renaming is possible because $\mathcal{N}_l(P,Q)$ is not contained in the support of $\mathcal{O}$.

We obtain by application of Theorem 5.4 with $A = P(k_i)^{;N-1}$, $B = Q(k_i)^{;N-1}$, $P_1(x) := P(x)^{;0}; out(k)$ and $P_2(x) := Q(x)^{;0}; out(k)$:

$$P^{;N}(k_i); out(k) \| Q^{;N}(k_i); out(k) \cong_{\mathcal{O}} P^{;N-1}(k_i); P(l)^{;0}; out(k) \| Q^{;N-1}(k_i); Q(l)^{;0}; out(k) \quad (I)$$

Now, with Theorem 5.2 applied on $P(l)^{;0}; out(k) \| Q(l)^{;0}; out(k) \cong_{\mathcal{O}} P(l)^{;0}; out(l') \| Q(l)^{;0}; out(l')$ with l' a fresh name, with $P := P(k_i)^{;N-1}$ and $Q := Q(k_i)^{;N-1}$, we obtain:

$$P(k_i)^{;N-1}; P(l)^{;0}; out(k) \| Q^{;N-1}(k_i); Q(l); out(k) \cong_{\mathcal{O}} P(k_i)^{;N-1}; P(l)^{;0}; out(l') \| Q(k_i)^{;N-1}; Q(l)^{;0}; out(l') \quad (II)$$

We also perform an application of Theorem 5.4 on (i) with $A = P(k_i)^{;N-1}$, $B = Q(k_i)^{;N-1}$, $P_1(k) := P(k_i)^{;0}; out(l)$ and $P_2(k) := Q(k_i)^{;0}; out(l)$:

$$P(k_i)^{;N-1}; P(l)^{;0}; out(l') | Q(k_i)^{;N-1}; P(l)^{;0}; out(l') \cong_{\mathcal{O}} P(k_i)^{;N}; out(l) \| Q(k_i)^{;N}; out(l) \quad (III)$$

We conclude by transitivity with (I),(II) and (III). ■

Simulatability is stable by binding names that do not appear in the protocol, which means that we will be able simulate at the same times two simulatable protocol who do not share long term secret.

Lemma B.1. *Given a functional model $\mathcal{M}_f$, a sequence of names $\overline{n}$, an oracle $\mathcal{O}$ with support $\overline{n}$ and a sequence of terms $\overline{t}$, if $\nu\overline{n}.\overline{t}$ is $\mathcal{O}$-*simulatable *, then for any sequence of names $\overline{m}$ such that $\overline{m} \cap \mathcal{N}(t_1, \ldots, t_n) = \emptyset$, $\nu\overline{n} \cup \overline{m}.\overline{t}$ is $\mathcal{O}$-*simulatable*.*

Proof. Let there be a functional model $\mathcal{M}_f$, a sequence of names $\overline{n}$, an oracle $\mathcal{O}$ with support $\overline{n}$ and a sequence of terms $\overline{t}$ *$\mathcal{O}$-simulatable*. As the names of $\overline{m}$ do not appear in $\overline{t}$, the probability of any event regarding $\overline{t}$ is independent from an event regarding $\overline{m}$ so we have for any PTOM $\mathcal{A}^{\mathcal{O}}$, η, sequences $\overline{c}, \overline{v}, \overline{w} \in \{0,1\}^*$,

$$\begin{aligned}
&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\mathcal{M}_f, m_1, \ldots, m_k, \rho_{r_2}, \eta) = \overline{c} \mid [\![\overline{n}]\!]^{\eta}_{\rho_s} = \overline{v}, [\![\overline{m}]\!]^{\eta}_{\rho_s} = \overline{w}\}\\
&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_{\mathcal{O}}}\{\mathcal{A}^{\mathcal{O}(\rho_s,\rho_{\mathcal{O}})}(\mathcal{M}_f, m_1, \ldots, m_k, \rho_{r_2}, \eta) = \overline{c} \mid [\![\overline{n}]\!]^{\eta}_{\rho_s} = \overline{v}\}\\
&\quad = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{[\![t_1, \ldots, t_n]\!]^{\eta}_{\rho_s,\rho_r,\rho_{\mathcal{O}}} = \overline{c} | [\![\overline{n}]\!]^{\eta}_{\rho_s} = \overline{v}\}\\
&\quad = \mathbb{P}_{\rho_s,\rho_r,\rho_{\mathcal{O}}}\{[\![t_1, \ldots, t_n]\!]^{\eta}_{\rho_s,\rho_r,\rho_{\mathcal{O}}} = \overline{c} | [\![\overline{n}]\!]^{\eta}_{\rho_s} = \overline{v}, [\![\overline{m}]\!]^{\eta}_{\rho_s} = \overline{w}\}
\end{aligned}$$

Thus $\nu\overline{n} \cup \overline{m}.\overline{t}$ is *$\mathcal{O}$-simulatable*. ■

Proposition 5.3. *Let $\mathcal{O}_r$ be an oracle parameterized by a sequence of names $\overline{s}$, and $\mathcal{O}$ an oracle. Let $\overline{p}$ be a sequence of names, $P(\overline{x})$, $R_i^1(\overline{x}, \overline{y}), \ldots, R_i^k(\overline{x}, \overline{y})$ and $Q(\overline{x})$ be protocols, such that $\mathcal{N}_l(R_i^1, \ldots, R_i^k)$ is disjoint of the oracle support. If we have, for sequences of names $\overline{lsid}^1, \ldots, \overline{lsid}^k$, with $\overline{s} = \{\overline{lsid}_i^j\}_{1 \leq j \leq k, i \in \mathbb{N}}$:*

1. *$\forall i, j \in \mathbb{N}, \nu\overline{p}, \overline{lsid}_i^j.R_i^j(\overline{p}, \overline{lsid}_i^j)$ is $\mathcal{O}_r$-simulatable.*
2. *$P(\overline{p}) \cong_{\mathcal{O}_r} Q(\overline{p})$*
3. *$\overline{s}$ is disjoint of the support of $\mathcal{O}$.*

Then, for any integers $N_1, \ldots, N_k$:

$$\begin{aligned}
&P(\overline{p}) \| ^{i \leq N_1} (R_i^1(\overline{p}, \overline{lsid}_i^1) \| \ldots \| ^{i \leq N_k} R_i^k(\overline{p}, \overline{lsid}_i^k)\\
&\quad \cong_{\mathcal{O}, \mathcal{O}_r} Q(\overline{p}) \| ^{i \leq N_1} R_i^1(\overline{p}, \overline{lsid}_i^1) \| \ldots \| ^{i \leq N_k} R_i^k(\overline{p}, \overline{lsid}_i^k)
\end{aligned}$$

Specifically, there exists a polynomial p_S (independent of all R^j) such that if p_{R^j} is the polynomial bound on the runtime of the simulator for R^j, we have,

$$\begin{aligned}
&\mathrm{Adv}^{P(\overline{p}) \| ^{i \leq N_1} (R_i^1(\overline{p}, \overline{lsid}_i^1) \| \ldots \| ^{i \leq N_k} R_i^k(\overline{p}, \overline{lsid}_i^k) \cong_{\mathcal{O}} Q(\overline{p}) \| ^{i \leq N_1} R_i^1(\overline{p}, \overline{lsid}_i^1) \| \ldots \| ^{i \leq N_k} R_i^k(\overline{p}, \overline{lsid}_i^k)}(t)\\
&\quad \leq \mathrm{Adv}^{P(\overline{p}) \cong_{\mathcal{O}, \mathcal{O}_r} Q(\overline{p})}\Big(p_S\big(t, N_1, |R^1|, \ldots, N_k, |R^k|, p_{R^1}(t), \ldots, p_{R^k}(t)\big)\Big)
\end{aligned}$$

Rather than proving the previous Theorem, where we recall that the protocols may depend on a predicate $T(x)$ whose interpretation depends on $\overline{s}$, we prove the version where P directly depends on $\overline{s}$.

Proposition B.1. *Let $\mathcal{O}_r$ be an oracle parameterized by a sequence of names $\overline{s}$. Let $\overline{p}$ be a sequence of names, $P(\overline{x})$, $R_i^1(\overline{x}, \overline{y}, z), \ldots, R_i^k(\overline{x}, \overline{y}, z)$ and $Q(\overline{x}, y)$ be protocols, such that $\mathcal{N}_l(R_i^1, \ldots, R_i^k)$ is disjoint of the oracle support. If we have, for sequences of names $\overline{lsid}^1, \ldots, \overline{lsid}^k$, with $\overline{s} = \{\overline{lsid}_i^j\}_{i,j \in \mathbb{N}}$:*

1. *$\forall i, j \in \mathbb{N}, \nu\overline{p}, \overline{lsid}_i^j.R_i^j(\overline{p}, \overline{lsid}_i^j, \overline{s})$ is $\mathcal{O}_r$-simulatable.*
2. *$P(\overline{p}) \cong_{\mathcal{O}} Q(\overline{p}, \overline{s})$*

Then, for any integers $N_1, \dots, N_k$:

$$P(\overline{p})\|^{i\leq N_1}(R_i^1(\overline{p},\overline{lsid_i}^1,\overline{s})\| \dots \|^{i\leq N_k} R_i^k(\overline{p},\overline{lsid_i}^k,\overline{s}) \\ \cong_{\mathcal{O}_r} Q(\overline{p},\overline{s})\|^{i\leq N_1} R_i^1(\overline{p},\overline{lsid_i}^1,\overline{s})\| \dots \|^{i\leq N_k} R_i^k(\overline{p},\overline{lsid_i}^k,\overline{s})$$

Specifically, there exists polynomial p_S (independent of all R^j)such that if p_{R^j} is the polynomial bound on the runtime of the simulator for R^j, we have,

$$\mathrm{Adv}^{P(\overline{p})\|^{i\leq N_1}(R_i^1(\overline{p},\overline{lsid_i}^1,\overline{s})\|\dots\|^{i\leq N_k}R_i^k(\overline{p},\overline{lsid_i}^k,\overline{s})\cong_{\mathcal{O}_r}Q(\overline{p},\overline{s})\|^{i\leq N_1}R_i^1(\overline{p},\overline{lsid_i}^1,\overline{s})\|\dots\|^{i\leq N_k}R_i^k(\overline{p},\overline{lsid_i}^k,\overline{s})}(t) \\ \leq \mathrm{Adv}^{P(\overline{p})\cong_{\mathcal{O}}Q(\overline{p},\overline{s})}\Big(p_S\big(t,N_1,|R^1|,\dots,N_k,|R^k|,p_{R^1}(t),\dots,p_{R^k}(t)\big)\Big)$$

Proof. We prove the result for $k=1$, denoting R^1 as R, as the generalization is immediate. Let there be an integer n.

Hypothesis 1 with Lemma B.1 gives us that for $1 \leq i \leq N$, $\nu lsid_i, \overline{p}.R_i(\overline{p}, lsid_i, \overline{s})$ is $\mathcal{O}_R$-simulatable.

Moreover, with $\delta = \{\overline{p}, \overline{s}\}$, $\mathcal{N}(R_i(\overline{p}, lsid_i, \overline{s})) \cap \delta = \{\overline{p}, lsid_i\}$, so thanks to Theorem 5.1, for $1 \leq i \leq N$, $\nu\delta.R(\overline{p}, lsid_i, \overline{s})$ is $\mathcal{O}_R$-simulatable.

Now, up to renaming of the local names of R (which is possible as they do not appear in the oracle support), we have that $\forall 1 \leq i < j \leq N. \mathcal{N}(R_i(\overline{p}, lsid_i, \overline{s})) \cap \mathcal{N}(R_j(\overline{p}, lsid_j, \overline{s})) \subset \delta$, so with Theorem 5.1 we have that $\|^{i\leq N} R_i(\overline{p}, lsid_i, \overline{s})$ is $\mathcal{O}_R$-simulatable.

Note that if R is simulatable by a simulator bounded by a polynomial $p_R(t)$ on an input of size t, then $\|i \leq N R(\overline{p}, lsid_i, \overline{s})$ is simulatable by a simulator bounded by a polynomial $q(n, p_R(t))$, where q is uniform in n and R.

Finally, we have that $\|^{i\leq N} R_i(\overline{p}, \overline{lsid_i}, \overline{s})$ is $\mathcal{O}_R$-simulatable and $P(\overline{p}, lsid_n) \cong_{\mathcal{O}} Q(\overline{p}, \overline{s})$, so we conclude with Theorem 5.2.

Instantiating the bound on the advantage from Theorem 5.2 with $|C| = n|R|$ and $p_C(t) = q(n, p_R(t))$ yields the desired result. ∎

Theorem 5.5. *Let $\mathcal{O}_r$, $\mathcal{O}$ be oracles both parameterized by a sequence of names $\overline{s}$. Let $\overline{p}$ be a sequence of names, $P_i(\overline{x}, \overline{y})$ and $Q_i(\overline{x}, \overline{y})$ be parameterized protocols, such that $\mathcal{N}_l(P, Q)$ is disjoint of the oracles support. If we have, for sequences of names $\overline{lsid}^P, \overline{lsid}^Q$, with $\overline{s} = \{\overline{lsid_i}^P, \overline{lsid_i}^Q\}_{i\in\mathbb{N}}$:*

1. *$\forall\, i \geq 1, \nu\overline{p}, \overline{lsid_i}^P.P_i(\overline{p}, \overline{lsid_i}^P)$ is $\mathcal{O}_r$-simulatable.*
2. *$\forall\, i \geq 1, \nu\overline{p}, \overline{lsid_i}^Q.Q_i(\overline{p}, \overline{lsid_i}^Q)$ is $\mathcal{O}_r$-simulatable.*
3. *$\overline{s}$ is disjoint of the support of $\mathcal{O}$.*
4. *$P_0(\overline{p}, \overline{lsid_0}^P) \cong_{\mathcal{O}_r,\mathcal{O}} Q_0(\overline{p}, \overline{lsid_0}^Q)$*

then,

$$||^i P_i(\overline{p}, \overline{lsid_i}^P) \cong_{\mathcal{O}} ||^i Q_i(\overline{p}, \overline{lsid_i}^Q)$$

We once again generalize with the explicit dependence in $\overline{s}$.

Theorem B.1. *Let $\mathcal{O}_r$, $\mathcal{O}$ be oracles both parameterized by a sequence of names $\overline{s}$. Let $\overline{p}$ be a sequence of names, $P_i(\overline{x}, \overline{y})$ and $Q_i(\overline{x}, \overline{y}, z)$ be parameterized protocols, such that $\mathcal{N}_l(P, Q)$ is disjoint of the oracles support. If we have, for sequences of names $\overline{lsid}^P, \overline{lsid}^Q$, with $\overline{s} = \{\overline{lsid_i}^P, \overline{lsid_i}^Q\}_{i\in\mathbb{N}}$:*

1. $\forall\ i \geq 1, \nu\overline{p}, \overline{lsid}_i^P.P_i(\overline{p}, \overline{lsid}_i^P)$ *is* $\mathcal{O}_r$*-simulatable.*
2. $\forall\ i \geq 1, \nu\overline{p}, \overline{lsid}_i^Q.Q_i(\overline{p}, \overline{lsid}_i^Q, \overline{s})$ *is* $\mathcal{O}_r$*-simulatable.*
3. $\overline{s}$ *is disjoint of the support of* $\mathcal{O}$*.*
4. $P_0(\overline{p}, \overline{lsid}_0^P) \cong_{\mathcal{O}_r,\mathcal{O}} Q_0(\overline{p}, \overline{lsid}_0^Q, \overline{s})$

then, $||^i P_i(\overline{p}, \overline{lsid}_i^P) \cong_{\mathcal{O}} ||^i Q_i(\overline{p}, \overline{lsid}_i^Q, \overline{s})$

Proof. By application of Theorem 5.5, we get that for all n_1, n_2,

$$\begin{gathered} P_0(\overline{p}, \overline{lsid}_0^P) \| ^{1<i\leq N_1} P_i(\overline{p}, \overline{lsid}_i^P) \| ^{1<i\leq N_2} Q_i(\overline{p}, \overline{s}, \overline{lsid}_i^Q) \\ \cong_{\mathcal{O}_r,\mathcal{O}} \\ Q_0(\overline{p}, \overline{s}, \overline{lsid}_0^Q) \| ^{1<i\leq N_1} P_i(\overline{p}, \overline{lsid}_i^P) \| ^{1<i\leq N_2} Q_i(\overline{p}, \overline{s}, \overline{lsid}_i^Q) \end{gathered}$$

By weakening of the attacker, we get:

$$\begin{gathered} P_0(\overline{p}, \overline{lsid}_0^P) \| ^{1<i\leq N_1} P_i(\overline{p}, \overline{lsid}_i^P) \| ^{1<i\leq N_2} Q_i(\overline{p}, \overline{s}, \overline{lsid}_i^Q) \\ \cong_{\mathcal{O}} \\ Q_0(\overline{p}, \overline{s}, \overline{lsid}_0^Q) \| ^{1<i\leq N_1} P_i(\overline{p}, \overline{lsid}_i^P) \| ^{1<i\leq N_2} Q_i(\overline{p}, \overline{s}, \overline{lsid}_i^Q) \end{gathered}$$

Then, for a polynomial p (assumed without loss of generality increasing), any $n = p(\eta)$, and all $j < n$:

$$\begin{gathered} P_0(\overline{p}, \overline{lsid}_0^P) \| ^{1<i\leq j-1} P_i(\overline{p}, \overline{lsid}_i^P) \| ^{1<i\leq N-j-1} Q_i(\overline{p}, \overline{s}, \overline{lsid}_i^Q) \\ \cong_{\mathcal{O}} \\ Q_0(\overline{p}, \overline{s}, \overline{lsid}_0^Q) \| ^{1<i\leq j-1} P_i(\overline{p}, \overline{lsid}_i^P) \| ^{1<i\leq N-j-1} Q_i(\overline{p}, \overline{s}, \overline{lsid}_i^Q) \end{gathered}$$

Through the renaming of the $\overline{lsid}$, which is possible as $\overline{s}$ is disjoint from the oracle support, we get that:

$$\begin{gathered} P_j(\overline{p}, \overline{lsid}_j^P) \| P_0(\overline{p}, \overline{lsid}_0^P) \dots \| P_{j-1}(\overline{p}, \overline{lsid}_{j-1}^P) \| Q_{j+1}(\overline{p}, \overline{s}, \overline{lsid}_{j+1}^Q) \| \dots Q_n(\overline{p}, \overline{s}, \overline{lsid}_n^Q) \\ \cong_{\mathcal{O}} \\ Q_j(\overline{p}, \overline{lsid}_j^Q, \overline{s}) \| P_0(\overline{p}, \overline{lsid}_0^P) \dots \| P_{j-1}(\overline{p}, \overline{lsid}_{j-1}^P) \| Q_{j+1}(\overline{p}, \overline{s}, \overline{lsid}_{j+1}^Q) \| \dots Q_n(\overline{p}, \overline{s}, \overline{lsid}_n^Q) \end{gathered}$$

Thanks to Theorem 5.5, there exist polynomial p_S such that, if p_P and p_Q are the polynomial bound on the runtime of the simulators for P or Q, for all j, we have that the advantage of any attacker running in time t against the previous indistinguishability, denoted D, is bounded by:

$$\mathrm{Adv}^D\Big(p_S\big(t, j-1, |P|, \dots, p(\eta)-j-1, |q|, p_P(t), \dots, p_Q(t)\big)\Big)$$

Thus, for all j, the advantage of any attacker against the corresponding game is uniformly bounded by:

$$\mathrm{Adv}^D\Big(p_S\big(t, p(\eta), |P|, \dots, p(\eta), |q|, p_P(t), \dots, p_Q(t)\big)\Big)$$

We then conclude with an hybrid argument.

■

We first prove a proposition which allows to reduce the security of n sessions in parallel to the security of one session with $N-1$ sessions in parallel. It is expressed in a more general way than required for basic key exchanges, so that we can reuse it for other results.

Proposition B.2. *Let $\mathcal{O}$ be an oracle and $KE_i[_1,_2] := I_i(lsid_i^I, id^I);_1 \| R_i(lsid_i^R, id^R);_2$ a key exchange protocol, such that I binds $x^I, x_{id}^I, x_{lsid}^I$, R binds $x^R, x_{id}^R, x_{lsid}^R$ and $\mathcal{N}_l(KE)$ is disjoint of the oracle support. Let id^I, id^R be names, $\overline{s}^I = \{lsid_i^I\}_{i\in\mathbb{N}}$, $\overline{s}^R = \{lsid_i^R\}_{i\in\mathbb{N}}$, $\overline{s} = \overline{s}^I \cup \overline{s}^R$ sets of names,*

Let $T_1(\overline{x})$,$T_2(\overline{x})$, $S_1(\overline{x})$,$S_2(\overline{x})$ be parametric processes with completely disjoint names. Let N be an integer (which may depend on η), and let $\overline{s} = \{lsid_i^I, lsid_i^R\}_{1\le i\le N}$ and $\mathcal{O}_{\overline{s}}$ an oracle. If $\overline{s}$ is disjoint of the support of $\mathcal{O}$ and if,

1. *$\nu\overline{s}.\mathbf{out}(\overline{s})$ is $\mathcal{O}_{\overline{s}}$-simulatable.*

$$\begin{array}{l}
\|^{i\le N-1} KE_i[\mathbf{out}(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I\rangle), \mathbf{out}(\langle x^R, lsid^R, x_{lsid}^R, x_{id}^R\rangle)] \\
\quad \|KE_n[\ \ \mathbf{if}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad S1(x^I, lsid^I, x_{lsid}^I, x_{id}^I) \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I\rangle), \\
\qquad \mathbf{if}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad S_2(x^R, lsid^R, x_{lsid}^R, x_{id}^R) \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^R, lsid^R, x_{lsid}^R, x_{id}^R\rangle)] \\
\quad \|^{i\le N-1} KE_i[\mathbf{out}(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I\rangle), \mathbf{out}(\langle x^R, lsid^R, x_{lsid}^R, x_{id}^R\rangle)]
\end{array} \cong_{\mathcal{O},\mathcal{O}_{\overline{s}}}$$

2.
$$\begin{array}{l}
\|\ \ KE_n[\ \ \mathbf{if}\ x_{lsid}^I = lsid_n^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad \mathbf{out}(\langle k, lsid_n^I, x_{lsid}, x_{id}\rangle) \\
\qquad \mathbf{if}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad T_1(x^I, lsid_n^I, x_{lsid}, x_{id}) \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I\rangle), \\
\qquad \mathbf{if}\ x_{lsid}^R = lsid_n^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad \mathbf{out}(\langle k, lsid_n^R, x_{lsid}, x_{id}\rangle) \\
\qquad \mathbf{if}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad T_2(x^R, lsid_n^R, x_{lsid}, x_{id}) \\
\qquad \mathbf{else}\ \mathbf{out}(x^R, lsid^R, x_{lsid}^R, x_{id}^R)
\end{array}$$

Then:

$$\begin{array}{l}
\|^{i\le N} KE_i[\ \ \mathbf{if}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad S1(x^I, lsid^I, x_{lsid}^I, x_{id}^I) \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I\rangle), \\
\qquad \mathbf{if}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad S_2(x^R, lsid^R, x_{lsid}^R, x_{id}^R) \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^R, lsid^R, x_{lsid}^R, x_{id}^R\rangle)] \\
\cong_{\mathcal{O}} \\
\quad \|^{i\le N}\ KE_i[\ \underset{1\le j\le N}{\mathbf{if}}\ x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id_R\ \mathbf{then} \\
\qquad \mathbf{out}(\langle k_{i,j}, lsid_i^I, x_{lsid}, x_{id}\rangle) \\
\qquad \mathbf{if}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad T_1(x^I, lsid_n^I, x_{lsid}, x_{id}) \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I\rangle), \\
\qquad \underset{1\le j\le N}{\mathbf{if}}\ x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id^I)\ \mathbf{then} \\
\qquad \mathbf{out}(\langle k_{j,i}, lsid_i^R, x_{lsid}, x_{id}\rangle) \\
\qquad \mathbf{if}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad T_2(x^R, lsid_n^R, x_{lsid}, x_{id}) \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^R, lsid^R, x_{lsid}^R, x_{id}^R\rangle)]
\end{array}$$

Proof. We fix N and define an ordering (arbitrary) on the couples $(i,j)_{1\le i,j\le N}$. We then set:

$$
\begin{aligned}
G^0_{(i,j)} := \\
\|_{r\leq N} KE_r[\ & \textbf{if}_{(r,t)\geq(i,j)}\ x^I_{lsid} = lsid^R_t \wedge x^I_{id} = id_R \textbf{ then} \\
& \quad \textbf{out}(\langle k_{r,t}, lsid^I_r, x^I_{lsid}, x^I_{id}\rangle) \\
& \textbf{if}_{(r,t)\geq(i,j)}\ x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
& \quad T_1(x^I, lsid^I_r, x^I_{lsid}, x^I_{id}) \\
& \textbf{else if } x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
& \quad S1(x^I, lsid^I, x^I_{lsid}, x^I_{id}) \\
& \textbf{else out}(\langle x^I, lsid^I, x^I_{lsid}, x^I_{id}\rangle), \\
& \\
& \textbf{if}_{(t,r)\geq(i,j)}\ x^R_{lsid} = lsid^I_t \wedge x^R_{id} = id^I) \textbf{ then} \\
& \quad \textbf{out}(\langle k_{t,r}, lsid^R_r, x^R_{lsid}, x^R_{id}) \\
& \textbf{if}_{(t,r)\geq(i,j)}\ x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
& \quad T_2(x^R, lsid^R_r, x^R_{lsid}, x^R_{id}) \\
& \textbf{else if } x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
& \quad S_2(x^R, lsid^R, x^R_{lsid}, x^R_{id}) \\
& \textbf{else out}(\langle x^R, lsid^R, x^R_{lsid}, x^R_{id}\rangle)]
\end{aligned}
$$

and

$$
\begin{aligned}
G^1_{(i,j)} := \\
\|_{r\leq N} KE_r[\ & \textbf{if}_{(r,t)>(i,j)}\ x^I_{lsid} = lsid^R_t \wedge x^I_{id} = id_R \textbf{ then} \\
& \quad \textbf{out}(\langle k_{r,t}, lsid^I_r, x^I_{lsid}, x^I_{id}\rangle) \\
& \textbf{if}_{(r,t)>(i,j)}\ x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
& \quad T_1(x^I, lsid^I_r, x^I_{lsid}, x^I_{id}) \\
& \textbf{else if } x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
& \quad S1(x^I, lsid^I, x^I_{lsid}, x^I_{id}) \\
& \textbf{else out}(\langle x^I, lsid^I, x^I_{lsid}, x^I_{id}\rangle), \\
& \\
& \textbf{if}_{(t,r)>(i,j)}\ x^R_{lsid} = lsid^I_t \wedge x^R_{id} = id^I) \textbf{ then} \\
& \quad \textbf{out}(\langle k_{t,r}, lsid^R_r, x^R_{lsid}, x^R_{id}) \\
& \textbf{if}_{(t,r)>(i,j)}\ x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
& \quad T_2(x^R, lsid^R_r, x^R_{lsid}, x^R_{id}) \\
& \textbf{else if } x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
& \quad S_2(x^R, lsid^R, x^R_{lsid}, x^R_{id}) \\
& \textbf{else out}(\langle x^R, lsid^R, x^R_{lsid}, x^R_{id}\rangle)]
\end{aligned}
$$

We note that $G^1_{(i,j)} = G^0_{(i,j)+1}$, that $G^0_{(0,0)}$ is the game on the right hand side of the goal, and that $G^0_{(n,n)}$ is the game on the left hand side of the goal.

Thus, if we have uniformly that $G^1_{(i,j)} \cong G^0_{(i,j)}$, we can conclude with a classical hybrid argument.

We remark that $G^1_{(i,j)}$ and $G^0_{(i,j)}$ only differ in two places, where a conditional is added in I_i and one in R_j.

Let us fix (i,j), we define the substitution $\sigma := \{lsid^I_n \mapsto lsid^I_i, lsid^R_N \mapsto lsid^R_j, lsid^I_i \mapsto lsid^I_n, lsid^R_j \mapsto lsid^R_n\}$ and denote $\overline{s}' = \overline{s}\sigma$. We apply the substitution both to the oracle and the protocol, and the hypothesis allows us to get, for all N:

$$
\begin{array}{l}
\|^{(r,s)\neq(i,j)} I_r(lsid_r^I, id^I); \mathbf{out}(\langle x^I, lsid_r^I, x_{lsid}^I, x_{id}^I\rangle) \| R_s(lsid_s^R, id^R); \mathbf{out}(\langle x^R, lsid_s^R, x_{lsid}^R, x_{id}^R\rangle) \\
\quad \| I_i(lsid_i^I, id^I); \quad \mathbf{if}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad\qquad\qquad S1(x^I, lsid_i^I, x_{lsid}^I, x_{id}^I) \\
\qquad\qquad\quad \mathbf{else\ out}(\langle x^I, lsid_i^I, x_{lsid}^I, x_{id}^I\rangle), \\
\quad \| R_j(lsid_j^R, id^R)[\quad \mathbf{if}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad\qquad\qquad S_2(x^R, lsid_j^R, x_{lsid}^R, x_{id}^R) \\
\qquad\qquad\quad \mathbf{else\ out}(\langle x^R, lsid_j^R, x_{lsid}^R, x_{id}\rangle) \\
\cong_{\mathcal{O},\mathcal{O}_{\overline{s}'}} \\
\|^{(r,s)\neq(i,j)} I_r(lsid_r^I, id^I); \mathbf{out}(\langle x^I, lsid_r^I, x_{lsid}^I, x_{id}^I\rangle) \| R_s(lsid_s^R, id^R); \mathbf{out}(\langle x^R, lsid_s^R, x_{lsid}^R, x_{id}^R\rangle) \\
\quad \| \quad I_i(lsid_i^I, id^I); \quad \mathbf{if}\ x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad\qquad\qquad \mathbf{out}(\langle k, lsid_i^I, x_{lsid}^I, x_{id}^I\rangle) \\
\qquad\qquad\quad \mathbf{if}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad\qquad\qquad T_1(x^I, lsid_r^I, x_{lsid}^I, x_{id}^I) \\
\qquad\qquad\quad \mathbf{else\ out}(\langle x^I, lsid_i^I, x_{lsid}, x_{id}\rangle) \\
\quad \| \quad R_j(lsid_j^R, id^R)[\quad \mathbf{if}\ x_{lsid}^R = lsid_i^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad\qquad\qquad \mathbf{out}(\langle k, lsid_i^R, x_{lsid}^R, x_{id}^R\rangle) \\
\qquad\qquad\quad \mathbf{if}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad\qquad\qquad T_2(x^R, lsid_r^R, x_{lsid}^R, x_{id}^R) \\
\qquad\qquad\quad \mathbf{else\ out}(\langle x^R, lsid_i^R, x_{lsid}^R, x_{id}^R\rangle)
\end{array}
$$

We remark that, for any r:

$$\nu\overline{s}.in(x,\overline{y}); \underset{(r,t)>(i,j)}{\mathsf{if}}\ x_{lsid}^{=} lsid_t^R \wedge x_{id} = id^R\ \mathbf{then\ out}(k_{r,t},\overline{y})\ \mathbf{else\ out}(x,\overline{y})$$

and

$$\nu\overline{s}.in(x,\overline{y}); \underset{(t,r)>(i,j)}{\mathsf{if}}\ x_{lsid}^R = lsid_t^I \wedge x_{id} = id^R\ \mathbf{then\ out}(k_{t,r},\overline{y})\ \mathbf{else\ out}(x,\overline{y})$$

and (resp. with S_1)

$$\nu\overline{s}.in(\overline{y}); \underset{(r,t)>(i,j)}{\mathsf{if}}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then}\ T_1(\overline{y})$$

and (resp. with S_2)

$$\nu\overline{s}.in(\overline{y}); \underset{(t,r)>(i,j)}{\mathsf{if}}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then}\ T_2(\overline{y})$$

are $\mathcal{O}_{\overline{s}}$-simulatable by the attacker as all $lsid_j^R, lsid_j^I$ are simulatable with $\mathcal{O}_{\overline{s}}$

They are then all simulatable in parallel at the same time (Theorem 5.1) and using function application (Theorem 5.4), we get:

$$
\begin{array}{l}
\|_{r\le N} KE_r[\ \underset{(r,t)>(i,j)}{\mathsf{if}}\ x^I_{lsid} = lsid^R_t \wedge x^I_{id} = id_R \textbf{ then} \\
\quad \textbf{out}(\langle k_{r,t}, lsid^I_r, x^I_{lsid}, x^I_{id}\rangle) \\
\quad \underset{(r,t)>(i,j)}{\mathsf{if}}\ x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
\quad T_1(x^I, lsid^I_r, x^I_{lsid}, x^I_{id}) \\
\quad \textbf{else if } x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
\quad S1(x^I, lsid^I, x^I_{lsid}, x^I_{id}) \\
\quad \textbf{else out}(\langle x^I, lsid^I, x^I_{lsid}, x^I_{id}\rangle),
\end{array}
\quad , \quad
\begin{array}{l}
\underset{(t,r)>(i,j)}{\mathsf{if}}\ x^R_{lsid} = lsid^I_t \wedge x^R_{id} = id^I) \textbf{ then} \\
\quad \textbf{out}(\langle k_{t,r}, lsid^R_r, x^R_{lsid}, x^R_{id}) \\
\quad \underset{(t,r)>(i,j)}{\mathsf{if}}\ x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
\quad T_2(x^R, lsid^R_r, x^R_{lsid}, x^R_{id}) \\
\quad \textbf{else if } x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
\quad S_2(x^R, lsid^R, x^R_{lsid}, x^R_{id}) \\
\quad \textbf{else out}(\langle x^R, lsid^R, x^R_{lsid}, x^R_{id}\rangle)]
\end{array}
$$

$$
\begin{array}{l}
\cong_{\mathcal{O}} \\
\quad \|_{(r,s)\neq(i,j)} I_r(lsid^r_I, id_I);\ \underset{(r,t)>(i,j)}{\mathsf{if}}\ x^I_{lsid} = lsid^R_t \wedge x^I_{id} = id_R \textbf{ then} \\
\qquad \textbf{out}(\langle k_{r,t}, lsid^I_r, x^I_{lsid}, x^I_{id}\rangle) \\
\qquad \underset{(r,t)>(i,j)}{\mathsf{if}}\ x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
\qquad T_1(x^I, lsid^I_r, x^I_{lsid}, x^I_{id}) \\
\qquad \textbf{else if } x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
\qquad S1(x^I, lsid^I, x^I_{lsid}, x^I_{id}) \\
\qquad \textbf{else out}(\langle x^I, lsid^I, x^I_{lsid}, x^I_{id}\rangle) \\
\\
\qquad \|R_s(lsid^s_R, id_R);\ \underset{(t,r)>(i,j)}{\mathsf{if}}\ x^R_{lsid} = lsid^I_t \wedge x^R_{id} = id^I) \textbf{ then} \\
\qquad \textbf{out}(\langle k_{t,r}, lsid^R_s, x^R_{lsid}, x^R_{id}) \\
\qquad \underset{(t,r)>(i,j)}{\mathsf{if}}\ x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
\qquad T_2(x^R, lsid^R_s, x^R_{lsid}, x^R_{id}) \\
\qquad \textbf{else if } x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
\qquad S_2(x^R, lsid^R_s, x^R_{lsid}, x^R_{id}) \\
\qquad \textbf{else out}(\langle x^R, lsid^R_s, x^R_{lsid}, x^R_{id}\rangle) \\
\quad I_i(lsid^i_I, id_I);\ \underset{(i,t)>(i,j)}{\mathsf{if}}\ x^I_{lsid} = lsid^R_t \wedge x^I_{id} = id_R \textbf{ then} \\
\qquad \textbf{out}(\langle k_{r,t}, lsid^I_i, x^I_{lsid}, x^I_{id}\rangle) \\
\qquad \textbf{if } x^I_{lsid} = lsid^R_j \wedge x^I_{id} = id_R) \textbf{ then} \\
\qquad \textbf{out}(\langle k, lsid^I_i, x^I_{lsid}, x^I_{id}\rangle) \\
\qquad \underset{(i,t)>(i,j)}{\mathsf{if}}\ x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
\qquad T_1(x^I, lsid^I_i, x^I_{lsid}, x^I_{id}) \\
\qquad \textbf{else if } x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
\qquad S1(x^I, lsid^I_i, x^I_{lsid}, x^I_{id}) \\
\qquad \textbf{else out}(\langle x^I, lsid^I_i, x^I_{lsid}, x^I_{id}\rangle) \\
\\
\quad R_j(lsid^j_R, id_R);\ \underset{(t,j)>(i,j)}{\mathsf{if}}\ x^R_{lsid} = lsid^I_t \wedge x^R_{id} = id^I) \textbf{ then} \\
\qquad \textbf{out}(\langle k_{t,r}, lsid^R_j, x^R_{lsid}, x^R_{id}\rangle) \\
\qquad \textbf{if } x^R_{lsid} = lsid^I_i \wedge x^R_{id} = id_I) \textbf{ then} \\
\qquad \textbf{out}(\langle k, lsid^R_j, x^R_{lsid}, x^R_{id}\rangle) \\
\qquad \underset{(t,j)>(i,j)}{\mathsf{if}}\ x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
\qquad T_2(x^R, lsid^R_j, x^R_{lsid}, x^R_{id}) \\
\qquad \textbf{else if } x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
\qquad S_2(x^R, lsid^R_j, x^R_{lsid}, x^R_{id}) \\
\qquad \textbf{else out}(\langle x^R, lsid^R_j, x^R_{lsid}, x^R_{id}\rangle)
\end{array}
$$

After α-renaming k into $k_{i,j}$, this is exactly $G^1_{(i,j)} \cong G^0_{(i,j)}$, which concludes the proof. Note that the advantage, for any (i,j), against $G^1_{(i,j)} \cong G^0_{(i,j)}$ is bounded, using the bound from Theorem 5.4, by the the advantage against $G^1_{(0,0)} \cong G^0_{(0,0)}$, the case where the most things are simulated.

■

Corollary B.1. *Let $\mathcal{O}_{ke}$, $\mathcal{O}$ be oracles and $KE_i[__1,__2] := I(lsid_i^I, id^I);__1 \| R(lsid_i^R, id^R);__2$ a key exchange protocol, such that I binds $x^I, x_{id}^I, x_{lsid}^I$, R binds $x^R, x_{id}^R, x_{lsid}^R$ and $\mathcal{N}_l(KE)$ is disjoint of the oracle support. Let id^I, id^R be names and $\overline{s}^I = \{lsid_i^I\}_{i\in\mathbb{N}}, \overline{s}^R = \{lsid_i^R\}_{i\in\mathbb{N}}$ sets of names :*

1. $\forall i \geq 1, (\nu lsid_i^I, id^I, lsid_i^R, id^R.$

$$KE_i[\textbf{out}(\langle x^I, lsid_i^I, x_{lsid}^I, x_{id}^I\rangle), \textbf{out}(\langle x^R, lsid_i^R, x_{lsid}^R, x_{id}^R\rangle)] \| \textbf{out}(\langle lsid_i^R, lsid_i^I\rangle)$$

   is $\mathcal{O}_{ke}$ simulatable)).
2. *$\overline{s}$ is disjoint of the support of $\mathcal{O}$.*
3. $KE_0[\textbf{out}(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle), \textbf{out}(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R)] \cong_{\mathcal{O}_{ke},\mathcal{O}}$
   $KE_0[$ **if** $x_{lsid}^I = lsid_0^R \wedge x_{id}^I = id^R$ **then**
   out$(\langle k, lsid_0^I, x_{lsid}, x_{id}\rangle)$
   else if $x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R$ **then**
   $\bot$
   else $out(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle)$,
   if $x_{lsid}^R = lsid_0^I \wedge x_{id}^R = id^I$ **then**
   out$(\langle k, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle)$
   else if $x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I$ **then**
   $\bot$
   else out$(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle)]$

Then, for any N which depends on the security parameter:

$$\begin{aligned}
&\|^{i\leq N} KE_i[\textbf{out}(x^I), \textbf{out}(x^R)] \cong_{\mathcal{O}} \\
&\quad \|^{i\leq N} \quad KE_i[\ \textbf{if } (x_{id}^I = id^R) \textbf{ then} \\
&\qquad\qquad \textbf{if}_{1\leq j\leq N} x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id_R \textbf{ then} \\
&\qquad\qquad\quad \textbf{out}(k_{i,j}) \\
&\qquad\qquad \textbf{else out}(x^I), \\
&\qquad\qquad \textbf{if } (x_{id}^R = id_I) \textbf{ then} \\
&\qquad\qquad \textbf{if}_{1\leq j\leq N} x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id_I \textbf{ then} \\
&\qquad\qquad\quad \textbf{out}(k_{j,i}) \\
&\qquad\qquad \textbf{else out}(x^R)]
\end{aligned}$$

Proof. Let us fix N, which may depend on the security parameter.

Ry direct application of Theorem 5.5, with $P := I(lsid^I, id^I); \mathbf{out}(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I\rangle) \| R(lsid^R, id^R); \mathbf{out}(\langle x^R, lsid^R, x$
$R := KE$, and Q being the right handside of hypothesis (3), we get that:

$$\begin{aligned}
&\|^{i\leq N} KE_i[\mathbf{out}(\langle x^I, lsid_i^I, x_{lsid}^I, x_{id}^I\rangle), \mathbf{out}(\langle x^R, lsid_i^R, x_{lsid}^R, x_{id}^R\rangle)] \\
&\cong_{\mathcal{O},\mathcal{O}_{ke}} \\
&\quad \|^{i\leq N-1} KE_i[\mathbf{out}(\langle x^I, lsid_i^I, x_{lsid}^I, x_{id}^I\rangle), \mathbf{out}(\langle x^R, lsid_i^R, x_{lsid}^R, x_{id}^R\rangle)] \\
&\| KE_0[\ \mathbf{if}\ x_{lsid}^I = lsid^R \wedge x_{id} = id^R\ \mathbf{then} \\
&\qquad \mathbf{out}(\langle k, lsid^I, x_{lsid}, x_{id}\rangle) \\
&\qquad \mathbf{else\ if}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
&\qquad \bot \\
&\qquad \mathbf{else}\ out(\langle x^I, lsid^I, x_{lsid}^I, x_{id}^I\rangle), \\
&\qquad \mathbf{if}\ x_{lsid}^R = lsid^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
&\qquad \mathbf{out}(\langle k, lsid^R, x_{lsid}^R, x_{id}^R\rangle) \\
&\qquad \mathbf{else\ if}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
&\qquad \bot \\
&\qquad \mathbf{else\ out}(\langle x^R, lsid^R, x_{lsid}^R, x_{id}^R\rangle)]
\end{aligned}$$

This allows us to obtain the hypothesis of Proposition B.2, where $\mathcal{O}_{\overline{s}}$ is instantiated with $\mathcal{O}_{ke}$. We thus conclude using Proposition B.2.

■

Corollary B.2. *Let $\mathcal{O}_T$, $\mathcal{O}_{ke}$, $\mathcal{O}_r$,$\mathcal{O}_{P,Q}$ be oracles and*

$KE_i[__1,__2] := I(lsid_i^I, id^I);__1 \| R(lsid_i^R, id^R);__2$ *a key exchange protocol, such that I binds $x^I, x_{id}^I, x_{lsid}^I$, R binds $x^R, x_{id}^R, x_{lsid}^R$ and $\mathcal{N}_l(KE)$ is disjoint of the oracle support. Let id^I, id^R be names, $\overline{s}^I = \{lsid_i^I\}_{i\in\mathbb{N}}$,$\overline{s}^R = \{lsid_i^R\}_{i\in\mathbb{N}}$ and $\overline{s} = \overline{s}^I \cap \overline{s}^R$ sets of names.*

Let $\overline{p} = \{id^I, id^R\}$, $P(x,\overline{y}) = P_1(x,\overline{y}) \| P_2(x,\overline{y})$ and $Q(x,\overline{y},\overline{z}) = Q_1(x,\overline{y},\overline{z}) \| Q_1(x,\overline{y},\overline{z})$ be parameterized protocols, such that $\mathcal{N}_l(P,Q)$ is disjoint of the oracle support.

I-1 $\forall i \geq 1, (\nu lsid_i^I, id^I, lsid_i^R, id^R. KE_i[\mathbf{out}(x^I), \mathbf{out}(x^R)] \| \mathbf{out}(\langle lsid_i^R, lsid_i^I\rangle)$ *is $\mathcal{O}_T$-simulatable)).*

I-2 *$\overline{s}$ is disjoint of the support of $\mathcal{O}_{P,Q}$.*

I-3

$$
\begin{array}{l}
KE_0[\mathbf{out}(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle), \mathbf{out}(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle) \cong_{\mathcal{O}_T,\mathcal{O}_{P,Q}} \\
\quad KE_0 \quad [\mathbf{if}\ x_{lsid}^I = lsid_0^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad \mathbf{out}(\langle k, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle) \\
\qquad \mathbf{else\ if}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \mathbf{then} \\
\qquad\quad \bot \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^I, lsid_0^I, x_{lsid}^I, x_{id}^I\rangle), \\
\qquad \mathbf{if}\ x_{lsid}^R = lsid^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad\quad \mathbf{out}(\langle k, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle) \\
\qquad \mathbf{else\ if}\ x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \mathbf{then} \\
\qquad\quad \bot \\
\qquad \mathbf{else}\ \mathbf{out}(\langle x^R, lsid_0^R, x_{lsid}^R, x_{id}^R\rangle)]
\end{array}
$$

and

R-1 $\forall\ 1 \leq i,j \leq n, \nu\overline{p}, k_{i,j}. P_0(\overline{p}, k_{i,j})$ *is $\mathcal{O}_r$-simulatable.*
R-2 $\forall\ 1 \leq i \leq n, \nu\overline{p}, k_{i,j}. Q_0(\overline{p}, k_{i,j})$ *is $\mathcal{O}_r$-simulatable.*
R-3 *$\overline{s}$ is disjoint of the support of $\mathcal{O}_k$.*
R-4 $P_0(\overline{p}, k) \cong_{\mathcal{O}_r,\mathcal{O}_{ke}} Q_0(\overline{p}, k)$

and

C-1 $\nu\overline{p}. in(x_i^I).P_i^I(x_i^I) \| in(x_i^R).P_i^R(x_i^R)$ *is $\mathcal{O}_{P,Q}$-simulatable.*

1. $\nu\overline{p}.$

$$
\begin{array}{l}
\|^{i\leq n} \quad KE_i[\quad \mathbf{if}\ (x_{id}^I = id^R)\ \mathbf{then} \\
\qquad \underset{1\leq j\leq n}{\mathbf{if}}\ (x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id_R)\ \mathbf{then} \\
\qquad\quad out(\langle i, j\rangle) \\
\qquad \mathbf{else}\ P_i^I(x_i^I), \\
\qquad \mathbf{if}\ (x_{id}^R = id_I)\ \mathbf{then} \\
\qquad \underset{1\leq j\leq n}{\mathbf{if}}\ (x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id_I)\ \mathbf{then} \\
\qquad\quad out(\langle i, j\langle) \\
\qquad \mathbf{else}\ P_i^R(x_i^R)]
\end{array}
\qquad \text{is } \mathcal{O}_{ke}\text{-simulatable.}
$$

Then, for any n which may depend on the security parameter:

$$
\|^{i\leq n} KE_i[P_i^I(x_i^I), P_i^R(x_i^R)] \cong
$$
$$
\|^{i\leq n} KE_i[\mathbf{if}\ x_{id}^I = id^R\ \mathbf{then}\ Q_i^I(x_i^I)\ \mathbf{else}\ P_i^I(x_i^I), \mathbf{if}\ x_{id}^R = id^I\ \mathbf{then}\ Q_i^R(x_i^R)\ \mathbf{else}\ P_i^R(x_i^R)]
$$

Proof. Using Corollary B.1 on hypothesis A-1,A-2 and A-3, we get that, for all N:

$$
\begin{array}{l}
\|^{i\leq N} KE_i[\mathbf{out}(x^I), \mathbf{out}(x^R)] \cong_{\mathcal{O}} \\
\quad \|^{i\leq N} \quad KE_i[\quad \mathbf{if}\ (x^I_{id} = id^R)\ \mathbf{then} \\
\qquad\qquad\qquad\qquad \underset{1\leq j\leq N}{\mathsf{if}}\ x^I_{lsid} = lsid^R_j \wedge x^I_{id} = id_R\ \mathbf{then} \\
\qquad\qquad\qquad\qquad\quad \mathbf{out}(k_{i,j}) \\
\qquad\qquad\qquad\quad \mathbf{else}\ \mathbf{out}(x^I), \\
\qquad\qquad\qquad \mathbf{if}\ (x^R_{id} = id_I)\ \mathbf{then} \\
\qquad\qquad\qquad\qquad \underset{1\leq j\leq N}{\mathsf{if}}\ x^R_{lsid} = lsid^I_j \wedge x^R_{id} = id_I\ \mathbf{then} \\
\qquad\qquad\qquad\qquad\quad \mathbf{out}(k_{j,i}) \\
\qquad\qquad\qquad\quad \mathbf{else}\ \mathbf{out}(x^R)]
\end{array}
$$

Now, as $\nu\overline{p}, lsid^I_i, lsid^R_i.in(x).P(x) \| in(x).Q(x)$ is $\mathcal{O}_p$-simulatable (hypothesis C-1), using twice Theorem 5.4 we get that :

$$
\begin{array}{l}
\|^{i\leq N} KE_i[P^I(x^I), P^R(x^R)] \cong_{\mathcal{O}_p} \\
\quad \|^{i\leq N} \quad KE_i[\quad \mathbf{if}\ (x^I_{id} = id^R)\ \mathbf{then} \\
\qquad\qquad\qquad\qquad \underset{1\leq j\leq N}{\mathsf{if}}\ x^I_{lsid} = lsid^R_j \wedge x^I_{id} = id_R\ \mathbf{then} \\
\qquad\qquad\qquad\qquad\quad P^I(k_{i,j}) \\
\qquad\qquad\qquad\quad \mathbf{else}\ P^I(x^I), \\
\qquad\qquad\qquad \mathbf{if}\ (x^R_{id} = id_I)\ \mathbf{then} \\
\qquad\qquad\qquad\qquad \underset{1\leq j\leq N}{\mathsf{if}}\ x^R_{lsid} = lsid^I_j \wedge x^R_{id} = id_I\ \mathbf{then} \\
\qquad\qquad\qquad\qquad\quad P^R(k_{j,i}) \\
\qquad\qquad\qquad\quad \mathbf{else}\ P^R(x^R)]
\end{array}
$$

and

$$
\begin{array}{l}
\|^{i\leq N} KE_i[\mathbf{if}\ x^I_{id} = id^R\ \mathbf{then}\ Q^I(x^I)\ \mathbf{else}\ P^I(x^I), \mathbf{if}\ x^R_{id} = id^I\ \mathbf{then}\ Q^R(x^R)\ \mathbf{else}\ P^R(x^R)] \cong_{\mathcal{O}_p} \\
\quad \|^{i\leq N} \quad KE_i[\quad \mathbf{if}\ (x^I_{id} = id^R)\ \mathbf{then} \\
\qquad\qquad\qquad\qquad \underset{1\leq j\leq N}{\mathsf{if}}\ x^I_{lsid} = lsid^R_j \wedge x^I_{id} = id_R\ \mathbf{then} \\
\qquad\qquad\qquad\qquad\quad Q^I(k_{i,j}) \\
\qquad\qquad\qquad\quad \mathbf{else}\ P^I(x^I), \\
\qquad\qquad\qquad \mathbf{if}\ (x^R_{id} = id_I)\ \mathbf{then} \\
\qquad\qquad\qquad\qquad \underset{1\leq j\leq N}{\mathsf{if}}\ x^R_{lsid} = lsid^I_j \wedge x^R_{id} = id_I\ \mathbf{then} \\
\qquad\qquad\qquad\qquad\quad Q^R(k_{j,i}) \\
\qquad\qquad\qquad\quad \mathbf{else}\ P^R(x^R)]
\end{array}
$$

Moreover, using Theorem 5.5 on hypothesis B-1,B-2,B-3 and B-4, we get that

$$\forall n\ \|^{i\leq N^2} P_i(\overline{p}, k_i) \cong_{\mathcal{O}_k} Q_i(\overline{p}, k_i)$$

Combined with Theorem 5.2 on the $\mathcal{O}_k$ simulatability of the key exchange (hypothesis C-2) we get:

$$
\begin{array}{lll}
\|^{i\leq N} & KE_i[& \textbf{if } (x^I_{id} = id^R) \textbf{ then} \\
& & \quad \textsf{if}_{1\leq j\leq N} x^I_{lsid} = lsid^R_j \wedge x^I_{id} = id_R \textbf{ then} \\
& & \quad\quad P^I(k_{i,j}) \\
& & \quad \textbf{else } P^I(x^I), \\
& & \textbf{if } (x^R_{id} = id_I) \textbf{ then} \\
& & \quad \textsf{if}_{1\leq j\leq N} x^R_{lsid} = lsid^I_j \wedge x^R_{id} = id_I \textbf{ then} \\
& & \quad\quad P^R(k_{j,i}) \\
& & \quad \textbf{else } P^R(x^R)] \\
& & \quad\quad \cong \\
\|^{i\leq N} & KE_i[& \textbf{if } (x^I_{id} = id^R) \textbf{ then} \\
& & \quad \textsf{if}_{1\leq j\leq N} x^I_{lsid} = lsid^R_j \wedge x^I_{id} = id_R \textbf{ then} \\
& & \quad\quad Q^I(k_{i,j}) \\
& & \quad \textbf{else } P^I(x^I), \\
& & \textbf{if } (x^R_{id} = id_I) \textbf{ then} \\
& & \quad \textsf{if}_{1\leq j\leq N} x^R_{lsid} = lsid^I_j \wedge x^R_{id} = id_I \textbf{ then} \\
& & \quad\quad Q^R(k_{j,i}) \\
& & \quad \textbf{else } P^R(x^R)]
\end{array}
$$

We thus conclude with transitivity.

■

Corollary B.3. *Let $\mathcal{O}_{KE}$, $\mathcal{O}_r$,$\mathcal{O}_{P,Q}$ be oracles and*

$$KE_i[__1, __2] := I_i(lsid^I_i, id^I); __1 | R_i(lsid^R i, id^R); __2$$

a key exchange protocol with $I_i(lsid^I_i, id^I) := I^0_i(lsid^I_i, id^I); I^1_i(x^I)$ and $R_i(lsid^R_i, id^R) := R^0_i(lsid^R_i, id^R); R^1_i(x^R)$ such that I^0 binds x^I, x_{id}, x_{lsid}, R^0 binds x^R, x_{id}, x_{lsid} and $\mathcal{N}_l(KE)$ is disjoint of the oracles support. Let $\overline{p} = \{id^I, id^R\}$, $P_i(x, \overline{y}) = P^I_i(x, \overline{y}) \| P^R_i(x, \overline{y})$,$Q(x, \overline{y}, \overline{z}) = Q^I_i(x, \overline{y}, \overline{z}) \| Q^R_i(x, \overline{y}, \overline{z})$, $C_i(\overline{z})$ and $D_i(\overline{z})$ be protocols, such that $\mathcal{N}_l(P, Q, C, D)$ is disjoint of the oracles support.

Let id^I, id^R be names, $\overline{s}^I = \{lsid^I_i\}_{i\in\mathbb{N}}$,$\overline{s}^R = \{lsid^R_i\}_{i\in\mathbb{N}}$ and $\overline{s} = \overline{s}^I \cap \overline{s}^R$ sets of names.

A-1 $\forall i \in \mathbb{N}, (\nu lsid^I_i, id^I, lsid^R_i, id^R. C_i(\overline{p}) \| I^0_i(lsid^I_i, id^I); \textbf{\textit{out}}(x^I) \| R^0_i(lsid^R_i, id^R); \textbf{\textit{out}}(x^R)$ *is $\mathcal{O}_{KE}$ simulatable)).*

A-2 *$\overline{s}$ is disjoint of the support of $\mathcal{O}_p$.*

A-3

$$
\begin{array}{lll}
C_i(\overline{p}) \| I^0_0(lsid^I_0, id^I); & \textbf{\textit{if}}\ x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{\textit{ then}} & \\
& \quad I^1(x^I); \textbf{\textit{out}}(x^I) & \\
& \textbf{\textit{else out}}(\langle x^I, lsid^I_0, x^I_{lsid}, x^I_{id}\rangle) & \\
\| R^0(lsid^R_0, id^R); & \textbf{\textit{if}}\ x^R_{lsid} \notin \overline{s}^I \wedge x_{id} = id^I \textbf{\textit{ then}} & \\
& \quad R^1(x^R); \textbf{\textit{out}}(x^R) & \\
& \textbf{\textit{else out}}(\langle x^R, lsid^R, x^R_{lsid}, x^R_{id}\rangle) & \\
\cong_{\mathcal{O}_{KE}, \mathcal{O}_p} & & \\
C_i(\overline{p}) \| I^0(lsid^I_0, id^I); & \textbf{\textit{if}}\ x^I_{lsid} = lsid^R \wedge x_{id} = id^R \textbf{\textit{ then}} & \\
& \quad \textbf{\textit{out}}(\langle k, lsid^I_0, x^I_{lsid}, x^I_{id}\rangle) & \\
& \textbf{\textit{else if}}\ x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{\textit{ then}} & \\
& \quad I^1(x^R); \perp & \\
& \textbf{\textit{else out}}(\langle x^I, lsid^I, x^I_{lsid}, x^I_{id}\rangle) & \\
\| R^0(lsid^R_0, id^R); & \textbf{\textit{if}}\ x^R_{lsid} = lsid^I \wedge x^R_{id} = id^I \textbf{\textit{ then}} & \\
& \quad \textbf{\textit{out}}(\langle k, lsid^R_0, x^R_{lsid}, x^R_{id}\rangle) & \\
& \textbf{\textit{else if}}\ x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{\textit{ then}} & \\
& \quad I^1(x^R); \perp & \\
& \textbf{\textit{else out}}(\langle x^R, lsid^R_0, x^R_{lsid}, x^R_{id}\rangle) &
\end{array}
$$

and for any N which may depend on the security parameter:

B-1 $\|^{i\leq N^2} D_i(\overline{p})\|I_i^1(k_i); P_i^I(\overline{p},k_i)\|B_i^1(k_i); P_i^R(\overline{p},k_i) \cong_{\mathcal{O}_r,\mathcal{O}_k} \|^{i\leq n^2} D_i(\overline{p})\|I_i^1(k_i); Q_i^I(\overline{p},k_i)\|B_i^1(k_i); Q_i^R(\overline{p},k_i)$

and

C-1 $\nu\overline{p}, lsid_i^I, lsid_i^R.D_i(\overline{p})\|in(x).P_i(x)\ \|in(x).Q_i(x)\|in(x).I_i^1(x); P_i^I(x)\|in(x).R_i^1(x); P_i^R(x)$
$\|in(x).I_i^1(x); Q_i^I(x)\|in(x).R_i^1(x); Q_i^R(x)$ *is $\mathcal{O}_p$ simulatable.*

$$\|^{i\leq N}\ C_i(\overline{p})\|I_i^0(lsid_i^I, id^I);\quad \underset{1\leq j\leq N}{\textit{if}}\ x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id^R\ \textbf{\textit{then}}\ \textbf{\textit{out}}(\langle i,j\rangle)\ \textbf{\textit{else if}}\ x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R\ \textbf{\textit{then}}\ I_i^1(x^I); \bot\ \textbf{\textit{else}}\ I_i^1(x^I); P_i^I(x^I)$$

C-2 $\nu\overline{p}.$

$$\|R_i^0(lsid_i^R, id^R)[\quad \underset{1\leq j\leq N}{\textit{if}}\ x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id^I\ \textbf{\textit{then}}\ \textbf{\textit{out}}(\langle i,j\rangle)\ \textbf{\textit{else if}}\ (x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I\ \textbf{\textit{then}}\ R_i^1(x^R); \bot\ \textbf{\textit{else}}\ R_i^1(x^R); P_i^R(x^R)$$

is $\mathcal{O}_k$ simulatable.

Then, for any n:

$$\|^{i\leq N} C_i(\overline{p})\|D_i(\overline{p})\|KE_i[P_i^I(x^I), P_i^R(x^R)] \cong$$
$$\|^{i\leq N} C_i(\overline{p})\|D_i(\overline{p})\|KE_i[\textbf{\textit{if}}\ x_{id}^I = id^R\ \textbf{\textit{then}}\ Q_i^I(x^I)\ \textbf{\textit{else}}\ P_i^I(x^I), \textbf{\textit{if}}\ x_{id}^R = id^I\ \textbf{\textit{then}}\ Q_i^R(x^R)\ \textbf{\textit{else}}\ P_i^R(x^R)]$$

Proof. Let N an integer, which may depend on the security parameter. Ry application of Theorem 5.5, with P and R as the left handside of hypothesis A-3, and Q being the right handside of hypothesis A-3, we get that:

$$\begin{array}{l}
\|^{i\leq n-1} C_i(\overline{p})\|I_i^0(lsid_i^I, id^I); \mathbf{out}(x^I)\|R_i^0(lsid_i^R, id^R); \mathbf{out}(x^R) \\
\quad C_n(\overline{p})\|I_n^0(lsid_n^I, id^I);\quad \textbf{if } (x_{lsid}^I \notin \overline{s}^R \wedge x_{id} = id^R \textbf{ then} \\
\qquad\qquad\qquad\qquad I_n^1(x^I); \mathbf{out}(x^I) \\
\qquad\qquad\qquad\qquad \textbf{else out}(x^I) \\
\| \quad \|R_n^0(lsid_n^R, id^R);\qquad \textbf{if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^I \textbf{ then} \\
\qquad\qquad\qquad\qquad R_n^1(x^R); \mathbf{out}(x^R) \\
\qquad\qquad\qquad\qquad \textbf{else out}(x^R) \\
\cong_{\mathcal{O}_P} \\
\quad \|^{i\leq N-1} C_i(\overline{p})\|I_i^0(lsid_i^I, id^I); \mathbf{out}(x^I)\|R_i^0(lsid_i^R, id^R); \mathbf{out}(x^R) \\
\quad\ \| \quad C_n(\overline{p})\|I_n^0(lsid_n^I, id^I);\quad \textbf{if } x_{lsid} = lsid_R^n \wedge x_{id}^I = id^R \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{out}(k) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad I_n^1(x^I); bad \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else out}(x^I) \\
\qquad\qquad \|R_n^0(lsid_n^R, id^R)[\qquad \textbf{if } x_{lsid}^R = lsid_I^n \wedge x_{id} = id^I \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{out}(k) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad R_n^1(x^R); bad \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else out}(x^R)
\end{array}$$

Using Proposition B.2, with $S_1 = I^1(x^I); \mathbf{out}(x^I), S_2 = R^1(x^R); \mathbf{out}(x^R), R_1 = I^1(x^I); \bot, R_2 = R^1(x^R); \bot$, we get that:

$$
\begin{array}{lll}
C_i(\overline{p})\|I_i^0(lsid_i^I, id^I); & \textbf{if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} & \\
 & \quad I_i^1(x^I); \textbf{out}(x^I) & \\
 & \textbf{else out}(x^I) & \\
\|^{i\leq N} \quad \|R_i^0(lsid_i^R, id^R); & \textbf{if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} & \\
 & \quad R_i^1(x^R); \textbf{out}(x^R) & \\
 & \textbf{else out}(x^R) &
\end{array}
$$

$$
\cong_{\mathcal{O}_P}
$$

$$
\begin{array}{lll}
\|^{i\leq N} \quad C_i(\overline{p})\|I_i^0(lsid_i^I, id^I); & \textbf{if } \underset{1\leq j\leq N}{} x_{lsid}^I = lsid_R^j \wedge x_{id}^I = id^R \textbf{ then} & \\
 & \quad \textbf{out}(k_{i,j}) & \\
 & \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} & \\
 & \quad I_i^1(x^I); bad & \\
 & \textbf{else out}(x^I) & \\
 \|R_i^0(lsid_i^R, id^R)[& \textbf{if } \underset{1\leq i\leq N}{} x_{lsid}^R = lsid_I^j \wedge x_{id}^R = id^I \textbf{ then} & \\
 & \quad \textbf{out}(k_{j,i}) & \\
 & \textbf{else if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} & \\
 & \quad R_i^1(x^R); bad & \\
 & \textbf{else out}(x^R) &
\end{array}
$$

Now, with this context, using twice using Theorem 5.4 with the simulatability of $\nu\overline{p}, lsid_i^I, lsid_i^R.D_i(\overline{p})$
$\|in(x).P_i(x)\|in(x).I_i^1(x); P_i^I(x)\|in(x).R_i^1(x); P_i^R(x)$ from C-1, we may get that:

$$
\begin{array}{lll}
\|^{i\leq N} C_i(\overline{p})\|D_i(\overline{p})\|I_i^0(lsid_i^I, id^I); & \textbf{if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} & \\
 & \quad I_i^1(x^I); P_i^I(x^I) & \\
 & \textbf{else } I_i^1(x^I); P_i^I(x^I) & \\
\|R_i^0(lsid_i^R, id^R); & \textbf{if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id} = id^I \textbf{ then} & \\
 & \quad R_i^1(x^R); P_i^R(x^R) & \\
 & \textbf{else } R_i^1(x^R); P_i^R(x^R) &
\end{array}
$$

$$
\cong_{\mathcal{O}_P}
$$

$$
\begin{array}{lll}
\|^{i\leq N} \quad C_i(\overline{p})\|D_i(\overline{p})\|I_i^0(lsid_i^I, id^I); & \textbf{if } \underset{1\leq j\leq N}{} x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id^R \textbf{ then} & \\
 & \quad I_i^1(k_{i,j}); P_i^1(k_{i,j}) & \\
 & \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} & \\
 & \quad I_i^1(x^I); \perp & \\
 & \textbf{else } I_i^1(x^I); P_i^I(x^I) & \\
 \|R_i^0(lsid_i^R, id^R)[& \textbf{if } \underset{1\leq k\leq m}{} x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id^I \textbf{ then} & \\
 & \quad R_i^1(k_{j,i}); P_i^R(k_{j,i}) & \\
 & \textbf{else if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} & \\
 & \quad R_i^1(x^R); \perp & \\
 & \textbf{else } R_i^1(x^R); P_i^R(x^R) &
\end{array}
$$

We can simplify the left handside of the equivalence and get that:

$$
\begin{array}{l}
\|^{i\leq N} C_i(\overline{p})\|D_i(\overline{p})\|I_i^0(lsid_i^I, id^I); \quad I_i^1(x^I); P_i^I(x^I) \\
\|R_i^0(lsid_i^R, id^R); \qquad\qquad\qquad\quad R_i^1(x^R); P_i^R(x^R) \\
\cong_{\mathcal{O}_P} \\
\qquad \|^{i\leq N} \quad C_i(\overline{p})\|D_i(\overline{p})\|I_i^0(lsid_i^I, id^I); \quad \textsf{if} \underset{1\leq j\leq m}{} x_{lsid}^I = lsid_R^j \wedge x_{id} = id^R \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad I_i^1(k_{i,j}); P_i^1(k_{i,j}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad I_i^1(x^I); bad \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } I_i^1(x^I); P_i^I(x^I) \\
\qquad\qquad\quad \|R_i^0(lsid_i^R, id^R); \qquad\quad \textsf{if} \underset{1\leq j\leq N}{} x_{lsid}^R = lsid_I^j \wedge x_{id}^R = id^I \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad R_i^1(k_{j,i}); P_i^R(k_{j,i}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad R_i^1(x^R); \bot \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } R_i^1(x^R); P_i^R(x^R)
\end{array}
$$

Ry performing the same operation with Q, we can also get:

$$
\begin{array}{l}
\qquad C_i(\overline{p})\|D_i(\overline{p})\|I_i^0(lsid_i^I, id^I); \quad \textbf{if } x_{id}^I = id^R \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad I_i^1(x^I); Q_i^I(x^I) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } I_i^1(x^I); P_i^I(x^I) \\
\|^{i\leq N} \quad \|R_i^0(lsid_i^R, id^R); \qquad\quad \textbf{if } x_{id}^R = id^I \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad R_i^1(x^R); Q_i^R(x^R) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } R_i^1(x^R); P_i^R(x^R) \\
\cong_{\mathcal{O}_P} \\
\qquad \|^{i\leq N} \quad C_i(\overline{p})\|D_i(\overline{p})\|I_i^0(lsid_i^I, id^I); \quad \textsf{if} \underset{1\leq j\leq N}{} x_{lsid}^I = lsid_j^R \wedge x_{id}^I = id^R \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad I_i^1(k_{i,j}); Q_i^1(k_{i,j}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else if } x_{lsid}^I \notin \overline{s}^R \wedge x_{id}^I = id^R \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad I_i^1(x^I); \bot \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } I_i^1(x^I); P^I(x^I) \\
\qquad\qquad\quad \|R_i^0(lsid_i^R, id^R)[\qquad\quad \textsf{if} \underset{1\leq j\leq N}{} x_{lsid}^R = lsid_j^I \wedge x_{id}^R = id^I \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad R_i^1(k_{j,i}); Q_i^R(k_{j,i}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else if } x_{lsid}^R \notin \overline{s}^I \wedge x_{id}^R = id^I \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad R_i^1(x^R); \bot \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } R_i^1(x^R); P_i^R(x^R)
\end{array}
$$

To conclude with transitivity, we must prove the equivalence between the two idealized version with either P or Q.

Combining Hypothesis B-1 with Theorem 5.2 on the $\mathcal{O}_k$ simulatability of the key exchange (hypothesis C-2) we do get the necessary equivalence to conclude:

$$\begin{array}{lll}
\|^{i\le N} & C_i(\overline{p})\|D_i(\overline{p})\|I_i^0(lsid_i^I, id^I); & \mathsf{if}_{1\le j\le N} x^I_{lsid} = lsid_j^R \wedge x_{id} = id^R \textbf{ then} \\
 & & \quad I_i^1(k_{i,j}); P_i^1(k_{i,j}) \\
 & & \textbf{else if } x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
 & & \quad I_i^1(x^I); bad \\
 & & \textbf{else } I_i^1(x^I); P_i^I(x^I) \\
 & \|R_i^0(lsid_i^R, id^R)[& \mathsf{if}_{1\le j\le N} x^R_{lsid} = lsid_I^j \wedge x^R_{id} = id^I \textbf{ then} \\
 & & \quad R_i^1(k_{j,i}); P_i^R(k_{j,i}) \\
 & & \textbf{else if } x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
 & & \quad R_i^1(x^R); bad \\
 & & \textbf{else } R_i^1(x^R); P_i^R(x^R)
\end{array}$$

$\cong$

$$\begin{array}{lll}
\|^{i\le N} & C_i(\overline{p})\|D_i(\overline{p})\|I_i^0(lsid_i^I, id^I); & \mathsf{if}_{1\le j\le N} x^I_{lsid} = lsid_j^R \wedge x^I_{id} = id^R \textbf{ then} \\
 & & \quad I_i^1(k_{i,j}); Q_i^1(k_{i,j}) \\
 & & \textbf{else if } x^I_{lsid} \notin \overline{s}^R \wedge x^I_{id} = id^R \textbf{ then} \\
 & & \quad I_i^1(x^I); \bot \\
 & & \textbf{else } I_i^1(x^I); P_i^I(x^I) \\
 & \|R_i^0(lsid_i^R, id^R); & \mathsf{if}_{1\le j\le N} x^R_{lsid} = lsid_j^I \wedge x^R_{id} = id^I \textbf{ then} \\
 & & \quad R_i^1(k_{j,i}); Q_i^R(k_{j,i}) \\
 & & \textbf{else if } x^R_{lsid} \notin \overline{s}^I \wedge x^R_{id} = id^I \textbf{ then} \\
 & & \quad R_i^1(x^R); \bot \\
 & & \textbf{else } R_i^1(x^R); P_i^R(x^R)
\end{array}$$

■

C Appendix of Part III

C.1 Proof of Chapter 7

Lemma 7.1. *Let $P_1, Q_1 \in \mathcal{P}(I,R)$, $P_2, Q_2 \in \overline{\mathcal{P}}(I,R)$. When $\sigma : I \to R_I$ is the substitution that replaces each variable in I by a fresh random variable in R_I, we have:*

$$P_1 \mid P_2 \approx_{D_k} Q_1 \mid Q_2 \Leftrightarrow (P_1\sigma, R_I) \mid P_2\sigma \approx_{D_k} (Q_1\sigma, R_I) \mid Q_2\sigma$$

Proof. Let $P_1, Q_1 \in \mathcal{P}(I,R)$,$P_2, Q_2 \in \overline{\mathcal{P}}(I,R)$, we have:

$$\begin{aligned}
&P_1 \mid P_2 \approx_{D_k} Q_1 \mid Q_2 \\
&\quad\Leftrightarrow \forall \vec{i} \in D_k^{|I|},\ \forall \vec{o} \in D_k^n.\ [(P_1,P_2)]^{\vec{i}}_{D_k}(\vec{o},\vec{0}) = [(Q_1,Q_2)]^{\vec{i}}_{D_k}(\vec{o},\vec{0}) \\
&\quad\Leftrightarrow \forall \vec{t} \in D_k^{|I|},\ \forall \vec{o} \in D_k^n\ .[(P_1\sigma,P_2\sigma,R_I)]^{\vec{i}}_{D_k}(\vec{o},\vec{0},\vec{t}) = [(Q_1\sigma,Q_2\sigma,R_I)]^{\vec{i}}_{D_k}(\vec{o},\vec{0},\vec{t}) \\
&\quad\Leftrightarrow \forall \vec{c'} \in D_k^{n+|I|}.\ [(P_1\sigma,R_I),P_2\sigma)]^{\vec{i}}_{D_k}(\vec{c'},\vec{0}) = [(Q_1\sigma,R_I),Q_2\sigma)]^{\vec{i}}_{D_k}(\vec{c'},\vec{0}) \\
&\quad\Leftrightarrow (P_1\sigma,R_I) \mid P_2\sigma \approx_{D_k} (Q_1\sigma,R_I) \mid Q_2\sigma
\end{aligned}$$

■

Lemma 7.2. *Let $P_1,\dots,P_n$ be programs over $\mathcal{P}(I,R)$, and $Y \subset R$.*

$$\perp^Y_{D_k}(P_1,\dots,P_n) \Leftrightarrow \perp_{D_k}(P_1\sigma,\dots,P_n\sigma)$$

where $\sigma : Y \to I_Y$ is the substitution that replaces each variable in Y by a fresh input variable in I_Y.

Proof.

$$\begin{aligned}
\perp^Y_{D_k}(P_1,\dots,P_n) &\Leftrightarrow \forall \vec{i} \in D_k^{|X|}, \forall \vec{i'} \in D_k^{|Y|}.\ [P_1,\dots,P_n]^{\vec{i},\vec{i'}}_{D_k} = ([P_1]^{\vec{i},\vec{i'}}_{D_k},\dots,[P_n]^{\vec{i},\vec{i'}}_{D_k}) \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|I \uplus I_Y|}.\ [(P_1,\dots,P_n)\sigma]^{\vec{i}}_{D_k} = ([P_1\sigma]^{\vec{i}}_{D_k},\dots,[P_n\sigma]^{\vec{i}}_{D_k}) \\
&\Leftrightarrow \perp_{D_k}(P_1\sigma,\dots,P_n\sigma)
\end{aligned}$$

■

Lemma 7.3. *Let $P_1,\dots,P_n$ be programs over $\mathcal{P}(I,\{r_1,\dots,r_m\})$*

$$\perp_{D_k}(P_1,\dots,P_n) \Leftrightarrow (P_1,\dots,P_n) \approx_{D_k} (P_1\sigma 1,\dots,P_n\sigma_n)$$

where σ_i is the substitution that to any r_j associates a fresh random variable r^i_j.

Proof.

$$\begin{aligned}
\perp_{D_k}(P_1,\dots,P_n) &\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P_1,\dots,P_n]^{\vec{i}}_{D_k} = ([P_1]^{\vec{i}}_{D_k},\dots,[P_n]^{\vec{i}}_{D_k}) \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P_1,\dots,P_n]^{\vec{i}}_{D_k} = ([P_1\sigma_1]^{\vec{i}}_{D_k},\dots,[P_n\sigma_n]^{\vec{i}}_{D_k}) \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P_1,\dots,P_n]^{\vec{i}}_{D_k} = [(P_1\sigma_1,\dots,P_n\sigma_n)]^{\vec{i}}_{D_k} \\
&\Leftrightarrow (P_1,\dots,P_n) \approx_{D_k} (P_1\sigma_1,\dots,P_n\sigma_n)
\end{aligned}$$

Indeed, for any $\vec{i} \in D_k^{|X|}$ we have that $[P_i]^{\vec{i}}_{D_k} = [P_i\sigma_i]^{\vec{i}}_{D_k}$ as we are only performing renaming. Moreover, $P_1\sigma_1,\dots,P_n\sigma_n$ do not share any variable, and thus trivially verify:

$$([P_1\sigma_1]^{\vec{i}}_{D_k},\dots,[P_n\sigma_n]^{\vec{i}}_{D_k}) = [(P_1\sigma_1,\dots,P_n\sigma_n)]^{\vec{i}}_{D_k}$$

■

Lemma 7.4. *Let D_k be a Σ-algebra that contains a binary symbol $+$ such that $+^{D_k}$ is right invertible. Let $P = (p_1, \ldots, p_k)$ be a program in $\mathcal{P}(I, R)$ with $\{r_1, \ldots, r_k\} \subseteq R$ and $x_1, \ldots, x_k$ fresh input variables. We have that*

$$P \approx_{D_k} r_1, \ldots, r_k \Leftrightarrow \perp_{D_k} ((p_1 + x_1, \ldots, p_k + x_k), (x_1, \ldots, x_k))$$

Proof. Let $P \in \mathcal{P}(I, R)$ of arity 1, with $r \in R$ and x a fresh deterministic variables, we have $p \approx_{D_k} r \Leftrightarrow \perp_{D_k} (p + x, x)$:

$$\begin{aligned}
P \approx_D r &\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P]^{\vec{i}}_{D_k} = [r]^{\vec{i}}_{D_k} \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P + x]^{\vec{i}}_{D_k} = [r + x]^{\vec{i}}_{D_k} \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P + x]^{\vec{i}}_{D_k} = \left[(+^{-1}(r, x)) + x\right]^{\vec{i}}_{D_k} \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P + x]^{\vec{i}}_{D_k} = [r]^{\vec{i}}_{D_k} \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ ([P + x]^{\vec{i}}_{D_k}, [x]^{\vec{i}}_{D_k}) = ([r]^{\vec{i}}_{D_k}, [x]^{\vec{i}}_{D_k}) \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ ([P + x]^{\vec{i}}_{D_k}, [x]^{\vec{i}}_{D_k}) = ([(r, x)]^{\vec{i}}_{D_k}) \\
&\Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ ([P + x]^{\vec{i}}_{D_k}, [x]^{\vec{i}}_{D_k}) = [(P + x, x)]^{\vec{i}}_{D_k} \\
&\Leftrightarrow \perp_{D_k} (P + x, x)
\end{aligned}$$

We may easily generalize for any tuple of values. The reduction is indeed polynomial as the size (number of symbols) of $P + x$ is the size of P plus two. ■

Lemma 7.5. *Assume that D_k is at least of size two, and contains a right invertible symbol $+$. There exists $T(P_1, P_2, Q_1, Q_2)$ such that for any $P_1, Q_1, P_2, Q_2 \in \overline{\mathcal{P}}(I, R)$ with $r \in R$,*

$$P_1 | P_2 \approx_{D_k} Q_1 | Q_2 \Leftrightarrow T(P_1, P_2, Q_1, Q_2) \approx_{D_k} r$$

Proof. Let $P_1, Q_1 \in \mathcal{P}(\emptyset, R)$, $P_2, Q_2 \in \overline{\mathcal{P}}(\emptyset, R)$ with $|P_1| = |Q_1| = n$. Consider the following program, $S(P, Q)$:

$$\begin{array}{l}
x \xleftarrow{\$} D; \vec{r} \xleftarrow{\$} D_k^{|R|}; \vec{r'} \xleftarrow{\$} D_k^{|R|}; \\
\text{if } x = 0 \text{ then} \\
\quad \text{if } \neg(P_1(\vec{r}) = \vec{0} \wedge P_2(\vec{r}) = \vec{0}) \text{ then} \\
\quad\quad \mathsf{return}\ 1 \\
\quad \text{else } \mathsf{return}\ 0 \\
\text{else if } x = 1 \\
\quad \text{if}(Q_1(\vec{r}) = \vec{0} \wedge Q_2(\vec{r}) = \vec{0}) \text{ then} \\
\quad\quad \mathsf{return}\ 1 \\
\quad \text{else } \mathsf{return}\ 0 \\
\\
\text{else } \mathsf{return}\ r
\end{array}$$

We fix any $\vec{i}$. Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$. The probability that $S(P, Q)$ returns 1, by case disjunction on the value of x:

$$\frac{1}{|D_k|}(1 - [P]^{\vec{i}}_{D_k}(\vec{0}, \vec{0})) + \frac{1}{|D_k|}([Q]^{\vec{i}}_{D_k}(\vec{0}, \vec{0}))) = \frac{1}{|D_k|} + \frac{[Q]^{\vec{i}}_{D_k}(\vec{0}, \vec{0})) - [P]^{\vec{i}}_{D_k}(\vec{0}, \vec{0}))}{|D_k|}$$

We have,

- $[S(P, Q)]^{\vec{i}}_{Al}(0) = \frac{1}{|D_k|} + \frac{[P]^{\vec{i}}_{D_k}(\vec{0}, \vec{0})) - [Q]^{\vec{i}}_{D_k}(\vec{0}, \vec{0}))}{|D_k|}$
- $[S(P, Q)]^{\vec{i}}_{Al}(1) = \frac{1}{|D_k|} + \frac{[Q]^{\vec{i}}_{D_k}(\vec{0}, \vec{0})) - [P]^{\vec{i}}_{D_k}(\vec{0}, \vec{0}))}{|D_k|}$

- for $c \notin \{0,1\}$, $[S(P,Q)]^{\vec{i}}_{Al}(c) = \frac{1}{|D_k|}$.

Thus, we obtain that

$$S(P,Q) \approx_{D_k} r \Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P]^{\vec{i}}_{D_k}(\vec{0},\vec{0}) = [Q]^{\vec{i}}_{D_k}(\vec{0},\vec{0})$$

We now use the right invertible function. We assume without loss of generality that 0 the is a neutral element for $+$. We have, for a constant $\vec{o} \in D^n$,

$$S((+^{-1}(P_1,c),P_2)(+^{-1}(Q_1,c),Q_2)) \approx_{D_k} r \Leftrightarrow \forall \vec{i} \in D_k^{|X|}.\ [P]^{\vec{i}}_{D_k}(\vec{o},\vec{0}) = [Q]^{\vec{i}}_{D_k}(\vec{o},\vec{0})$$

Thus, if we use $x \in I$ a fresh input variable, with $T(P_1,P_2,Q_1,Q_2) = S((+^{-1}(P_1,x),P_2)(+^{-1}(Q_1,x),Q_2))$ we finally have that

$$T(P_1,P_2,Q_1,Q_2) \approx_{D_k} r \Leftrightarrow \forall \vec{o} \in D^n, \forall \vec{i} \in D_k^{|X|}.\ [P]^{\vec{i}}_{D_k}(\vec{o},\vec{0}) = [Q]^{\vec{i}}_{D_k}(\vec{o},\vec{0})$$

■

Proposition 7.1. *Let $P,Q \in \mathcal{P}(I,R)$.*

$$\begin{aligned} P \approx_{D_k} Q &\Leftrightarrow \forall \vec{i} \in D_k^{|I|}, \forall \vec{o} \in D_k^{|P|}.\ \left|\{\vec{r} \in D_k^{|R|} | [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = \vec{o}\}\right| = \left|\{\vec{r} \in D_k^{|R|} | [\![Q]\!]^{\vec{i},\vec{r}}_{D_k} = \vec{o}\}\right| \\ &\Leftrightarrow \exists f \in \mathsf{bij}^{D_k^{|R|}}, \forall \vec{i} \in D_k.\ [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = [\![Q]\!]^{\vec{i},f(\vec{r})}_{D_k} \end{aligned}$$

Proof. The first equivalence is a direct application of the definition, counting the number of points rather than taking the probability. We focus on proving the second equivalence.
⇐
For any input value $\vec{i}$ and any c, the equality of the deterministic semantics implies equality of the preimage for each c, and then

$$\begin{aligned} \left|\{\vec{r} \in D_k^{|R|} | [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = c\}\right| &= \left|\{\vec{r} \in D_k^{|R|} | [\![Q]\!]^{\vec{i},f(\vec{r})}_{D_k} = c\}\right| \\ &= \left|\{f^{-1}(\vec{r}) \in D_k^{|R|} | [\![Q]\!]^{\vec{i},\vec{r}}_{D_k} = c\}\right| \\ &= \left|f^{-1}\{\vec{r} \in D_k^{|R|} | [\![Q]\!]^{\vec{i},\vec{r}}_{D_k} = c\}\right| \\ &= \left|\{\vec{r} \in D_k^{|R|} | [\![Q]\!]^{\vec{i},\vec{r}}_{D_k} = c\}\right| \end{aligned}$$

⇒
For any input value $\vec{i}$ and any c, we have

$$\left|\{\vec{r} \in D_k^{|R|} | [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = c\}\right| = \left|\{\vec{r} \in D_k^{|R|} | [\![Q]\!]^{\vec{i},\vec{r}}_{D_k} = c\}\right|$$

This means that for each c we can define a bijection $f_c : \{\vec{r} \in D_k^{|R|} | [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = c\} \mapsto \{\vec{r} \in D_k^{|R|} | [\![Q]\!]^{\vec{i},\vec{r}}_{D_k} = c\}$, which verifies $\forall \vec{r} \in \{\vec{r} \in D_k^{|R|} | [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = c\}, [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = c = [\![Q]\!]^{\vec{i},\vec{r}}_{D_k} = c$. Moreover, $g : \vec{r} \in D_k^{|R|} \mapsto [\![P]\!]^{\vec{i},\vec{r}}_{D_k}$ is a function whose image is contained in $D_k^{|P|}$, so

$$D_k^{|R|} = \bigcup_{c \in D_k^{|P|}} (\{\vec{r} \in D_k^{|R|} | [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = c\})$$

i.e., the domains of all the f_c form a partition of $D_k^{|R|}$. Therefore the function f, defined as the union of all the f_c, is a bijective function that verifies $\forall \vec{r} \in D_k^{|R|}, [\![P]\!]^{\vec{i},\vec{r}}_{D_k} = [\![Q]\!]^{\vec{i},f(\vec{r})}_{D_k}$. ■

C.2 Proofs of Chapter 8

Corollary C.1. *For any $k \in \mathbb{N}$, q^k-*equivalence *and q^k-*conditional equivalence *restricted to programs of fixed arity and without inputs are in $\mathsf{C_=P}$.*

Proof. We only have to consider *q^k-conditional equivalence* as it is harder than *q^k-equivalence*. Let $P_1, Q_1 \in \mathcal{P}_q(\emptyset, R)$,$P_2, Q_2 \in \overline{\mathcal{P}}_q(\emptyset, R)$ with $|P_1| = |Q_1| = 1$.

For some c, we have that verifying if

$$[P_1, P_2]_{\mathbb{F}_{q^k}}(c, \mathbf{0}) = [Q_1, Q_2]_{\mathbb{F}_{q^k}}(c, \mathbf{0})$$

is in $\mathsf{C_=P}$. As $\mathsf{C_=P}$ is closed under finite intersection [Tor88], we can decide in $\mathsf{C_=P}$ if:

$$\bigwedge_{c \in \mathbb{F}_q} [P_1, P_2]_{\mathbb{F}_{q^k}}(c, \mathbf{0}) = [Q_1, Q_2]_{\mathbb{F}_{q^k}}(c, \mathbf{0})$$

This is exactly the definition of conditional equivalence, and thus it concludes the proof. ■

Corollary 8.1. *For any $k \in \mathbb{N}$, $\mathbb{F}_{q^k}$-*equivalence *and $\mathbb{F}_{q^k}$-*conditional equivalence *are in $\mathsf{coNP}^{\mathsf{C_=P}}$.*

Proof. First, we only have to consider $\mathsf{C{-}EQUIV}_q$ as it is a generalization of equivalence. Next, we only have to consider $\mathsf{C{-}EQUIV}_q$ restricted to program without inputs with Lemma 7.1. Let $P_1, Q_1 \in \mathcal{P}_q(\emptyset, R)$,$P_2, Q_2 \in \overline{\mathcal{P}}_q(\emptyset, R)$ with $|P_1| = |Q_1| = n$.

Now,

$$P_1 \mid P_2 \approx_{q^k} Q_1 \mid Q_2 \Leftrightarrow \forall c \in \mathbb{F}_{q^k}^n [P_1, P_2]_{\mathbb{F}_{q^k}}(c, \mathbf{0}) = [Q_1, Q_2]_{\mathbb{F}_{q^k}}(c, \mathbf{0})$$

For some $c \in \mathbb{F}_{q^k}^n$, we have that deciding if

$$[P_1, P_2]_{\mathbb{F}_{q^k}}(c, \mathbf{0}) = [Q_1, Q_2]_{\mathbb{F}_{q^k}}(c, \mathbf{0})$$

is in $\mathsf{C_=P}$.

The decision problem is then directly in $\mathsf{coNP}^{\mathsf{C_=P}}$. ■

Lemma 7.2. *Let $P_1, \ldots, P_n$ be programs over $\mathcal{P}(I, R)$, and $Y \subset R$.*

$$\perp^Y_{D_k}(P_1, \ldots, P_n) \Leftrightarrow \perp_{D_k}(P_1\sigma, \ldots, P_n\sigma)$$

where $\sigma : Y \to I_Y$ is the substitution that replaces each variable in Y by a fresh input variable in I_Y.

Proof.

$$\begin{aligned}
\perp^Y_{q^k}(P_1, \ldots, P_n) &\Leftrightarrow \forall \vec{i} \in \mathbb{F}_{q^k}^{|X|}, \forall \vec{i'} \in \mathbb{F}_{q^k}^{|Y|}.\ [\![P_1, \ldots, P_n]\!]\mathbb{F}_{q^k}\vec{i}, \vec{i'} = ([\![P_1]\!]\mathbb{F}_{q^k}\vec{i}, \vec{i'}, \ldots, [\![P_n]\!]\mathbb{F}_{q^k}\vec{i}, \vec{i'}) \\
&\Leftrightarrow \forall \vec{i} \in \mathbb{F}_{q^k}^{|I \uplus I_Y|}.\ [\![(P_1, \ldots, P_n)\sigma]\!]\mathbb{F}_{q^k}\vec{i} = ([\![P_1\sigma]\!]\mathbb{F}_{q^k}\vec{i}, \ldots, [\![P_n\sigma]\!]\mathbb{F}_{q^k}\vec{i}) \\
&\Leftrightarrow \cong_{q^k}(P_1\sigma, \ldots, P_n\sigma)
\end{aligned}$$

■

Lemma 7.3. *Let $P_1, \ldots, P_n$ be programs over $\mathcal{P}(I, \{r_1, \ldots, r_m\})$*

$$\perp_{D_k}(P_1, \ldots, P_n) \Leftrightarrow (P_1, \ldots, P_n) \approx_{D_k} (P_1\sigma 1, \ldots, P_n\sigma_n)$$

where σ_i is the substitution that to any r_j associates a fresh random variable r^i_j.

Proof.

$$\begin{aligned}\cong_{q^k}(P_1,\dots,P_n) &\Leftrightarrow \forall \vec{i} \in \mathbb{F}_{q^k}^{|X|}.\ [\![P_1,\dots,P_n]\!]\mathbb{F}_{q^k}\vec{i} = ([\![e_1]\!]\mathbb{F}_{q^k}\vec{i},\dots,[\![P_n]\!]\mathbb{F}_{q^k}\vec{i})\\ &\Leftrightarrow \forall \vec{i} \in \mathbb{F}_{q^k}^{|X|}.\ [\![P_1,\dots,P_n]\!]\mathbb{F}_{q^k}\vec{i} = ([\![P_1\sigma_1]\!]\mathbb{F}_{q^k}\vec{i},\dots,[\![P_n\sigma_n]\!]\mathbb{F}_{q^k}\vec{i})\\ &\Leftrightarrow \forall \vec{i} \in \mathbb{F}_{q^k}^{|X|}.\ [\![P_1,\dots,P_n]\!]\mathbb{F}_{q^k}\vec{i} = [\![(P_1\sigma_1,\dots,P_n\sigma_n)]\!]\mathbb{F}_{q^k}\vec{i}\\ &\Leftrightarrow (P_1,\dots,P) \approx_{q^k} (P_1\sigma_1,\dots,P_n\sigma_n)\end{aligned}$$

Indeed, for any $\vec{i} \in \mathbb{F}_{q^k}^{|X|}$ we have that $[\![P_i]\!]\mathbb{F}_{q^k}\vec{i} = [\![P_i\sigma_i]\!]\mathbb{F}_{q^k}\vec{i}$ as we are only performing renaming. Moreover, $P_1\sigma_1,\dots,P_n\sigma_n$ do not share any variable, and thus trivially verify:

$$([\![P_1\sigma_1]\!]\mathbb{F}_{q^k}\vec{i},\dots,[\![P_n\sigma_n]\!]\mathbb{F}_{q^k}\vec{i}) = [\![(P_1\sigma_1,\dots,P_n\sigma_n)]\!]\mathbb{F}_{q^k}\vec{i}$$

■

Theorem 8.1. $\mathbb{F}_{q^k}$-conditional independence *is* $\mathsf{coNP}^{\mathsf{C}_=\mathsf{P}}$*-complete.*

Proof. Only the hardness remains. Given a CNF formula $\phi(I,R)$ over two sets of variables and $(\vee,\wedge)$ we set $P = \phi' \in \mathcal{P}_2(I,R)$ obtained according to Lemma C.1. With r a fresh random variable, recall that:

$$\begin{aligned}P \approx_2 r &\Leftrightarrow \text{for all valuation of } I,\ \phi \text{ is true for half of the valuation of } R\\ &\Leftrightarrow \phi(I,R) \in \mathsf{A{-}halfSAT}\end{aligned}$$

But, with x a fresh deterministic variable and r' a fresh random variable:

$$\begin{aligned}P \approx_2 r &\Leftrightarrow P + x \approx_2 r + x\\ &\Leftrightarrow P + x \approx_2 r\\ &\Leftrightarrow (P + r', r') \approx_2 (r, r')\\ &\Leftrightarrow \perp_2^{\emptyset} (P + r', r')\end{aligned}$$

And thus, we conclude with:

$$\perp_2^{\emptyset} (P + r, r) \Leftrightarrow \phi(I,R) \in \mathsf{A{-}halfSAT}$$

■

Lemma 8.3. *For any* $k \in \mathbb{N}$, $\mathbb{F}_{q^k}$-0-majority *reduces in polynomial time to* $\mathbb{F}_{q^k}$-0-majority *with* $r = 1$.

Proof. Indeed, for any n, let us denote by D_n any subset of $\mathbb{F}_q^m$, where $m = |\frac{n}{q}|$, such that $|D_n| = n$. If we denote by d_n a fixed element of D_m, let T_n be the program:

$$\begin{array}{l} r_1,\dots,r_m \xleftarrow{\$} \{x \in \mathbb{F}_q^m \mid \bigvee_{d\in D_n} x = d\}\\ \textbf{if } (r_1,\dots,r_m) = d_n \textbf{ then}\\ \quad \mathsf{return}\ \vec{0}\\ \textbf{else}\\ \quad \mathsf{return}\ \vec{1}\end{array}$$

Notice that by construction $[T_n]_{\mathbb{F}_{q^k}}(\mathbf{0}) = \frac{1}{n}$. This is only the most naive version of this encoding, simpler polynomials can be found for many specific cases. And finally, for any $r, s \in \mathbb{N}$, assuming

the probabilities are non zero, we have:

$$\begin{aligned}\forall \vec{i} \in \mathbb{F}_{q^k}^{|I|}.\ \frac{[\![P]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0})}{[\![Q]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0})} \leq \frac{r}{s} &\Leftrightarrow \forall \vec{i} \in \mathbb{F}_{q^k}^{|I|}.\ \frac{[\![P]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0})}{r} \leq \frac{[\![Q]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0})}{s} \\ &\Leftrightarrow \forall \vec{i} \in \mathbb{F}_{q^k}^{|I|}.\ [\![P]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0})[\![T_r]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0}) \leq [\![Q]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0})[\![T_s]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0}) \\ &\Leftrightarrow \forall \vec{i} \in \mathbb{F}_{q^k}^{|I|}.\ [\![(P,T_r)]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0}) \leq [\![(Q,T_s)]\!]\mathbb{F}_{q^k}\vec{i}(\mathbf{0}) \\ &\Leftrightarrow (P,T_r) \prec_{q^k} (Q,T_s)\end{aligned}$$

■

Lemma 8.4. *For any $k \in \mathbb{N}$, $\mathbb{F}_{q^k}$-0-majority restricted to inputless programs is in* PP.

Proof. Let $P, Q \in \mathcal{P}_q(\emptyset, R)$. Let us reuse the polynomial time Turing Machine M defined in Lemma 8.1. Given P_1, P_2, Q_1, Q_2 and $\vec{o}$, it was such that:

$$[P_1, P_2]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) = [Q_1, Q_2]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) \Leftrightarrow M \text{ accepts exactly half of the time}$$

Now, by replacing equals by $>$ signs in the proof, we directly have that:

$$[P_1, P_2]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) \leq [Q_1, Q_2]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) \Leftrightarrow M \text{ accepts at least half of the time}$$

Thus, we do have:

$$\begin{aligned}P \prec_{q^k} Q &\Leftrightarrow [P, 0]_{\mathbb{F}_{q^k}}(\mathbf{0}, 0) \leq [Q]_{\mathbb{F}_{q^k}}(0, \mathbf{0}) \\ &\Leftrightarrow M \text{ accepts at least half of the time on input } (P, 0, Q, 0, \mathbf{0})\end{aligned}$$

■

Lemma 8.5. $\mathbb{F}_2$-0-majority *is* PP*-hard (even for inputless programs).*

Proof. We show PP-hardness by reduction from MAJSAT. Given a CNF formula $\phi(R)$ over two sets of variables and $(\vee, \wedge)$ we set $P = \phi' \in \mathcal{P}_2(R)$ obtained according to Lemma C.1. We then have:

$$\begin{aligned}\phi \in \mathsf{MAJSAT} &\Leftrightarrow |X \in \mathbb{F}_2^m \mid P(X) = \mathbf{0}| \leq 2^{m-1} \\ &\Leftrightarrow |X \in \mathbb{F}_2^m \mid P(X) = \mathbf{0}| \leq |X \in \mathbb{F}_2^m \mid x_1 = 0| \\ &\Leftrightarrow P \prec_2 x_1\end{aligned}$$

■

Lemma 8.6. $\mathbb{F}_{q^k}$-majority *is* $\mathsf{coNP}^{\mathsf{PP}}$ *complete.*

Proof. Hardness Let ϕ a CNF formula built over two sets of variables I and R. We use the same construction as in Lemma 8.5 to obtain a polynomial $P \in \mathcal{P}_2(I, R)$ whose truth value is equivalent of ϕ.

We have, for some variable r in R:

$$\phi \in \mathsf{A{-}MINSAT} \Leftrightarrow r \prec_2 P$$

Membership

Let $P, Q \in \mathcal{P}_{q^k}(I, R)$. We slightly modify M from Lemma 8.4, so that it takes as extra argument a valuation for the variables in I, and every evaluation of P or Q is made according to the valuation.

Then, we directly have:

$$P \prec_q Q \Leftrightarrow \forall \vec{i} \in \mathbb{F}_p^{|I|}, M \text{ accepts with probability greater than half on input } \vec{i}$$

This problem is then directly in $\mathsf{coNP}^{\mathsf{PP}}$. ■

Lemma 8.7. $\mathbb{F}_{q^\infty}$-equivalence *restricted to linear programs is in* PTIME.

Proof. Without loss of generality, we only consider programs without input variables (Lemma 7.1).

Given a set of variables R, we assume that there is an ordering over the variables in R. We say that an expression is in normal form if it is of one of the following form: 0 or 1, or e, or $1 \oplus e$, where e is built from variables and $\oplus$ (but no constants), and variables appear at most once in increasing order.

Every linear expression can easily be put in normal form, using the commutativity of $\oplus$, and the normal form is indeed unique thanks to the ordering on variables.

We now assume that all polynomials are in normal form.

Given $P_1, \dots, P_n \in \overline{\mathcal{P}}_q(\emptyset, R)$ without multiplications, we iterate over each P_i, where, after initializing a set S to the empty set:

- if $\mathsf{vars}(P_i) \cap S \neq \emptyset$, let $r = \mathsf{min}(\mathsf{vars}(P_i) \cap S)$ and:
  - replace P_i by r;
  - set $S := S \cup \{r\}$;
  - for each $j \geq i$, replace P_j by $P_j[P_i \oplus r/r]$.
- else, continue.

This produces a normal form for any tuple $(P_1, \dots, P_n)$, where each P_i is either a fresh random variable (not appearing in the previous Ps), or a linear combination of the previous $P_1, \dots, P_{i-1}$.

Finally, two programs are universally equivalent if and only if they have the same normal form (up to α-renaming). Indeed, if they have the same normal form, they are trivially universally equivalent. Now, if they do not have the same normal form, there exists some i such that P_i and Q_i are two different expressions, and this imply non equivalence.

This basic decision procedures gives us a $\mathcal{O}(n \times |R|)$ complexity. Indeed, we treat each polynomial P_i or Q_i only once, first to apply the currently known substitutions, and then to transform it into a fresh random if required. Applying the currently known substitutions may take up to $|R|$ loops, hence the considered complexity. ■

Corollary 8.5. $\mathbb{F}_{q^\infty}$-conditional equivalence *and* $\mathbb{F}_{q^\infty}$-equivalence *restricted to arithmetic programs are in* EXP.

Proof. [LW06, Corollary 2] provides a precise complexity for the evaluation of $Z(P)$. They provide an algorithm to compute $Z(P)$ for which there exist an explicit polynomial R such that it runs in time $R(p^m k^m d^{m^2} 2^n))$, where d is the sum of the degrees of the P^i. It is then polynomial in the degrees of the polynomials and the size of the finite fields, but exponential in the number of

variables. In our, case, we need to compute three times Z, on polynomials depending over $2m$ variables (has we duplicate variables), which gives us en exponential in the size of our arithmetic programs.

■

Lemma C.1. *Given a CNF formula $\phi(I,R)$ over two sets of variables and $(\vee,\wedge)$, we can produce in polynomial time a program $P \in \mathcal{P}_2(I,R)$ equivalent to ϕ.*

Proof. Given a CNF formula $\phi(I,R)$ over two sets of variables and $(\vee,\wedge)$ we transform ϕ into an equivalent formula ϕ' over $I \uplus R$ and $\oplus,\wedge$ in polynomial time w.r.t the size of the formula. Indeed, given a clause of ϕ of the form $x \vee y \vee z$, we have that $x \vee y \vee z = (x \oplus y \oplus xy) \vee z = (x \oplus y \oplus xy) \oplus z \oplus (x \oplus y \oplus xy)z = x \oplus y \oplus xy \oplus z \oplus xz \oplus yz \oplus xyz = x \oplus y \oplus z \oplus xy \oplus yz \oplus xz \oplus xyz$. With this transformation, we have $|\phi'| \leq 5 \times |\phi|$.

And then, $P = \phi' \in \mathcal{P}_2(I,R)$ is a program equivalent to ϕ. ■

Lemma C.2. *Let $P,Q \in \overline{\mathcal{P}}_2(\emptyset,R)$ without any multiplication.*

$$P \approx_2 Q \Leftrightarrow P \approx_{\mathbb{F}_{2^\infty}} Q$$

Proof.
$\Leftarrow$ Trivial direction.
$\Rightarrow$ As outlined in [BDK+10], one can decide if $P \approx_2 Q$ by constructing a bijection represented by only linear terms (thanks to the weak primality of $\mathbb{F}_2$ restricted to addition). We thus have a bijection σ without multiplication such that $P = Q\sigma$. σ is then a bijection over all $\mathbb{F}_{2^k}$, and we do have $P \approx_{\mathbb{F}_{2^\infty}} Q$.

■

Lemma C.3. *Let b be a propositional formula built over built over atoms of the form $B = 0$ or $B \neq 0$ with $B \in \mathbb{F}_q[X]$. There exists $X' \supset X$ and polynomials $B_1,\ldots,B_n \in \mathbb{F}_q[X']$ so that:*

$$\left|X \in \mathbb{F}_{q^k}^m \mid b\right| = \left|X' \in \mathbb{F}_{q^k}^m \mid \bigwedge_{1\leq i\leq n} B_i = 0\right|$$

Those polynomials can be computed in exponential time.

Proof. We prove by induction of the formula that for any formula b, there exists polynomials $B_1,\ldots,B_n$ so that:

$$\left|X \in \mathbb{F}_{q^k}^m \mid b\right| = \left|X' \in \mathbb{F}_{q^k}^m \mid \bigwedge_{1\leq i\leq n} B_i = 0\right|$$

We will assume that the formula are in conjunctive normal form, hence the exponential time.
$\underline{b := B = 0}$ Direct, with $X' = X$ and $B_1 = B$.

$\underline{b := B' \neq 0}$ For any k and c we have that:

$$\left|X \in \mathbb{F}_{q^k}^m \mid B \neq 0\right| = \left|X \in \mathbb{F}_{q^k}^m, t \in \mathbb{F}_{q^k} \mid tB - 1 = 0\right|$$

Indeed, B is different from zero if and only if it is invertible, and thus if and only if there exist a single value t such that $tB = 1$.
$b := \bigvee_{1\leq i\leq l} B_i = 0$

$$\left|X \in \mathbb{F}_{q^k}^m \mid \bigvee_{1\leq i\leq l} B_i = 0\right| = \left|X \in \mathbb{F}_{q^k}^m \mid (\prod_{1\leq i\leq l} B_i) = 0\right|$$

$b := \bigwedge_{1\leq i\leq k} b_i$ By induction hypothesis on each b_i we get $B_1^i, \ldots, B_{n_i}^i$ so that all of them verify:

$$\left|X \in \mathbb{F}_{q^k}^m \mid b\right| = \left|X \in \mathbb{F}_{q^k}^m, t \in \mathbb{F}_{q^k} \mid \bigwedge_{1\leq i\leq k} \bigwedge_{1\leq j\leq n_i} B_j^i = 0\right|$$

$b := b_1 \vee b_2$ By induction hypothesis on b_1 we get $B_1, \ldots, B_n$, and on b_2 $B'_1, \ldots, B'_n$, which satisfies

$$\left|X \in \mathbb{F}_{q^k}^m \mid b\right| = \left|X \in \mathbb{F}_{q^k}^m, t \in \mathbb{F}_{q^k} \mid \bigwedge_{1\leq i\leq n} B_i = 0 \vee \bigwedge_{1\leq i\leq n} B'_i = 0\right|$$

■

Lemma 8.11. *For any $k \in \mathbb{N} \cup \{\infty\}$, $\mathbb{F}_{q^k}$-*conditional equivalence *restricted to programs without failures reduces in exponential time to $\mathbb{F}_{q^k}$-*conditional equivalence *restricted to arithmetic programs.*

Proof. Let $P_1, Q_1 \in \mathcal{P}_q(\emptyset, R)$,$P_2, Q_2 \in \overline{\mathcal{P}}_q(\emptyset, R)$, without failures.

We reason by induction on the total number n of conditional branching in P_1 and Q_1. By basic transformations of the conditionals, we can assume that all conditions are of the form $B \neq 0$ (one can easily encode negations, conjunction and disjunction using conditionals branching).
$n = 0$ If there are no conditionals branching, the result is trivial.

$n > 1$ We consider one of the inner most branching in P_1, i.e $P_1 := C[\textbf{if } B \neq 0 \textbf{ then } P_1^t \textbf{ else } P_1^f]$ for some context C, and P_1^t, P_1^f arithmetic programs.

For a fixed k, we have a classical encoding of the if then else in polynomials (cf CSF19):

$$\left[\textbf{if } B \neq 0 \textbf{ then } P_1^t \textbf{ else } P_1^f\right]_{\mathbb{F}_{q^k}} = \left[P_1^f + B^{q^k-1}(P_1^t - P_1^f)\right]_{\mathbb{F}_{q^k}}$$

We then have that:

$$C[\textbf{if } B \neq 0 \textbf{ then } P_1^t \textbf{ else } P_1^f] \mid P_2 \approx_{q^k} Q_1 \mid Q_2 \Leftrightarrow C[P_1^f + B^{q^k-1}(P_1^t - P_1^f)] \mid P_2 \approx_{q^k} Q_1 \mid Q_2$$

A difficulty of this encoding is that it depends on the k, so it cannot be lifted to universal conditional equivalence. However, we can remove this difficulty by using an extra variable t to encode the B^{q^k-1}.

With t a fresh variable, we write

$$\mathsf{if}\ B, P_1^t, P_1^f\ \mathsf{then}\ =\ \mathsf{else}\ (P_1^f + tB(P_1^t - P_1^f), B(Bt-1), t(Bt-1))$$

Now, for any k and c we have that:

$$\begin{aligned}
&\left[(P_1^f + B^{q^k-1}(P_1^t - P_1^f), P_2)\right]_{\mathbb{F}_{q^k}} (c, \mathbf{0}) \\
&\quad = \left|X \in \mathbb{F}_{q^k}^m \mid P_1^f + B^{q^k-1}(P_1^t - P_1^f) = c \wedge P_2 = \mathbf{0}\right| \times \tfrac{1}{\left|\mathbb{F}_{q^k}^m\right|} \\
&\quad = \left|X \in \mathbb{F}_{q^k}^m, t \in \mathbb{F}_{q^k} \mid \mathsf{if}\ B, P_1^t, P_1^f\ \mathsf{then}\ =\ \mathsf{else}\ (c, \mathbf{0}) \wedge P_2 = \mathbf{0}\right| \times \tfrac{1}{\left|\mathbb{F}_{q^k}^m\right|}
\end{aligned}$$

Indeed, for any variable t and polynomial B:

$$(B(Bt-1) = 0 \wedge t(Bt-1) = 0) \Leftrightarrow t = B^{q^k-2}$$

Finally:

$$\begin{aligned}
&\left[(P_1^f + B^{q^k-1}(P_1^t - P_1^f), P_2)\right]_{\mathbb{F}_{q^k}} (c, \mathbf{0}) \\
&\quad = \left|X \in \mathbb{F}_{q^k}^m, t \in \mathbb{F}_{q^k} \mid \mathsf{if}\ B, P_1^t, P_1^f\ \mathsf{then}\ =\ \mathsf{else}\ (c, \mathbf{0}) \wedge P_2 = \mathbf{0}\right| \times \tfrac{1}{\left|\mathbb{F}_{q^k}^m\right| + |\mathbb{F}_q|} \\
&\quad = \left[(P_1^f + tB(P_1^t - P_1^f), B(Bt-1), t(Bt-1), P_2)\right]_{\mathbb{F}_{q^k}} (c, \mathbf{0})
\end{aligned}$$

Putting everything together, we get that:

$$\begin{aligned}
C[\mathbf{if}\ B \neq 0\ \mathbf{then}\ P_1^t\ \mathbf{else}\ P_1^f] \mid P_2 \approx_{q^k} Q_1 \mid Q_2 &\Leftrightarrow C[P_1^f + B^{q^k-1}(P_1^t - P_1^f)] \mid P_2 \approx_{q^k} Q_1 \mid Q_2 \\
&\Leftrightarrow C[P_1^f + tB(P_1^t - P_1^f)] \mid (B(Bt-1), t(Bt-1), P_2) \approx_{q^k} Q_1 \mid Q_2
\end{aligned}$$

And we finally have:

$$P_1 \mid P_2 \approx_{\mathbb{F}_{q^\infty}} Q_1 \mid Q_2 \Leftrightarrow C[P_1^f + tB(P_1^t - P_1^f)] \mid (B(Bt-1), t(Bt-1), P_2) \approx_{\mathbb{F}_{q^\infty}} Q_1 \mid Q_2$$

The conditional equivalence on the right-side contains strictly one less conditional, we thus conclude by induction hypothesis.

Conclusion We have shown by induction that we can remove all conditional branching. Each removal produces a new instance of polynomial size, and there is necessarily a polynomial number of conditional branching in the programs. We thus reduces in exponential time $\mathsf{C{-}EQUIV}_{q^\infty}$ to $\mathsf{C{-}EQUIV}_{q^\infty}$ over programs without conditionals (recall that removing the failure cost an exponential). ■

Lemma 8.12. *For any $k \in \mathbb{N} \cup \{\infty\}$, $\mathbb{F}_{q^k}$-conditional equivalence reduces to $\mathbb{F}_{q^k}$-conditional equivalence restricted to programs without failures in exponential time.*

Proof. Let $P_1, Q_1 \in \mathcal{P}_q(\emptyset, R)$,$P_2, Q_2 \in \overline{\mathcal{P}}_q(\emptyset, R)$.

Recall that observe are expressed using conditionals with a failure branch, and that sampling in some specific set can be encoded using the observe primitive. Without loss of generality, we can

consider that $\bot$ appears only once, as we can merge the conditions of the different failure branches in a single one.

Then, P_1 is of the form $P_1 := \textbf{if } b \textbf{ then } P_1^t \textbf{ else } \bot$ for some program P_1^t which cannot fail.

Now, with Lemma C.3, we have $R' \supset R$ and polynomials $B_1, \dots, B_n \in \mathbb{F}_q[R']$ so that:

$$\left|R \in \mathbb{F}_{q^k}^m \mid b\right| = \left|R' \in \mathbb{F}_{q^k}^m \mid \bigwedge_{1 \leq i \leq n} B_i = 0\right|$$

And then:

$$\begin{aligned}
&[(\textbf{if } b \textbf{ then } P_1^t \textbf{ else } \bot, P_2)]_{\mathbb{F}_{q^k}}(\vec{o}, \mathbf{0}) \\
&\quad = \tfrac{\mathbb{P}\{P_1^t = \vec{o} \wedge P_2 = \mathbf{0} \wedge b\}}{\mathbb{P}\{\neg b\}} \\
&\quad = \left|R' \in \mathbb{F}_{q^k}^m \mid P_1^t = \vec{o} \wedge P_2 = \mathbf{0} \wedge \textstyle\bigwedge_{1 \leq i \leq n} B_i = 0\right| \times \tfrac{1}{\left|R' \in \mathbb{F}_{q^k}^m \mid \neg \bigwedge_{1 \leq i \leq n} B_i = 0\right|} \\
&\quad = \left|R' \in \mathbb{F}_{q^k}^m \mid P_1^t = \vec{o} \wedge P_2 = \mathbf{0} \wedge \textstyle\bigwedge_{1 \leq i \leq n} B_i = 0\right| \times \tfrac{1}{\left|R' \in \mathbb{F}_{q^k}^m, t_i \in \mathbb{F}_q \mid \prod_{1 \leq i \leq n}(t_i B_i - 1) = 0\right|}
\end{aligned}$$

This allows us to conclude, when σ maps random variables to fresh ones, that:

$$\textbf{if } b \textbf{ then } P_1^t \textbf{ else } \bot \mid P_2 \approx_{q^k} Q_1 \mid Q_2 \Leftrightarrow P_1^t \mid P_2, B_1, \dots, B_n \approx_{q^k} Q_1 \mid Q_2, \textstyle\prod_{1 \leq i \leq n}(t_i B_i \sigma - 1)$$

We thus removed the failure on the left side of the conditional equivalence. Proceeding similarly on the right side yield the expected result. ■

Lemma 8.13. $\mathbb{F}_q$-equivalence *reduces in polynomial time to* $\mathbb{F}_{q^\infty}$-equivalence.

Proof. Let $P, Q \in \mathcal{P}_q(\emptyset, \{r_1, \dots, r_m\})$. We directly have:

$$\begin{aligned}
P \approx_q Q \Leftrightarrow\ & \left|X \in \mathbb{F}_q^m \mid P(X) = \mathbf{0}\right| = \left|X \in \mathbb{F}_q^m \mid Q(X) = \mathbf{0}\right| \\
\Leftrightarrow\ & \textbf{if } \textstyle\bigwedge_{1 \leq i \leq m}(\bigvee_{c \in \mathbb{F}_q} r_i = c) \textbf{ then } P \textbf{ else } \mathbf{0} \\
& \qquad \approx_{\mathbb{F}_{2^\infty}} \\
& \textbf{if } \textstyle\bigwedge_{1 \leq i \leq m}(\bigvee_{c \in \mathbb{F}_q} r_i = c) \textbf{ then } Q \textbf{ else } \mathbf{0}
\end{aligned}$$

■

Lemma 8.14. $\mathbb{F}_{q^\infty}$*-0*-majority *restricted to linear programs is in* PTIME.

Proof. We show that for linear programs $P \prec^r_{q^k} Q$ implies that $P \approx_{q^k} Q$. Thus, universal majority is decidable, as universal equivalence is decidable for linear programs (and in PTIME).

Given $P_1, \dots, P_n \in \overline{\mathcal{P}}_q(\emptyset, R)$ without multiplications, let us consider once again the normal form for linear programs. In this normal form, each P_i is either a random r_i, or a linear combination of some r_j, with $j < i$. Let I_P be the set of indices i such that $P_i = r_i$. We denote $P = (P_1, \dots, Pn)$, and given $\vec{o} \in \mathbb{F}_{q^k}^n$, we have that $[P]_{\mathbb{F}_{q^k}}(\vec{o}) = \begin{cases} \frac{1}{q^{k \times |I_P|}} & \text{if the linear constraints are satisfiable} \\ 0 & \text{else} \end{cases}$

Indeed, $\vec{o}$ imposes the values of each r_i for $i \in I$, and then for those values, either the other elements of the program coincides, and if they do not, the program is never equal to $\vec{o}$.

Let $P, Q \in \overline{\mathcal{P}}_q(\emptyset, R)$ without multiplications, we know that:

1. $\forall \vec{o} \in \mathbb{F}_{q^k}^n, [P]_{\mathbb{F}_{q^k}}(\vec{o}) = \frac{1}{q^{k \times |I_P|}}$ or 0
2. $\forall \vec{o} \in \mathbb{F}_{q^k}^n, [Q]_{\mathbb{F}_{q^k}}(\vec{o}) = \frac{1}{q^{k \times |I_Q|}}$ or 0
3. $\sum_{\vec{o} \in \in \mathbb{F}_{q^k}^n} [P]_{\mathbb{F}_{q^k}}(\vec{o}) = \sum_{\vec{o} \in \in \mathbb{F}_{q^k}^n} [Q]_{\mathbb{F}_{q^k}}(\vec{o})$

Now, let us assume that there exists $\vec{o}$ such that $[P]_{\mathbb{F}_{q^k}}(\vec{o}) = 0$ and $[Q]_{\mathbb{F}_{q^k}}(\vec{o}) \neq 0$. Then, for any r, we have $Q \not\prec_{q^k}^r P$. Moreover, if for all $\vec{c'} \neq \vec{o}$, $[P]_{\mathbb{F}_{q^k}}(\vec{c'}) = 0$ or $[Q]_{\mathbb{F}_{q^k}}(\vec{c'}) \neq 0$, it yields a contradiction with Hypothesis (3). Thus, there exists $\vec{c'}$ such that $[P]_{\mathbb{F}_{q^k}}(\vec{c'}) \neq 0$ and $[Q]_{\mathbb{F}_{q^k}}(\vec{c'}) = 0$. This also implies that for all r, $P \not\prec_{q^k}^r Q$.

Let us assume that for all k, $P \prec_{q^k}^r Q$. Then, by the previous development, we know that for all $\vec{o}$, $[P]_{\mathbb{F}_{q^k}}(\vec{o}) \neq 0$ and $[Q]_{\mathbb{F}_{q^k}}(\vec{o}) \neq 0$. If $|I_P| \neq |I_Q|$, it would yield a contradiction with Hypothesis (3). We thus conclude that $|I_P| = |I_Q|$, and based on Hypothesis (1) and (2), we have that for all $\vec{o}$, $[P]_{\mathbb{F}_{q^k}}(\vec{o}) = [P]_{\mathbb{F}_{q^k}}(\vec{o})$. We thus conclude that $P \approx_{q^k} Q$.

We have proven that $P \prec_{q^\infty}^r Q \Leftrightarrow P \approx_{q^\infty} Q$, when restricted to linear programs without multiplications. ■

Lemma 8.15. $\mathbb{F}_{2^\infty}$-0-majority *is* PP-*hard.*

Proof. We prove that 2-0-*majority* reduces to 2^∞-0-*majority* in polynomial time.

Let $P, Q \in \mathcal{P}_2(\emptyset, R)$.

$$
\begin{aligned}
P \prec_2 Q &\Leftrightarrow |X \in \mathbb{F}_2^m \mid P(X) = \mathbf{0}| \leq |X \in \mathbb{F}_2^m \mid q(X) = \mathbf{0}| \\
&\Leftrightarrow \forall k, \left|X \in \mathbb{F}_{2^k}^m \mid P(X) = \mathbf{0} \wedge X \in \mathbb{F}_2^m\right| \leq \left|X \in \mathbb{F}_{2^k}^m \mid Q(X) = \mathbf{0} \wedge X \in \mathbb{F}_2^m\right| \\
&\Leftrightarrow \forall k, \left|X \in \mathbb{F}_{2^k}^m \mid P(X) = \mathbf{0} \wedge x_1(x_1+1) = 0 \wedge \cdots \wedge x_m(x_m+1) = 0\right| \\
&\qquad \leq \left|X \in \mathbb{F}_{2^k}^m \mid Q(X) = \mathbf{0} \wedge x_1(x_1+1) = 0 \wedge \cdots \wedge x_m(x_m+1) = 0\right| \\
&\Leftrightarrow \forall k, \left|X \in \mathbb{F}_{2^k}^m \mid (P(X), x_1(x_1+1), \ldots, x_m(x_m+1)) = \mathbf{0}\right| \\
&\qquad \leq \left|X \in \mathbb{F}_{2^k}^m \mid (Q(X), x_1(x_1+1), \ldots, x_m(x_m+1)) = \mathbf{0}\right| \\
&\Leftrightarrow (P, x_1(x_1+1), \ldots, x_m(x_m+1)) \prec_2^\infty (Q, x_1(x_1+1), \ldots, x_m(x_m+1))
\end{aligned}
$$

■

C.3 Proofs of Section 9.4.1

Given a multilinear map $\hat{e}$, $g_1, .., g_n, g_t$ a set of groups generators, let X be a set of public names sampled in $\mathbb{F}_q$, Y be a set of private names sampled in $\mathbb{F}_q$, $f_1, ...f_k, h \in \mathbb{K}[X, Y]$ be a set of polynomials over both public and secret names and Γ be a coherent set of axioms.

Our deducibility problem is to decide if $\Gamma \models X, g_{i_1}^{f_1}, ..., g_{i_k}^{f_k} \vdash_{\mathcal{E}} g_t^h$ Without loss of generality, we consider here the case of a bilinear map, to simplify the writing, but the proofs scale up to multilinear maps.

C.3.1 Saturation into the Target Group

First, we reduce our problem to the case of a single group. This result comes from the Proposition 1 of [KMT12]. Their constructive proof can trivially be used to obtain the following proposition:

Proposition C.1. *For any sets X and Y, polynomials $f_1, ...f_n, h \in \mathbb{K}[X, Y]$ and groups elements $g_{i_1}^{f_1}, ..., g_{i_n}^{f_n}$, we denote*

$$\begin{aligned}(g_t^{e_i}) = \quad & \{\hat{e}(g_{i_j}, g_{i_k}) | 1 \leq j \leq k \leq n, g_{i_j} \in \mathbb{G}_1, g_{i_k} \in \mathbb{G}_2\} \\ & \cup \{\hat{e}(g_{i_j}, 1) | 1 \leq j \leq n, g_{i_j} \in \mathbb{G}_1, \} \\ & \cup \{\hat{e}(1, g_{i_j}) | 1 \leq j \leq n, g_{i_j} \in \mathbb{G}_2, \}\end{aligned}$$

Then $\Gamma \models X, g_{i_1}^{f_1}, ..., g_{i_n}^{f_n} \vdash_{\mathcal{E}} g_t^h \Leftrightarrow \Gamma \models X, g_t^{e_1}, ..., g_t^{e_N} \vdash_{\mathcal{E}-\hat{e}} g_t^h$.

We obtain a problem where we only have elements in the target group, we can therefore reduce the general problem to the single group case.

C.3.2 Reduction to Polynomials

Lemma C.4. *For any sets X and Y, polynomials $w_1, ...w_N, h \in \mathbb{K}[X, Y]$ we have $\Gamma \models X, g_t^{w_1}, ..., g_t^{w_N} \vdash_{\mathcal{E}} g_t^h$ if and only if:*

$$\exists (e_i, g_i) \in \mathbb{K}[X], (\forall i, \Gamma \models g_i \neq 0) \wedge \sum_i e_i \times \frac{f_i}{g_i} = h$$

Proof. If $\Gamma = \emptyset$, the adversary can construct elements of the form $(g_t^{w_i})^{e_i}$, where $e_i \in \mathbb{K}[X]$, i.e. e_i is a polynomial constructed over variables fully known by the adversary, and then multiply this kind of term, yielding a sum in the exponent. If $\Gamma \neq \emptyset$, they may also divide by some $g_t^{g_i}$, with $g_i \in \mathbb{K}[X]$. We capture here the three capabilities of the adversary, which when looking in the exponent immediately translate into the formula on the right side. ■

To handle this new problem, we notice that we can actually compute the set $\{g | \Gamma \models g \neq 0\}$. Indeed, for each axiom $f \neq 0$, we can extract a finite set of non zero irreducible polynomials by factorizing them (for example using Lenstra algorithm [Len85]). Any non annulling polynomial will be a product of all these irreducible polynomials. We can then obtain a finite set $Gs = (g_i)$ such that $G = \{g | \Gamma \models g \neq 0\} = \{\prod_{g \in Gs} g^{k_g} | \forall g, k_g \in \mathbb{N}\}$. With these notations, we can simplify proposition 1, because we know the form of the g_i. Moreover, as we do not want to deal with fractions, we multiply by the common denominator of all the $\frac{w_i}{g_i}$.

Lemma C.5. *For any sets X and Y, polynomials $w_1, ...w_N, h \in \mathbb{K}[X,Y]$ we have $\Gamma \models X, g_t^{w_1}, ..., g_t^{w_N} \vdash_{\mathcal{E}} g_t^h$ if and only if:*

$$\exists (e_i) \in \mathbb{K}[X], (k_g) \in \mathbb{N}, \sum_i e_i \times w_i = h \prod_{g \in Gs} g^{k_g}$$

We do not prove this lemma, we will rather reformulate it using more refined mathematical structures and then prove it. Let us call $M = \{\sum_i e_i \times w_i | e_i \in \mathbb{K}[X]\}$ the free $\mathbb{K}[X]$-module generated by the (w_i). We recall that a S-module is a set stable by multiplication by S and addition, and that $\langle (w_i) \rangle_S$ is the S-module generated by (w_i). We also recall the definition of the saturation :

Definition C.1. Given a S-module T, $f \in S$ and $S \subset S'$, the saturation of T by f in S' is $T :_{S'} (f)^\infty = \{g \in S' | \exists n \in \mathbb{N}, f^n g \in T\}$.

The previous lemma can be reformulated using saturation; if M is the module generated by $w_1, ..., w_N$:

Lemma C.6. $\Gamma \models X, g_t^{w_1}, ..., g_t^{w_N} \vdash_{\mathcal{E}} g_t^h \Leftrightarrow h \in M :_{\mathbb{K}[X,Y]} (g_1 ... g_n)^\infty$

Proof. We recall that:

$$M :_{\mathbb{K}[X,Y]} (g_1 ... g_n)^\infty = \{x \in \mathbb{K}[X,Y] | \exists k \in \mathbb{N}, (g_1 ... g_n)^k \times x \in M\}$$

$\Rightarrow$ We have $\sum_i e_i \times w_i = h \prod_{g \in Gs} g^{k_g}$. With $K = max(k_g)$, we multiply both sides by $\prod_g g^{K - k_g}$ to get $h \prod_{g \in Gs} g^K = \sum_i \prod_g g^{K - k_g} e_i \times w_i \in M$. Which proves that $h \in M :_{\mathbb{K}[X,Y]} (g_1 ... g_n)^\infty$.
$\Leftarrow$ If $h \in M :_{\mathbb{K}[X,Y]} (g_1 ... g_n)^\infty$, we instantly have $(e_i) \in \mathbb{K}[X], k \in \mathbb{N}$ such that $h \prod_{g \in Gs} g^{k_g} = \sum_i e_i f_i$. ■

We then simplify the saturation, by transforming it into the membership of the intersection of modules.

Lemma C.7. *For any sets X and Y, $f_1, ... f_n, h \in \mathbb{K}[X,Y]$, $g \in \mathbb{K}[X]$, let $M = \{\sum_i e_i \times f_i | e_i \in \mathbb{K}[X]\}$. Then, with t a fresh variable $M :_{\mathbb{K}[X,Y]} g^\infty = \langle (f_i) \cup ((gt-1)Y^{\vec{j}})_{\vec{j} \in \{deg_Y(f_i)\}} \rangle_{\mathbb{K}[X,t]} \cap \mathbb{K}[X,Y]$.*

Proof. $\subset$. Let there be $v \in M :_{\mathbb{K}[X,Y]} g^\infty$. Then, we have k such that $g^k \times v \in M$. The following equalities shows that v is in the right side set $v = g^k t^k v - (1 + gt + ... + g^{k-1} t^{k-1})(gt - 1)v$. Indeed, $g^k t^k v \in M\mathbb{K}[X,t]$, so we have $(e_i) \in \mathbb{K}[X,t]$ such that $g^k t^k v = \sum_i e_i f_i$. Moreover, $g^k \times v \in M$ and $g \in \mathbb{K}[X]$ implies that $deg_Y(v) \subset \{deg_Y(f_i)\}$. So we do have $(e'_i) \in \mathbb{K}[X,t]$ and $(\vec{j_i}) \subset \{deg_Y(f_i)\}$ such that

$$(1 + gt + ... + g^{k-1} t^{k-1})(gt - 1)v = \sum e'_i (gt - 1) Y^{\vec{j_i}}$$

Finally, $v \in \langle (f_i) \cup ((gt-1)Y^{\vec{j}})_{\vec{j} \in \{deg_Y(f_i)\}} \rangle_{\mathbb{K}[X,t]} \cap \mathbb{K}[X,Y]$.

$\supset$. Let there be $v \in \langle (f_i) \cup ((gt-1)Y^{\vec{j}})_{\vec{j} \in \{deg_Y(f_i)\}} \rangle_{\mathbb{K}[X,t]} \cap \mathbb{K}[X,Y]$. Then we have $(e_i), (e'_i) \in K[X,t]$ and $(\vec{j_i}) \subset \{deg_Y(f_i)\}$ such that :

$$v = \sum_i e_i f_i + \sum_i e'_i (gt - 1) Y^{\vec{j_i}}$$

We have that $v \in \mathbb{K}[X,Y]$, so v is invariant by t. So, if we substitute t with $\frac{1}{g}$, we have that $v = \sum_i e_i(X, \frac{1}{g}) f_i$. Let us consider g^k the common denominator of all those fractions and call $e_i'' = g^k e_i \in \mathbb{K}[X]$. We finally have $g^k \times v = \sum_i e_i'' f_i \in M$, which means that $v \in M :_{\mathbb{K}[X,Y]} g^\infty$. ■

The Buchberger algorithm allows us to compute a Gröbner basis of any free $\mathbb{K}[X]$-module [Eis13] and then decide the membership problem for a module. We thus solve our membership problem using this method.

Theorem C.1. *For any sets X and Y, polynomials $f_1, ...f_n, h \in \mathbb{K}[X,Y]$, group elements $g_{i_1}, ..., g_{i_n}$ and a set of axioms Γ we can decide if $\Gamma \models X, g_{i_1}^{f_1}, ..., g_{i_n}^{f_n} \vdash_\mathcal{E} g_t^h$*

Proof. To decide if h is deducible, we first reduce to a membership problem with Lemma C.6 that can be solved using Lemma C.7 by computing the Gröbner basis of $\langle (f_i) \cup ((gt - 1)Y^{\vec{j}})_{\vec{j} \in \{deg_Y(f_i)\}} \rangle_{\mathbb{K}[X,t]}$, keeping only the elements of the base that are independent of t and then checking if the reduced form of h is 0. ■

As a side note, being able to decide the deducibility in this setting allows us to decide another classical formal method problem, the static equivalence. Indeed the computation of the Gröbner basis allows us to find generators of the corresponding syzygies (Theorem 15.10 of [Eis13]), which actually captures all the possible distinguishers of a frame.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Preuves de protocoles cryptographiques : méthodes symboliques et attaquants puissants

Mots clés: Protocols cryptographiques, méthodes formelles, modèle calculatoire, modèle symbolique, logique BC

Résumé: L'utilisation des protocoles de communication est omniprésente dans notre société, mais leur utilisation comporte des risques de sécurité ou d'atteinte à la vie privée. Pour réduire ces risques, il faut exiger de solides garanties, i.e. des preuves formelles, approfondies, modulaires et vérifiées par ordinateur. Toutefois, de telles preuves sont très difficiles à obtenir. Nous essayons dans cette thèse de faciliter ce processus dans le cas des protocoles cryptographiques et d'attaquants puissants. Nos contributions principales sont

1. une méthodologie d'analyse approfondies dans le cas de l'authentification multi-facteurs;
2. des résultats de composition permettant des preuves modulaires de protocoles complexes dans le modèle calculatoire;
3. l'automatisation d'étapes élémentaires de preuves calculatoires via des méthodes symboliques appliquées à des programmes probabilistes;
4. un prototype d'assistant de preuve dans le modèle de l'attaquant symbolique calculatoirement complet.

Title: Proofs of Security Protocols: Symbolic Methods and Powerful Attackers

Keywords: Security Protocols, Formal Methods, Computational Model, Symbolic Model, BC logic

Abstract: The use of communication protocols has become pervasive at all levels of our society. Yet, their uses come with risks, either about the security of the system or the privacy of the user. To mitigate those risks, we must provide the protocols with strong security guarantees: we need formal, extensive, modular and machine-checked proofs. However, such proofs are very difficult to obtain in practice. In this Thesis, we strive to ease this process in the case of cryptographic protocols and powerful attackers. The four main contributions of this Thesis, all based on symbolic methods, are

1. a methodology for extensive analyses via a case study of multi-factor authentication;
2. composition results to allow modular proofs of complex protocols in the computational model;
3. symbolic methods for deciding basic proof steps in computational proofs, formulated as problems on probabilistic programs;
4. a prototype of a mechanized prover in the Computationally Complete Symbolic Attacker model.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France