A Machine-Checked Formalization of Sigma-Protocols

Gilles Barthe, Daniel Hedin, Santiago Zanella-Béguelin, Benjamin Grégoire, Sylvain Heraud

A Machine-Checked Formalization of Sigma-Protocols

Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin
IMDEA Software, Madrid, Spain
{*Gilles.Barthe, Daniel.Hedin, Santiago.Zanella*}*@imdea.org*

Benjamin Grégoire, Sylvain Heraud
INRIA Sophia Antipolis-Méditerranée, France
{*Benjamin.Gregoire, Sylvain.Heraud*}*@inria.fr*

***Abstract*—Zero-knowledge proofs have a vast applicability in the domain of cryptography, stemming from the fact that they can be used to force potentially malicious parties to abide by the rules of a protocol, without forcing them to reveal their secrets. Σ-protocols are a class of zero-knowledge proofs that can be implemented efficiently and that suffice for a great variety of practical applications. This paper presents a first machine-checked formalization of a comprehensive theory of Σ-protocols. The development includes basic definitions, relations between different security properties that appear in the literature, and general composability theorems. We show its usefulness by formalizing—and proving the security—of concrete instances of several well-known protocols. The formalization builds on CertiCrypt, a framework that provides support to reason about cryptographic systems in the Coq proof assistant, and that has been previously used to formalize security proofs of encryption and signature schemes.**

I. Introduction

Proofs of knowledge [1], [2] are two-party interactive protocols where one party, called the *prover*, convinces the other one, called the *verifier*, that she knows *something*. Typically, both parties share a common input x and *something* refers to a witness w of membership of the input x to an $\mathcal{NP}$ language. Proofs of knowledge are useful to enforce honest behavior of potentially malicious parties [3]: the knowledge witness acts as an authentication token used to establish that the prover is a legitimate user of a service provided by the verifier, or as evidence that a message sent by the prover has been generated in accordance to the rules of a protocol. Proofs of knowledge must be complete, so that a prover that has indeed knowledge of a witness can convince a honest verifier, and sound, so that a dishonest prover has little chance of being convincing. In addition, practical applications often require to preserve secrecy or anonymity; in such cases the proof should not leak anything about the witness. Zero-knowledge proofs are computationally convincing proofs of knowledge that achieve this goal, i.e. they are convincing and yet the verifier does not learn anything from interacting with the prover beyond the fact that the prover knows a witness for their common input. This property has an elegant formulation: a protocol is said to be zero-knowledge when transcripts of protocol runs between a prover P and a (possibly dishonest) verifier V can be efficiently simulated without ever interacting with the prover—but with access to the strategy of V. In particular, this implies that proofs are not transferable; a conversation is convincing only for the verifier interacting with the prover and cannot be replayed to convince a third party.

In his PhD dissertation [4], Cramer introduced the concept of Σ-protocols, a class of three-move interactive protocols that are suitable as a basis for the design of many efficient and secure cryptographic services. Cramer described Σ-protocols as abstract modules and showed that they are realizable when instantiated for most computational assumptions commonly considered in cryptography, including the difficulty of computing discrete logarithms or factoring integers, or the existence of some abstract function families (e.g. one-way group homomorphisms). In addition, he gave an effective method to combine Σ-protocols to obtain zero-knowledge proofs of any Boolean formula constructed using the AND and OR operators from formulæ for which Σ-protocols exist. This means that Σ-protocols can be used in a practical setting as building blocks to achieve various cryptographic goals. Applications of Σ-protocols notably include secure multi-party computation, identification schemes, secret ballot electronic voting, and anonymous attestation credentials.

This paper reports on a fully machine-checked formalization of a comprehensive theory of Σ-protocols using CertiCrypt [5]–[7], a general framework for reasoning about cryptographic schemes built on top of the Coq [8] proof assistant. Our formalization consists of more than 10,000 lines of Coq code, and covers the basics of Σ-protocols: definitions, relations between different notions of security, general constructions and composability theorems. We show its applicability by formalizing several well-known protocols, including the Schnorr, Guillou-Quisquater, Okamoto, and Feige-Fiat-Shamir protocols. The highlight of the formalization is a generic account of Σ^ϕ-protocols, that prove knowledge of a preimage under a group homomorphism ϕ. We use the module system of Coq to define and relate the classes of Σ^ϕ- and Σ-protocols. Our formalization of Σ^ϕ-protocols provides sufficient conditions (the so-called specialness conditions) on the group homomorphism ϕ so that every Σ^ϕ-protocol can be construed as a Σ-protocol. Moreover, we show that special homomorphisms are closed under direct product, which yields a cheap way of AND-combining Σ^ϕ proofs. We exploit the generality of our result to achieve short proofs of completeness, special soundness, and (honest verifier) zero-knowledge for many protocols in the literature.

Related work

Our work participates to an upsurge of interest in Σ-protocols, and shares some motivations and commonalities with recently published papers. Specifically, our account of Σ^{ϕ}-protocols coincides with Maurer's unifying treatment of proofs of knowledge for preimages of group homomorphisms [9]. Concretely, Maurer exhibits a main protocol that uses a group homomorphism—which in our setting corresponds to the definition of the module of Σ^{ϕ}-protocols in Section IV—and shows (in his Theorem 3) that under suitable hypotheses the main protocol is a Σ-protocol. He gives several instances of the main protocol by picking suitable group homomorphisms and showing that they satisfy these hypotheses.

Our work is also closely connected to a recent effort by Bangerter and co-workers [10], [11] to design and implement efficient zero-knowledge proofs of knowledge. They provide both a set of sufficient conditions on a homomorphism ϕ under which the corresponding Σ^{ϕ}-protocol can be viewed as a Σ-protocol [10, Theorem 1], and a generalization that allows to consider sets of linear relations among preimages of group homomorphisms [10, Theorem 2]. The latter result is used to justify the soundness of a compiler that generates efficient code from high-level descriptions of protocols. As future work, the authors of [10] mention that they plan to make their compiler certifying, so that it would generate proofs accompanying the code. Doing so from scratch remains a daunting task. By building on CertiCrypt, our formalization could be readily used as a stepping stone for a modular certifying compiler, in which (high-level, unoptimized) code is certified, and then compiled to efficient code using a certified or certifying compiler; see e.g. [12], [13] for an instance of applying ideas from certified/certifying compilation to cryptography.

Cryptographic primitives need not only be secure; they must also be used correctly. In a series of papers, Backes and co-authors [14], [15] develop sound analysis methods for protocols that use zero-knowledge proofs, and apply their analyses to verify the authentication and secrecy properties of the Direct Anonymous Attestation Protocol. One extremely ambitious objective would be to use their results, which complement ours, to fully certify the security of the protocol in the computational model. Intermediate results would involve formalizing computational soundness results [16], [17], which represents a substantial amount of work on its own.

II. A CERTICRYPT PRIMER

The goal of this section is to provide a brief overview of the CertiCrypt framework. We first present the syntax and semantics of the language used to describe protocols, and then some of the reasoning tools that we use. It should be noted, however, that the formalization of Σ-protocols only uses a limited number of features of the CertiCrypt framework; we refer the reader to [5] for an account of features that are not required for this paper.

A. Syntax and semantics of games

We will use games to describe the interaction between the entities participating in a protocol and procedures to represent the entities themselves. In CertiCrypt, a game is simply a program in a probabilistic programming language together with an environment mapping procedures to their implementation.

Given sets $\mathcal{V}$ and $\mathcal{P}$ of variable and procedure identifiers, respectively, the language is inductively defined as follows,

$$\begin{array}{lcll}
\mathcal{I} & ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{deterministic assignment} \\
& | & \mathcal{V} \overset{\$}{\leftarrow} \mathcal{DE} & \text{random assignment} \\
& | & \textsf{if } \mathcal{E} \textsf{ then } \mathcal{C} \textsf{ else } \mathcal{C} & \text{conditional} \\
& | & \textsf{while } \mathcal{E} \textsf{ do } \mathcal{C} & \text{while loop} \\
& | & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call} \\
\mathcal{C} & ::= & \textsf{skip} & \text{nop} \\
& | & \mathcal{I};\ \mathcal{C} & \text{sequence}
\end{array}$$

where $\mathcal{E}$ is a set of deterministic expressions and $\mathcal{DE}$ is a set of expressions that evaluate to distributions and from which values can be sampled in random assignments. CertiCrypt allows the core language to be extended with user-defined types and operators. Our development puts this feature to good use by defining various extensions including types for groups, and operators for homomorphisms and permutations on those groups.

The semantics of programs is defined in terms of the measure monad $\mathcal{D}(X)$ of Audebaud and Paulin [18], whose type constructor is defined as

$$\mathcal{D}(X) \stackrel{\text{def}}{=} (X \to [0,1]) \to [0,1]$$

The semantics of a command $c \in \mathcal{C}$ is given by a function $[\![c]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$, that relates an initial memory $\mu \in \mathcal{M}$ to the expectation operator of the (sub) probability distribution of final memories resulting from its execution. This allows to define the probability of an event A in a game G and an initial memory μ in terms of its characteristic function $\mathbb{1}_A$, as follows

$$\Pr\left[\mu : G\right] E \stackrel{\text{def}}{=} [\![G]\!]\ \mu\ \mathbb{1}_A$$

B. Complexity

In the context of zero-knowledge proofs one often needs to prove that some constructions are probabilistic polynomial-time (PPT) programs (e.g. the simulator that justifies the zero-knowledge property of a protocol). CertiCrypt provides an instrumented semantics that accounts for the execution cost of programs, and that is used to characterize the class of PPT programs, i.e. of programs that execute in polynomial time and polynomial memory. The characterization relies on an axiomatization of the execution time and the memory usage of expressions:

- we postulate the execution time of each operator, in the form of a function that depends on the inputs of the operator—which corresponds to the so-called functional time model;
- we postulate for each datatype a size measure, in the form of a function that assigns to each value its memory footprint.

We stress that making complexity assumptions on operators is perfectly legitimate. It is a well-known feature of dependent type theories (as is the case of the calculus of Coq) that they cannot express the cost of the computations they purport without using computational reflection, i.e. formalizing an execution model (e.g. probabilistic Turing machines) within the theory itself and proving that functions in type theory denote machines that execute in polynomial time. In our opinion, such a step is overkill. A simpler solution to the problem is to restrict in as much as possible the set of primitive operators, so as to minimize the set of assumptions upon which the complexity proofs rely. For instance, one could, instead of defining Euclidean division as a type-theoretical function, define a procedure that performs the computation, and show that the computation is PPT provided addition is. Again, we claim that such a step would be an overkill.

C. *Reasoning in CertiCrypt*

Most of the security properties of interest in this paper can be stated either in terms of an equivalence between two games or in terms of the probability of an event occurring in a game. Sometimes directly computing the probability of an event E_1 in a game G_1 may be difficult. The essence of game-based proofs is to introduce a sequence of intermediate games and events $(G_1, E_1), \dots, (G_n, E_n)$ in such a way that the probability of the event E_1 in the original game can be obtained from the probability of the event E_n in the last game and the relation between the probability of events in consecutive games. Typically, relating the probability of events in consecutive games boils down to proving that both games satisfy some form of program equivalence.

In CertiCrypt, program equivalence is formalized using a probabilistic relational Hoare logic. In its more general form, this logic deals with judgments of the form $\vdash G_1 \sim G_2 : \Phi \Rightarrow \Psi$, where Φ and Ψ are relations over program states (memories). A simplified form of judgment, more amenable to mechanization via syntactic manipulation, is obtained when the relations Φ, Ψ are the equality relation on subsets of program variables. We define this formally next.

Definition 1 (Observational Equivalence)**.** Observational equivalence is defined relative to a set of input variables I and a set of output variables O. Two programs G_1 and G_2 are observationally equivalent with respect to I and O, written $\vdash G_1 \simeq_O^I G_2$, if and only if for any memories μ_1, μ_2 that coincide on I and functions $f, g : \mathcal{M} \to [0,1]$ depending only on the value of variables in O, we have

$$[\![G_1]\!]\ \mu_1\ f = [\![G_2]\!]\ \mu_2\ g$$

Note that observational equivalence is only a partial equivalence relation; reflexivity only holds if the distribution over O induced by the program is completely determined by the initial values of variables in I. We will sometimes use a slightly more general form of equivalence $G_1 \simeq_O^{I \wedge \Phi} G_2$, denoting observational equivalence with respect to I, and O with the additional demand that the relation Φ holds as a precondition.

Preservation of Probability. The following proof rule relates observational equivalence and probability. Assume that an expression E denoting an event depends only on a set of variables O (a sufficient, syntactic condition that ensures this is that the set of free variables of E be a subset of O). Then, to show that the probability of E is identical in two games G_1, G_2 executed in initial memories μ_1, μ_2 respectively, it is sufficient to exhibit a set of variables I such that μ_1, μ_2 coincide on I, and $\vdash G_1 \simeq_O^I G_2$:

$$\frac{\mathsf{fv}(E) \subseteq O \quad \vdash G_1 \simeq_O^I G_2 \quad \mu_1 =_I \mu_2}{\Pr\left[\mu_1 : G_1\right] E = \Pr\left[\mu_2 : G_2\right] E}\ [\text{PrEq}]$$

Mechanized Reasoning. In addition to the standard tactics provided by Coq, CertiCrypt provides tactics that help mechanize reasoning about an observational equivalence goal of the form $\vdash c_1 \simeq_O^I c_2$. In order to deal with procedure calls the user needs to provide the tactics with specifications of the procedures appearing in programs. These specifications include information about the variables a procedure may modify and an observational equivalence statement saying under which conditions one can guarantee that two calls to a procedure are equivalent. In most cases, a simple static analysis of the code of a procedure is sufficient to compute a meaningful specification; CertiCrypt provides means of computing specifications in this way that lift the burden from the user of having to craft them. We briefly describe next the most relevant tactics used to mechanize the proofs in this paper:

- `eqobs_in`: solves goals of the form $\vdash c \simeq_O^I c$ by performing a dependency analysis to determine a set I' such that $\vdash c \simeq_O^{I'} c$ and checking whether $I' \subseteq I$. It handles loops by trying to compute a fixpoint in a bounded number of iterations and relies on specifications to handle procedure calls. Variants of `eqobs_in` include `eqobs_tl`, that does not require that the two commands coincide, but instead acts only on the longest common suffix of them. In addition, CertiCrypt provides the converse tactic `eqobs_hd` that strips off the longest common prefix of two programs and a tactic `eqobs_ctxt` that combines both.

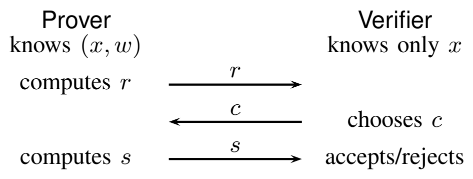


Figure 1. Characteristic 3-step interaction in a run of a Σ-protocol

- `deadcode`: performs dead code elimination; it removes from programs the instructions that have no influence on the set of output variables O.
- `ep`: implements expression propagation; it relies on a generic function for dataflow analysis on programs, transforming programs by performing partial evaluation using the result of this analysis. The program analyzer may be extended with simplification rules for expressions; we often use this to normalize expressions based on algebraic properties of operators (e.g. expressions involving homomorphisms). A variant `ep_eq` e_1 e_2 can be used to jump start the analysis with the directed equality $e_1 = e_2$ which is then left as an obligation.
- `swap`: reorders instructions in programs to generate a longest common suffix while preserving observational equivalence.
- `alloc`: takes as arguments a variable x and a fresh variable y, and introduces a copy of variable x in variable y, consistently replacing all uses of x by y.
- `clean_nm`: simplifies the goal by removing from the set O variables in I that are not modified by the programs.
- `inline`: unfolds procedure calls; the tactic `sinline` combines `inline`, `alloc`, `ep`, and `deadcode` to simplify the goal after inlining a procedure call.

CertiCrypt provides certified implementations of tactics that, in addition to the *two-sided* versions described above, yield one-sided versions that apply the transformation only to the program on the left (suffix `l`) or on the right hand side (suffix `r`) of the goal.

III. Sigma-protocols

A Σ-protocol is a 3-step interactive protocol where a prover P interacts with a verifier V. Both parties have access to a common input x, and the goal of the prover is to convince the verifier that she knows some value w suitably related to x, without revealing anything beyond this assertion. The protocol begins with the prover sending a commitment r to the verifier, who responds by sending back a random challenge c chosen uniformly from a set C; the prover then computes a response s to the challenge and sends it to the verifier, who either accepts or rejects the conversation. Figure 1 shows a diagram of a run of a Σ-protocol.

Formally, a Σ-protocol is defined with respect to a knowledge relation R. This terminology comes from interpreting the proof system as proving membership of the common input to an $\mathcal{NP}$ language L. Each $\mathcal{NP}$ language admits an efficient membership verification procedure and a polynomial-time recognizable relation R_L such that

$$L = \{x \mid \exists w, (x, w) \in R_L\}$$

Proving that x belongs to the language amounts to proving knowledge of a witness w related to x via R_L. In CertiCrypt, the class of Σ-protocols is formalized as a module type parametrized over the knowledge relation R, and a number of procedures specifying the different phases of the prover and the verifier; the module type specifies as well the properties that any given protocol instance must satisfy. In the remainder of this section we describe in detail our formal definition of Σ-protocols and comment on an alternative—but in some sense equivalent—specification of the zero-knowledge property.

Definition 2 (Σ-protocol)**.** A Σ-protocol for a knowledge relation R is a 3-step protocol between a prover P and a verifier V, whose interaction is described by the following parametrized program:

```
Protocol$(x, w)$ :
  $(r, state) \leftarrow \mathsf{P}_1(x, w)$;
  $c \leftarrow \mathsf{V}_1(x, r)$;
  $s \leftarrow \mathsf{P}_2(x, w, state, c)$;
  $b \leftarrow \mathsf{V}_2(x, r, c, s)$
```

In the above program, the two phases of the prover are described by the procedures P_1 and P_2, while the phases of the verifier are described by V_1 and V_2. Note that the protocol explicitly passes state between the phases of the participants; we could have used instead global variables shared between P_1 and P_2 on one hand, and V_1 and V_2 on the other, but that would unnecessarily complicate the proofs because we would need to specify that the procedures representing one party do not have access to the shared state of the other party. All the protocols that we consider in the following are *public-coin*, meaning that a honest verifier chooses the challenge uniformly from some predefined set.

A Σ-protocol must satisfy the following three properties,

1) Completeness: Given a public input x and a witness w such that $(x, w) \in R$, the prover is always able to convince the verifier, i.e., when the protocol is run in a memory μ where $R(\mu(x), \mu(w))$, the final value of b is always true:

$$\forall \mu, R(\mu(x), \mu(w)) \Longrightarrow \Pr[\mathsf{Protocol}, \mu : b = \mathsf{true}] = 1$$

2) Special Honest Verifier Zero-Knowledge (sHVZK): There exists a probabilistic polynomial-time simulator S that given $x \in \mathsf{dom}(R)$ and a challenge c, computes triples (r, c, s) with the same distribution as a valid conversation. The property is formalized in terms of a

version of the protocol where the challenge c is fixed,

$$\boxed{\begin{array}{l}\textbf{Protocol}(x,w,c):\\ (r,\mathit{state}) \leftarrow \mathsf{P}_1(x,w);\\ s \leftarrow \mathsf{P}_2(x,w,\mathit{state},c);\\ b \leftarrow \mathsf{V}_2(x,r,c,s)\end{array}} \simeq^{\{x,c\}\wedge R(x,w)}_{\{r,c,s\}} (r,s) \leftarrow \mathsf{S}(x,c)$$

3) Special Soundness: Given two accepting conversations (r,c_1,s_1), (r,c_2,s_2) for an input x, with distinct challenges but with the same commitment r, there exists a PPT knowledge extractor procedure KE that computes a witness w such that $(x,w) \in R$. Formally, for any memory μ,

$$\left.\begin{array}{l}\mu(c_1) \neq \mu(c_2)\\ \Pr[b \leftarrow \mathsf{V}_2(x,r,c_1,s_1),\mu : b = \mathsf{true}] = 1\\ \Pr[b \leftarrow \mathsf{V}_2(x,r,c_2,s_2),\mu : b = \mathsf{true}] = 1\end{array}\right\} \Longrightarrow$$
$$\Pr[w \leftarrow \mathsf{KE}(x,r,c_1,c_2,s_1,s_2),\mu : (x,w) \in R] = 1$$

Special soundness might seem a relatively weak property at first sight. It can be shown using a rewinding argument (although we did not formalize this result in Coq) that thanks to special soundness, any Σ-protocol with challenge set C can be seen as a *proof of knowledge* with *soundness error* $|C|^{-1}$ [19]. Informally, this means that any efficient prover (possibly dishonest) that manages to convince a honest verifier for a public input x with a probability greater than $|C|^{-1}$ can be turned into an efficient procedure that computes a witness for x.

A. Relation between sHVZK and HVZK

Some authors require that Σ-protocols satisfy a somewhat weaker property known as honest verifier zero-knowledge rather than the *special* version of this property mentioned above. The difference is that in the former the simulator is allowed to choose the challenge while in the latter the challenge is fixed. In other words, plain HVZK requires that there exists a PPT simulator S that given just $x \in \mathsf{dom}(R)$ computes a triple (r,c,s) with the same distribution as the verifier's view of a conversation. The relation between the two notions has been studied by Cramer [4]. As an illustration of the use of CertiCrypt and the Σ-protocol framework, the formalization of this relation is discussed below.

Theorem 3 (sHVZK implies HVZK)**.** *A Σ-protocol satisfying sHVZK also satisfies HVZK.*

Proof: A HVZK simulator S' can be built from the sHVZK simulator S:

$$\boxed{\begin{array}{l}\textbf{Simulator } \mathsf{S}'(x):\\ \quad c \xleftarrow{\$} \{0,1\}^k;\\ \quad (r,s) \leftarrow \mathsf{S}(x,c);\\ \quad \mathsf{return}\ (r,c,s)\end{array}}$$

The fact that S' perfectly simulates conversations of the protocol can be proved by means of the following sequence of games:

$$\begin{array}{rl}\mathsf{Protocol}(x,w) & \simeq^{\{x\}\wedge R(x,w)}_{\{r,c,s\}} c \xleftarrow{\$} \{0,1\}^k; \mathsf{Protocol}(x,w,c)\\ & \simeq^{\{x\}\wedge R(x,w)}_{\{r,c,s\}} c \xleftarrow{\$} \{0,1\}^k; (r,s) \leftarrow \mathsf{S}(x,c)\\ & \simeq^{\{x\}\wedge R(x,w)}_{\{r,c,s\}} (r,c,s) \leftarrow \mathsf{S}'(x)\end{array}$$

The first and last equivalences are easily proved by unfolding procedure calls using the tactic `inline`, and reordering instructions in the resulting programs using `swap`. To prove the second equivalence, the tactic `eqobs_hd` is used to get rid of the instruction $c \xleftarrow{\$} \{0,1\}^k$ that is common to both games; the resulting goal matches exactly the definition of sHVZK for S. ■

In a sense, sHVZK is a stronger property than HVZK, because a protocol satisfying sHVZK can be shown to satisfy HVZK, while the converse is not generally true. However, from every protocol (P,V) that satisfies HVZK it is possible to construct a protocol $(\mathsf{P}',\mathsf{V}')$ that satisfies sHVZK and is nearly as efficient as the original protocol:

$$\begin{array}{lcl}\mathsf{P}'_1(x,w) & \overset{\text{def}}{=} & (r,\mathit{state}) \leftarrow \mathsf{P}_1(x,w); c' \xleftarrow{\$} \{0,1\}^k;\\ & & \mathsf{return}\ ((r,c'),(\mathit{state},c'))\\ \mathsf{P}'_2(x,w,(\mathit{state},c'),c) & \overset{\text{def}}{=} & s \leftarrow \mathsf{P}_2(x,w,\mathit{state},c \oplus c'); \mathsf{return}\ s\\ \mathsf{V}'_1(x,(r,c')) & \overset{\text{def}}{=} & c \leftarrow \mathsf{V}_1(x,r);\ \mathsf{return}\ (c \oplus c')\\ \mathsf{V}'_2(x,(r,c'),c,s) & \overset{\text{def}}{=} & b \leftarrow \mathsf{V}_2(x,r,c \oplus c',s);\ \mathsf{return}\ b\end{array}$$

Essentially, the construction creates a new protocol for which HVZK and sHVZK coincide. The difference is that in the new protocol the challenge that the verifier chooses is xor-ed with a random bitstring sampled by the prover at the beginning of the protocol.

Theorem 4 (sHVZK from HVZK)**.** *If a protocol (P,V) is a Σ-protocol as in Definition 2 but satisfying HVZK instead of sHVZK, then the protocol $(\mathsf{P}',\mathsf{V}')$ defined above is a Σ-protocol.*

Proof:

Completeness: Follows easily from the completeness of protocol (P,V) and the following algebraic property of the exclusive or operator:

$$(c \oplus c') \oplus c' = c$$

Special Honest Verifier Zero-Knowledge: The following is a sHVZK simulator for the protocol

$$\mathsf{S}'(x,c) \overset{\text{def}}{=} (\hat{r},\hat{c},\hat{s}) \leftarrow \mathsf{S}(x);\ \mathsf{return}\ ((\hat{r},c \oplus \hat{c}),\hat{s})$$

(The variables of the original protocol are decorated with a hat.) We prove this by means of a sequence of program equivalences,

$$\begin{array}{rl}\mathsf{Protocol}'(x,w,c) & \simeq^{\{x,c\}\wedge R(x,w)}_{\{r,c,s\}} \begin{array}{l}\mathsf{Protocol}(x,w);\\ r \leftarrow (\hat{r},c \oplus \hat{c});\\ s \leftarrow \hat{s}\end{array}\\ & \simeq^{\{x,c\}\wedge R(x,w)}_{\{r,c,s\}} \begin{array}{l}(\hat{r},\hat{c},s) \leftarrow \mathsf{S}(x);\\ r \leftarrow (\hat{r},c \oplus \hat{c})\end{array}\\ & \simeq^{\{x,c\}\wedge R(x,w)}_{\{r,c,s\}} (r,s) \leftarrow \mathsf{S}'(x,c)\end{array}$$

The first and last equivalences are proved without much difficulty using the program transformation tactics described in Section II, while the second can be reduced to the HVZK of S using the `alloc` and `eqobs_tl` tactics to simplify the goal.

Soundness: From a conversation $((r, c'), (c \oplus c'), s)$ of $(\mathsf{P}', \mathsf{V}')$ a conversation (r, c, s) of the original protocol can be trivially recovered. Thus, the following knowledge extractor proves special soundness of $(\mathsf{P}', \mathsf{V}')$:

```
$\mathsf{KE}'(x, (r, c'), c_1, c_2, s_1, s_2)$ :
  $w \leftarrow \mathsf{KE}(x, r, c' \oplus c_1, c' \oplus c_2, s_1, s_2)$; return $w$
```

■

IV. Sigma protocols based on special homomorphisms

An important class of Σ-protocols are the so-called Σ^ϕ-protocols, that prove knowledge of a preimage under a homomorphism. The Schnorr protocol [20], one of the most archetypal zero-knowledge proofs, is an instance of a Σ^ϕ-protocol that proves knowledge of a discrete logarithm in a cyclic group, i.e. the homomorphism is in this case exponentiation, $\phi(x) = g^x$, where g is a generator of the group.

Our formalization of Σ^ϕ-protocols is constructive. We provide a functor that, given a homomorphism ϕ together with proofs that it satisfies certain properties, builds a concrete Σ-protocol for proving knowledge of a preimage under ϕ. This protocol comes with proofs of completeness, soundness, and sHVZK. Thus, all that it takes to build an instance of a Σ^ϕ-protocol is to specify a homomorphism and prove that it has the necessary properties. In this way, we give several examples of Σ^ϕ-protocols, including the Schnorr, Guillou-Quisquater and Feige-Fiat-Shamir protocols. Although using the Σ^ϕ construction spares us the hassle of proving each time the properties in Definition 2, these instantiations remain non-trivial because one needs to formalize the homomorphisms themselves, which in turn requires to give representations of the groups over which they are defined.

In the remaining of this subsection we let $(\mathcal{G}, \oplus)$ be a finite additive group and $(\mathcal{H}, \otimes)$ a multiplicative group.

Definition 5 (Σ^ϕ-protocol)**.** Let $\phi : \mathcal{G} \to \mathcal{H}$ be a homomorphism, and define $R \overset{\text{def}}{=} \{(x, w) \mid x = \phi(w)\}$. The Σ^ϕ-protocol for relation R with challenge set C is the Σ-protocol (P, V) defined as follows:

$$\begin{array}{lll}
\mathsf{P}_1(x, w) & \overset{\text{def}}{=} & y \overset{\$}{\leftarrow} \mathcal{G};\ \text{return } (\phi(y), y) \\
\mathsf{P}_2(x, w, y, c) & \overset{\text{def}}{=} & \text{return } (y \oplus cw) \\
\mathsf{V}_1(x, r) & \overset{\text{def}}{=} & c \overset{\$}{\leftarrow} C;\ \text{return } c \\
\mathsf{V}_2(x, r, c, s) & \overset{\text{def}}{=} & \text{return } (\phi(s) = r \otimes x^c)
\end{array}$$

It can be shown that the above protocol satisfies the properties of a Σ-protocol when $C = \{0, 1\}$. However, a cheating prover could convince the verifier with probability $1/2$; this probability may be reduced to $1/2^n$ (at the cost of efficiency) by repeating the protocol n rounds. We will see that a certain class of homomorphisms defined below admits a much larger challenge set, and thus achieves a lower soundness error in a single execution of the protocol.

Definition 6 (Special Homomorphism)**.** We say that a homomorphism $\phi : \mathcal{G} \to \mathcal{H}$ is special if there exists a value $v \in \mathbb{Z} \setminus \{0\}$ (called *special exponent*) and a PPT algorithm that given $x \in \mathcal{H}$ computes $u \in \mathcal{G}$ such that $\phi(u) = x^v$.

To formalize Σ^ϕ-protocols, we extended the language of CertiCrypt with types for the groups $\mathcal{G}, \mathcal{H}$ and operators for computing the group operation, exponentiation/product, and inverse; we also added operators $\phi(\cdot)$, $u(\cdot)$, and a constant expression v denoting the special exponent of the homomorphism as in Definition 6. In addition, we wrote an expression normalizer that simplifies arithmetic expressions by applying the homomorphic property of ϕ; normalization is done as part of the `ep` tactic.

A Σ^ϕ-protocol built from a special homomorphism admits as a challenge set any natural interval of the form $[0..c^+]$, where c^+ is smaller than the smallest prime divisor of the special exponent v. Let p be the smallest prime divisor of v (and assume $|v| \geq 2$), then a maximal c^+ can be chosen as $p - 1$. We provide a functor to construct the largest challenge set for any special homomorphism ϕ; it can then be plugged into the corresponding Σ^ϕ-protocol to minimize the soundness error.

Theorem 7 (Σ^ϕ-protocols for special homomorphisms)**.** *If ϕ is special and c^+ is smaller than any prime divisor of the special exponent v, then the protocol in Definition 5 is a Σ-protocol with challenge set $C = [0..c^+]$.*

Proof:

Completeness: We must prove that a honest prover always succeeds in convincing a verifier, i.e.

$$\forall \mu.\ R(\mu(x), \mu(w)) \Longrightarrow \Pr[\mathsf{Protocol}, \mu : b = \mathsf{true}] = 1$$

Note that this can be reformulated in terms of a program equivalence as follows

$$\mathsf{Protocol}(x, w) \simeq^{R(x,w)}_{\{b\}} b \leftarrow \mathsf{true}$$

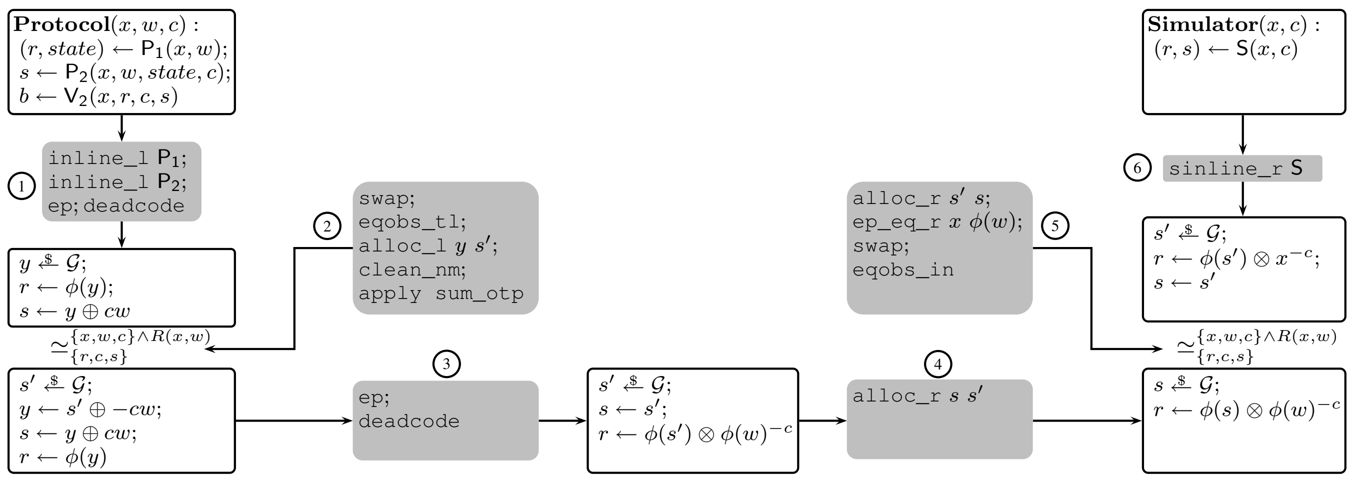
To prove this, we inline all procedure calls in the protocol and simplify the resulting program performing expression propagation, normalization, and dead code elimination. We use the following proof script:

```
inline $\mathsf{P}_1$; inline $\mathsf{P}_2$; inline $\mathsf{V}_1$; inline $\mathsf{V}_2$;
ep; deadcode.
```

The resulting goal has the form

$$\begin{array}{l} y \overset{\$}{\leftarrow} \mathcal{G};\ c \overset{\$}{\leftarrow} [0..c^+]; \\ b \leftarrow \phi(y) \otimes \phi(w)^c = \phi(y) \otimes x^c \end{array} \simeq^{\phi(w)=x}_{\{b\}} \quad b \leftarrow \mathsf{true}$$

We use the tactic `ep_eq` x $\phi(w)$ to simplify the last instruction in the game on the left hand side to $b \leftarrow \mathsf{true}$,



$\textbf{Protocol}(x, w, c)$:
$(r, state) \leftarrow \mathsf{P}_1(x, w)$;
$s \leftarrow \mathsf{P}_2(x, w, state, c)$;
$b \leftarrow \mathsf{V}_2(x, r, c, s)$
1
inline_l P_1;
inline_l P_2;
ep; deadcode
$y \xleftarrow{\$} \mathcal{G}$;
$r \leftarrow \phi(y)$;
$s \leftarrow y \oplus cw$
$\simeq^{\{x,w,c\} \wedge R(x,w)}_{\{r,c,s\}}$
2
swap;
eqobs_tl;
alloc_l y s';
clean_nm;
apply sum_otp
$s' \xleftarrow{\$} \mathcal{G}$;
$y \leftarrow s' \oplus -cw$;
$s \leftarrow y \oplus cw$;
$r \leftarrow \phi(y)$
3
ep;
deadcode
$s' \xleftarrow{\$} \mathcal{G}$;
$s \leftarrow s'$;
$r \leftarrow \phi(s') \otimes \phi(w)^{-c}$
4
alloc_r s s'
alloc_r s' s;
ep_eq_r x $\phi(w)$;
swap;
eqobs_in
5
$\textbf{Simulator}(x, c)$:
$(r, s) \leftarrow \mathsf{S}(x, c)$
6
sinline_r S
$s' \xleftarrow{\$} \mathcal{G}$;
$r \leftarrow \phi(s') \otimes x^{-c}$;
$s \leftarrow s'$
$\simeq^{\{x,w,c\} \wedge R(x,w)}_{\{r,c,s\}}$
$s \xleftarrow{\$} \mathcal{G}$;
$r \leftarrow \phi(s) \otimes \phi(w)^{-c}$

Figure 2. A game-based proof that S is a sHVZK simulator for the Σ^{ϕ}-protocol in Theorem 7.

tactic `deadcode` to remove the first two instructions that are no longer relevant, and `eqobs_in` to conclude.

Special Honest Verifier Zero-Knowledge: The following is a sHVZK simulator for the protocol:

```
Simulator S(x, c) :
  $s \xleftarrow{\$} \mathcal{G}$;
  $r \leftarrow \phi(s) \otimes x^{-c}$;
  return $(r, c, s)$
```

A proof that S perfectly simulates conversations of the protocol is illustrated in Figure 2; we briefly explain the numbered steps in the figure.

1) Similarly to the proof above, we inline calls to P_1 and P_2, and simplify the goal using tactics `ep` and `deadcode`.
2) We introduce an intermediate game using the transitivity of observational equivalence. To prove that the new game is observationally equivalent to the previous one, we first reorder the instructions using `swap` to obtain a common suffix which we then remove using the tactic `eqobs_tl`. The resulting goal is

$$\vdash y \xleftarrow{\$} \mathcal{G} \simeq^{\{x,w,c\} \wedge R(x,w)}_{\{y,w,c\}} s' \xleftarrow{\$} \mathcal{G};\ y \leftarrow s' \oplus -cw$$

Since variables w and c are not modified, we can remove them from the output set using tactic `clean_nm`. We next use tactic `alloc` y s' to sample s' instead of y in the game on the left, and we weaken the precondition to true, which results in the goal

$$\vdash s' \xleftarrow{\$} \mathcal{G};\ y \leftarrow s' \simeq_{\{y\}} s' \xleftarrow{\$} \mathcal{G};\ y \leftarrow s' \oplus -cw$$

This equivalence holds because $-cw$ acts as a one-time pad; we have proved this as a lemma called `sum_otp` that we apply to conclude the proof.
3) Using `ep`, we propagate throughout the code the value assigned to y and then remove the assignment using `deadcode`. The expression normalizer automatically simplifies $(s' \oplus -cw) \oplus cw$ to s', and $\phi(s' \oplus -cw)$ to $\phi(s') \otimes \phi(w)^{-c}$ using the homomorphic property of ϕ.
4) We introduce a new intermediate game; to prove that is equivalent to the previous one, we allocate variable s into s'; the resulting game is identical to the one on the left hand side.
5) We substitute variable s for s' in the game on the right hand side of the equivalence, and use the precondition $R(x, w)$—which boils down to $x = \phi(w)$—to substitute x by $\phi(w)$. The resulting games are identical modulo reordering of instructions.
6) We conclude by inlining the procedure S in the simulation

Soundness: Soundness requires the existence of a PPT algorithm KE that given two accepting conversations (x, r, c_1, s_1), (x, r, c_2, s_2), with $c_1 \neq c_2$, computes w such that $x = \phi(w)$. We propose the following knowledge extractor,

```
KE(x, c_1, c_2, s_1, s_2) :
  $(a, b, d) \leftarrow \mathsf{extended\_gcd}(c_1 - c_2, v)$;
  $w \leftarrow a(s_1 \oplus -s_2) \oplus b\ u(x)$;
  return $w$
```

where extended_gcd efficiently implements the extended Euclidean algorithm. For integers a, b, $\mathsf{extended\_gcd}(a, b)$ computes a triple of integers (x, y, d) such that d is the greatest common divisor of a and b, and x, y satisfy the Bézout's identity

$$ax + by = \gcd(a, b) = d$$

Since all computations done by the knowledge extractor can be efficiently implemented, KE is a PPT algorithm; we prove this in Coq using an automated procedure `PPT_proc` that proves that a program without loops or recursive procedure calls is PPT by computing polynomial bounds for its time and memory footprints, provided expressions appearing in the program are efficiently evaluable. We have to prove as well that KE computes a preimage of the public input x. For two accepting conversations (x, r, c_1, s_1) and (x, r, c_2, s_2), we have

$$\phi(s_1) = r \otimes x^{c_1} \ \wedge \ \phi(s_2) = r \otimes x^{c_2}$$

and thus

$$x^{c_1 - c_2} = \phi(s_1 \oplus -s_2) \tag{1}$$

Furthermore, since ϕ is special we can efficiently compute u such that $x^v = \phi(u)$. The triple (a, b, d) given by the extended Euclidean algorithm satisfies the Bézout's identity

$$a(c_1 - c_2) + bv = \gcd(c_1 - c_2, v) = d \tag{2}$$

Both c_1 and c_2 are bounded by c^+, which is in turn smaller than the smallest prime that divides v. Thus, no divisor of $|c_1 - c_2|$ can divide v and $d = \gcd(c_1 - c_2, v) = 1$. In addition, since ϕ is a homomorphism, from (1) and (2) we conclude

$$\phi(w) = \phi(a(s_1 \oplus -s_2) \oplus bu) = x^{a(c_1 - c_2)} \otimes x^{bv} = x$$

■

A. Concrete instances of Σ^ϕ-protocols

We have formalized several Σ^ϕ-protocols using the functor described in the previous section. For each protocol, we specify the groups $\mathcal{G}, \mathcal{H}$ and the underlying special homomorphism $\phi : \mathcal{G} \to \mathcal{H}$, and provide appropriate interpretations for the operator $u(\cdot)$ and the constant special exponent v. Table I summarizes all the protocols that we have formalized.

The Schnorr [20] and Okamoto [21] protocols are based on the discrete logarithm problem. For prime numbers p and q such that q divides $p-1$, a Schnorr group is a multiplicative subgroup of $\mathbb{Z}_p^*$ of order q with generator g. A Σ-protocol for proving knowledge of discrete logarithms in the Schnorr group is obtained by instantiating the construction of Definition 5 with the homomorphism $\phi : \mathbb{Z}_q^+ \to \mathbb{Z}_p^*$ defined as $\phi(x) = g^x$. Since the order q of the Schnorr group is known, it suffices to take q as the special exponent of the homomorphism, and $u(x) = 0$ for all $x \in \mathbb{Z}_q^+$. The Okamoto protocol is similar to Schnorr protocol but it works with two Schnorr subgroups of $\mathbb{Z}_p^*$ with generators g_1 and g_2, respectively (it can be naturally generalized to any number of generators). In this case ϕ maps a pair (x_1, x_2) to $g_1^{x_1} \otimes g_2^{x_2}$.

Let N be an RSA modulus with prime factors p and q, and let e be a public exponent; e must be co-prime with the totient $\varphi = (p-1)(q-1)$ (i.e. $\gcd(e, \varphi(N)) = 1$). The Guillou-Quisquater [22], Fiat-Shamir [23], and Feige-Fiat-Shamir [24] protocols are based on the difficulty of solving the RSA problem: given N, e and $y \equiv x^e \mod N$, compute x, the e^{th}-root of y modulo N. The Guillou-Quisquater protocol is obtained by taking $\phi : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$, $\phi(x) = x^e$. The Fiat-Shamir protocol is obtained as a special case when $e = 2$. The Feige-Fiat-Shamir is obtained by taking $\phi : \{-1, 1\} \times \mathbb{Z}_N^* \to \mathbb{Z}_N^*$, $\phi(s, x) = s.x^2$.

Remark. We note that our results hold independently of any computational assumption. Certainly, it is the difficulty of inverting the underlying homomorphism what makes a Σ^ϕ-protocol interesting, but this is inessential for establishing the properties we prove about the protocol.

B. Composition of Σ^ϕ-protocols

Let $\phi_1 : \mathcal{G}_1 \to \mathcal{H}_1$ and $\phi_2 : \mathcal{G}_2 \to \mathcal{H}_2$ be two special homomorphisms with special exponents v_1, v_2 and associated algorithms u_1, u_2, respectively. We give below two useful ways of combining the Σ^ϕ-protocols induced by these homomorphisms.

Theorem 8 (Direct Product of Special Homomorphisms)**.** *The following homomorphism from the direct product of $\mathcal{G}_1$ and $\mathcal{G}_2$ to the direct product of $\mathcal{H}_1$ and $\mathcal{H}_2$ is a special homomorphism:*[1]

$$\begin{array}{lcl} \phi & : & \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{H}_1 \times \mathcal{H}_2 \\ \phi(x_1, x_2) & \stackrel{\text{def}}{=} & (\phi_1(x_1), \phi_2(x_2)) \end{array}$$

Proof: It suffices to take

$$\begin{array}{lcl} v & \stackrel{\text{def}}{=} & \mathrm{lcm}(v_1, v_2) \\ u(x_1, x_2) & \stackrel{\text{def}}{=} & (u_1(x_1)^{v/v_1}, u_2(x_2)^{v/v_2}) \end{array}$$

Indeed,

$$\begin{array}{lcl} \phi(u(x_1, x_2)) & = & (\phi_1(u_1(x_1)^{v/v_1}), \phi_2(u_2(x_2)^{v/v_2})) \\ & = & (x_1^{v_1 v/v_1}, x_2^{v_2 v/v_2}) \\ & = & (x_1, x_2)^v \end{array}$$

■

Theorem 9 (Equality of Preimages)**.** *Suppose that the domain of both homomorphisms is the same, $\mathcal{G}_1 = \mathcal{G}_2 = \mathcal{G}$, $v_1 = v_2$, and u_1, u_2 are such that*

$$\forall x_1 \in H_1,\ x_2 \in H_2.\ u_1(x_1) = u_2(x_2)$$

Then, the following homomorphism from $\mathcal{G}$ to the direct product of $\mathcal{H}_1$ and $\mathcal{H}_2$ is a special homomorphism:

$$\begin{array}{lcl} \phi & : & \mathcal{G} \to \mathcal{H}_1 \times \mathcal{H}_2 \\ \phi(x) & \stackrel{\text{def}}{=} & (\phi_1(x), \phi_2(x)) \end{array}$$

Proof: Take $v \stackrel{\text{def}}{=} v_1$ and $u(x_1, x_2) \stackrel{\text{def}}{=} u_1(x_1)$,

$$\begin{array}{lcl} \phi(u(x_1, x_2)) & = & (\phi_1(u_1(x_1)), \phi_2(u_2(x_2))) \\ & = & (x_1^{v_1}, x_2^{v_2}) \\ & = & (x_1, x_2)^v \end{array}$$

[1]This yields an effective means of AND-combining assertions proved by Σ^ϕ-protocols. The result generalizes the protocol in [9, Theorem 6.2]; we do not require that the special exponent be the same.

Protocol	G	H	ϕ	u	v
Schnorr	$\mathbb{Z}_q^+$	$\mathbb{Z}_p^*$	$x \mapsto g^x$	$x \mapsto 0$	q
Okamoto	$(\mathbb{Z}_q^+, \mathbb{Z}_q^+)$	$\mathbb{Z}_p^*$	$(x_1, x_2) \mapsto g_1^{x_1} \otimes g_2^{x_2}$	$x \mapsto (0, 0)$	q
Fiat-Shamir	$\mathbb{Z}_N^*$	$\mathbb{Z}_N^*$	$x \mapsto x^2$	$x \mapsto x$	2
Guillou-Quisquater	$\mathbb{Z}_N^*$	$\mathbb{Z}_N^*$	$x \mapsto x^e$	$x \mapsto x$	e
Feige-Fiat-Shamir	$\{-1, 1\} \times \mathbb{Z}_N^*$	$\mathbb{Z}_N^*$	$(s, x) \mapsto s.x^2$	$x \mapsto (1, x)$	2

Table I
SPECIAL HOMOMORPHISMS IN SELECTED Σ^ϕ-PROTOCOLS.

■

We can use this latter theorem to construct a Σ-protocol that proves correctness of Diffie-Hellman keys. Given a group with prime order q and a generator g, this amounts to prove that triples of group elements of the form (α, β, γ) are Diffie-Hellman triples, i.e. that if $\alpha = g^a$ and $\beta = g^b$, then $\gamma = g^{ab}$. We instantiate the above construction for homomorphisms $\phi_1(x) = g^x$, and $\phi_2(x) = \beta^x$. Knowledge of a preimage a of (α, γ) implies that (α, β, γ) is a Diffie-Hellman triple (and thus that γ is a valid Diffie-Hellman shared key).

V. SIGMA PROTOCOLS BASED ON CLAW-FREE PERMUTATIONS

This section describes a general construction in the same flavor as the Σ^ϕ construction discussed in the previous section, but based on pairs of *claw-free* permutations [4] rather than on special homomorphisms.

Definition 10 (Trapdoor Permutation)**.** A family of *trapdoor permutations* is a triple $(\mathcal{KG}, f, f^{-1})$, where $\mathcal{KG}$ is a randomized key generator procedure that generates pair of keys of the form (pk, sk), such that $f(pk, \cdot)$ is a permutation, and $f^{-1}(sk, \cdot)$ is its inverse.

Definition 11 (Claw-Free Permutation Pair [1])**.** A pair of trapdoor permutations $f = (f_0, f_1)$ on the same domain D is claw-free if it is unfeasible to compute $x, y \in D$ such that $f_0(x, pk) = f_1(y, pk)$.

Given a claw-free permutation pair f, and a bitstring $a \in \{0, 1\}^k$, we define

$$f_{[a]}(b) \overset{\text{def}}{=} f_{a_1}(f_{a_2}(\ldots(f_{a_k}(b))\ldots))$$

where a_i denotes the i^{th} bit of a.

Theorem 12 (Σ-protocol Based on Claw-Free Permutations)**.** *Let (f_0, f_1) be a pair of claw-free permutations on D and let R be such that*

$$\begin{aligned} R(pk, sk) \iff \\ \forall x, f_0(pk, f_0^{-1}(sk, x)) = f_0^{-1}(sk, f_0(pk, x)) = x \ \wedge \\ f_1(pk, f_1^{-1}(sk, x)) = f_1^{-1}(sk, f_1(pk, x)) = x \end{aligned}$$

The following protocol is a Σ-protocol for relation R:

$$\begin{aligned} \mathsf{P}_1(pk, sk) &\overset{def}{=} y \xleftarrow{\$} D;\ \mathsf{return}\ (y, y) \\ \mathsf{P}_2(pk, sk, y, c) &\overset{def}{=} \mathsf{return}\ f_{[c]}^{-1}(sk, y) \\ \mathsf{V}_1(pk, r) &\overset{def}{=} c \xleftarrow{\$} \{0, 1\}^k;\ \mathsf{return}\ c \\ \mathsf{V}_2(pk, r, c, s) &\overset{def}{=} \mathsf{return}\ \left(f_{[c]}(pk, s) = r\right) \end{aligned}$$

except that it might not satisfy the knowledge soundness property.

Proof:

Completeness: The proof follows almost the same structure as the completeness proof for Σ^ϕ-protocols. After inlining procedure calls in the protocol, we are left with the goal

$$b \leftarrow f_{[c]}(pk, f_{[c]}^{-1}(sk, y)) = y \simeq_{\{b\}}^{R(pk,sk)} b \leftarrow \mathsf{true}$$

We use the fact that the pair (pk, sk) is in R to prove that $f_{[c]}(pk, f_{[c]}^{-1}(sk, y)) = y$ by induction on c.

Special Honest Verifier Zero-Knowledge: The following is a sHVZK simulator for the protocol,

```
Simulator S(pk, c) :
  s ←$ D;
  r ← f_[c](pk, s);
  return (r, s)
```

To prove that

$$\mathsf{Protocol}(pk, sk, c) \simeq_{\{r,c,s\}}^{\{pk,c\} \wedge R(pk,sk)} (r, s) \leftarrow \mathsf{S}(pk, c)$$

we inline every procedure call in both games and perform expression propagation and deadcode elimination, we are left with the following goal:

$$\begin{array}{l} r \xleftarrow{\$} D; \\ s \leftarrow f_{[c]}^{-1}(sk, r) \end{array} \simeq_{\{r,s\}}^{\{pk,sk,c\}} \begin{array}{l} s \xleftarrow{\$} D; \\ r \leftarrow f_{[c]}(pk, s) \end{array}$$

which is provable using the fact f is a permutation pair. ■

We observed that the above protocol does not necessarily satisfy the special soundness property. Instead, it satisfies a property known as *collision intractability*: no efficient algorithm can find two accepting conversations with different challenges but same commitment (a collision) with non-negligible probability. Interactive proof protocols that are complete, sHVZK but only satisfy *collision intractability* have important applications as signature protocols.

Theorem 13. *It is unfeasible to find a collision for the protocol in Theorem 12.*

Proof: By contradiction. Assume two accepting conversations (r, c_1, s_1), (r, c_2, s_2) for a public input pk with $c_1 \neq c_2$. We show that it is possible to efficiently compute a claw (b, b') such that $f_0(b) = f_1(b')$. Since the two conversations are accepting,

$$f_{[c_1]}(pk, s_1) = f_{[c_2]}(pk, s_2) = r$$

The following algorithm computes a claw

```
find_claw($s_1, c_1, s_2, c_2$) :
  if head($c_1$) = head($c_2$)
  then find_claw(tail($c_1$), $s_1$, tail($c_2$), $s_2$)
  else if head($c_1$) = 0
          then($f_{[\mathsf{tail}(c_1)]}(pk, s_1), f_{[\mathsf{tail}(c_2)]}(pk, s_2)$)
          else ($f_{[\mathsf{tail}(c_2)]}(pk, s_2), f_{[\mathsf{tail}(c_1)]}(pk, s_1)$)
```

The algorithm executes in polynomial-time provided permutations f_0 and f_1 can be evaluated in polynomial time, and c_1, c_2 are polynomially bounded. For a polynomially bounded challenge set, this contradicts the assumption that (f_0, f_1) is claw-free. ■

A. *A Σ-protocol based on a family of claw-free permutations*

Goldwasser, Micali and Rivest proved that claw-free pairs of permutations exist provided factoring integers is hard [1]; Cramer [4] used this construction to define a Σ-protocol. We next overview a formalization we developed in Coq of this family of claw-free trapdoor permutations for which we instantiate the protocol in Theorem 12.

Recall that the Jacobi symbol for an integer a and an odd prime p is defined as follows[2]

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \pmod{p} \\ +1 & \text{if } a \not\equiv 0 \pmod{p} \text{ and } a \text{ is a perfect square} \\ -1 & \text{otherwise} \end{cases}$$

The Jacobi symbol of an integer a and a composite number is defined as the product of the Jacobi sybmols of its prime factors, so that if $N = pq$,

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$$

Consider two distinct primes p and q, such that $p \equiv 3 \pmod{8}$, and $q \equiv 7 \pmod{8}$, and define $n = pq$—such n is usually called a Blum integer. It follows that -1 is a non-quadratic residue modulo p and q, and thus

$$\left(\frac{-1}{n}\right) = 1 \quad \text{and} \quad \left(\frac{2}{n}\right) = -1$$

For any such integer n, define

$$D_n \stackrel{\text{def}}{=} \left\{ x \;\middle|\; 0 < x < \tfrac{n}{2} \wedge \left(\frac{x}{n}\right) = 1 \right\}$$

[2]The Jacobi symbol reduces to the Legendre symbol in this case.

The following pair of permutations on D_n is claw-free:

$$f_0(x) \stackrel{\text{def}}{=} \begin{cases} x^2 \bmod n & \text{if } 0 < x^2 \bmod n < n/2 \\ -x^2 \bmod n & \text{if } n/2 < x^2 \bmod n < n \end{cases}$$

$$f_1(x) \stackrel{\text{def}}{=} \begin{cases} 4x^2 \bmod n & \text{if } 0 < 4x^2 \bmod n < n/2 \\ -4x^2 \bmod n & \text{if } n/2 < 4x^2 \bmod n < n \end{cases}$$

If the prime factors of n are known, then these permutations can be efficiently inverted by computing square roots in $\mathbb{Z}_n^*$ and applying the Chinese Remainder Theorem; moreover, it can be shown that from a claw (x, y), the prime factors p and q can be efficiently extracted [4]. Therefore, we can use the construction in Theorem 12 to obtain a Σ-protocol for the relation $R(n, (p, q)) \stackrel{\text{def}}{=} n = pq$.

VI. Combination of Sigma-protocols

There are two immediate, but essential, ways of combining two Σ-protocols (P^1, V^1) and (P^2, V^2) with knowledge relations R_1 and R_2 respectively: AND-combination, and OR-combination. The former allows a prover to prove knowledge of witnesses w_1, w_2 such that $(x_1, w_1) \in R_1$ and $(x_2, w_2) \in R_2$. The latter allows a prover to prove knowledge of a witness w such that either $(x_1, w) \in R_1$ or $(x_2, w) \in R_2$, without revealing which is the case. This can be naturally extended to proofs of any monotone Boolean formula by nested combination (although there exist direct, more efficient constructions). Even though simple, such constructions are incredibly powerful and form the basis of many practical protocols, like secure electronic voting protocols [4].

A. *AND-combination*

Two Σ-protocols can be combined into a Σ-protocol that proves simultaneous knowledge of witnesses for both underlying knowledge relations, i.e. a Σ-protocol with a knowledge relation:

$$R \stackrel{\text{def}}{=} \{((x_1, x_2), (w_1, w_2)) \mid (x_1, w_1) \in R_1 \wedge (x_2, w_2) \in R_2\}$$

We have formalized a functor AND, that combines two public-coin Σ-protocols (P^1, V^1) and (P^2, V^2) in this form. Without loss of generality, we assume that both protocols mandate that honest verifiers choose challenges uniformly from a set of bitstrings of a certain length k. The construction is straightforward; the combination is essentially a parallel composition of the two sub-protocols using the same randomly chosen challenge:

```
$\mathsf{P}_1((x_1,x_2),(w_1,w_2)) \overset{\text{def}}{=}$
  $(r_1, state_1) \leftarrow \mathsf{P}_1^1(x_1,w_1);$
  $(r_2, state_2) \leftarrow \mathsf{P}_1^2(x_2,w_2);$
  return $((r_1,r_2),(state_1,state_2))$
$\mathsf{P}_2((x_1,x_2),(w_1,w_2),state_1,state_2,c) \overset{\text{def}}{=}$
  $s_1 \leftarrow \mathsf{P}_2^1(x_1,w_1,state_1,c);$
  $s_2 \leftarrow \mathsf{P}_2^2(x_2,w_2,state_1,c);$
  return $(s_1,s_2)$
$\mathsf{V}_1((x_1,x_2),(r_1,r_2)) \overset{\text{def}}{=} c \overset{\$}{\leftarrow} \{0,1\}^k;$ return $c$
$\mathsf{V}_2((x_1,x_2),(r_1,r_2),c,(s_1,s_2)) \overset{\text{def}}{=}$
  $b_1 \leftarrow \mathsf{V}_2^1(x_1,r_1,c,s_1)$
  $b_2 \leftarrow \mathsf{V}_2^2(x_2,r_2,c,s_2)$
  return $(b_1 = \mathsf{true} \wedge b_2 = \mathsf{true})$
```

Observe that V_1 is not built from V_1^1 and V_1^2. The reason for this is that in order to prove soundness, two runs of the protocol for the same public input x with the same commitment r, but with different challenges $c \neq c'$, must yield two runs of each of the sub-protocols with distinct challenges. If the challenge for the main protocol were built from the challenges computed by V_1^1 and V_1^2, e.g. by concatenating them, we would not be able to conclude that the challenges in each pair of conversations extracted for the sub-protocols are different—one could only conclude that this is the case for one of the sub-protocols. Instead, we make use of the public-coin property and simply draw in V_1 a new random challenge that is used in both sub-protocols. This solves the above problem, but also requires that the sub-protocols satifsy the special honest verifier zero-knowledge property, since we need to be able to simulate the sub-protocols for any fixed challenge.

Since AND combination essentially amounts to pairing the two sub-protocols while respecting the structure of a Σ-protocol, all proofs have the same general form: procedure calls are first inlined, and then the goal is manipulated using program transformations to put it in a form where the properties of the sub-protocols can be applied to conclude. We give below a proof sketch of sHVZK and special soundness; a more detailed proof of these properties and a proof of completeness can be found in Appendix A.

sHVZK: The sHVZK simulator for the protocol simply runs the simulators of the sub-protocols to obtain a conversation for each sub-protocol with the same challenge c, the conversations are then combined to obtain a conversation of the main protocol:

```
**Simulator** $\mathsf{S}((x_1,x_2),c)$ :
  $(r_1,s_1) \leftarrow \mathsf{S}_1(x_1,c);$
  $(r_2,s_2) \leftarrow \mathsf{S}_2(x_2,c);$
  return$((r_1,r_2),(s_1,s_2))$
```

Soundness: Soundness requires us to give a PPT knowledge extractor that computes a witness for the knowledge relation R from two accepting runs of the protocol with different challenges but the same commitment. This amounts to computing a witness for each of the sub-protocols and can be done using the corresponding knowledge extractors as follows:

```
$\mathsf{KE}((x_1,x_2),(r_1,r_2),c,c',(s_1,s_2),(s_1',s_2'))$ :
  $w_1 \leftarrow \mathsf{KE}^1(x_1,r_1,c,c',s_1,s_1');$
  $w_2 \leftarrow \mathsf{KE}^2(x_2,r_2,c,c',s_2,s_2');$
  return $(w_1,w_2)$
```

Note that since we use the challenge for the main protocol as the challenge for the underlying sub-protocols, for each sub-protocol we can extract two accepting runs with different challenges since $c \neq c'$. Concretely, from

$$\Pr\left[b \leftarrow \mathsf{V}_2((x_1,x_2),(r_1,r_2),c\ ,(s_1,s_2)),\mu : b = \mathsf{true}\right] = 1$$
$$\Pr\left[b \leftarrow \mathsf{V}_2((x_1,x_2),(r_1,r_2),c',(s_1',s_2')),\mu : b = \mathsf{true}\right] = 1$$

we can prove that for $i = 1,2$,

$$\Pr\left[w_i \leftarrow \mathsf{KE}^i(x_i,r_i,c,c',s_i,s_i'),\mu : (x_i,w_i) \in R_i\right] = 1$$

from the soundness of the sub-protocols and from the fact that

$$\Pr\left[b_i \leftarrow \mathsf{V}_2^i(x_i,r_i,c\ ,s_i),\mu : b_i = \mathsf{true}\right] = 1$$
$$\Pr\left[b_i \leftarrow \mathsf{V}_2^i(x_i,r_i,c',s_i'),\mu : b_i = \mathsf{true}\right] = 1$$

B. OR-combination

Two Σ-protocols can also be combined to obtain a protocol that proves knowledge of a witness for the knowledge relation of one of the sub-protocols, but without revealing which. The construction relies on the ability to simulate accepting runs; the basic idea is that the prover runs the real protocol for which she knows a witness, and uses the simulator to generate a run of the other protocol. The knowledge relation suggested in, e.g. [25],

$$\hat{R} \overset{\text{def}}{=} \{((x_1,x_2),w) \mid (x_1,w) \in R_1 \ \vee (x_2,w) \in R_2\}$$

suffers from placing unrealistic demands on the simulator. As pointed out in [4], it is important to allow the simulator to fail on an input $x \notin \mathsf{dom}(R)$. However, in order to prove completeness for the above relation, the simulator must be able to perfectly simulate outside the domain of the respective knowledge relation. Instead, we can prove completeness (and sHVZK) of the combination with respect to a knowledge relation whose domain is restricted to the Cartesian product of the domains of the knowledge relations of the sub-protocols, i.e.

$$R \overset{\text{def}}{=} \left\{((x_1,x_2),w) \;\middle|\; \begin{array}{l} ((x_1,w) \in R_1 \wedge x_2 \in \mathsf{dom}(R_2)) \ \vee \\ ((x_2,w) \in R_2 \wedge x_1 \in \mathsf{dom}(R_1)) \end{array} \right\}$$

Unfortunately, we cannot prove soundness with respect to R, we can only prove it with respect to $\hat{R}$. The reason for this is that an accepting run of the combined protocol only guarantees the existence of a witness for the public input of one of the protocols, the simulation of the other protocol may succeed even if the input is not in the domain of the respective relation. Put more technically, from two accepting runs of the combined protocol with distinct challenges we might not be able to extract two accepting runs with distinct challenges for each of the sub-protocols; we can only guarantee we can do that for one of them. Observe that we do not really lose anything by proving completeness with respect to the smaller relation R. If we admitted pairs (x_1, x_2) as public input where one component does not belong to the domain of the corresponding knowledge relation, we would not be able to say anything about the success of the simulator. The simulator might as well fail, trivially revealing that the prover could not have known a witness for the corresponding input, and rendering the protocol pointless for such inputs.

Compared to the AND combination, the OR combination is harder to fit into the structure of a Σ-protocol. The reason for this is that the first phase of the prover needs to use the simulator of one of the protocols, which results in a full (accepting) run that has to be passed over throughout the whole execution of the protocol. Given $(x_1, w) \in R_1$, the OR prover runs the prover of the first protocol and the simulator of the second, and returns as a commitment a pair with the commitments of each protocol; it passes over in the state the challenge and the reply of the simulated conversation,

```
$\mathsf{P}_1((x_1, x_2), w) \stackrel{\text{def}}{=}$
    if $(x_1, w) \in R_1$ then
        $(r_1, state_1) \leftarrow \mathsf{P}^1_1(x_1, w)$;
        $c_2 \xleftarrow{\$} \{0,1\}^k$;
        $(r_2, s_2) \leftarrow \mathsf{S}_2(x_2, c_2)$;
        $state \leftarrow (state_1, c_2, s_2)$
    else
        $(r_2, state_2) \leftarrow \mathsf{P}^2_1(x_2, w)$;
        $c_1 \xleftarrow{\$} \{0,1\}^k$;
        $(r_1, s_1) \leftarrow \mathsf{S}_1(x_1, c_1)$;
        $state \leftarrow (state_2, c_1, s_1)$
    return $((r_1, r_2), state)$
```

Above, the test $(x_1, w) \in R_1$ is an encoding of the fact that the prover knows to which knowledge relation corresponds the witness w, and thus which protocol she can run for real, while simulating the other one. The commitment (r_1, r_2) is passed along to the verifier that simply replies by returning a randomly chosen bitstring to the prover, the combination is a public-coin protocol,

$$\mathsf{V}_1((x_1, x_2), (r_1, r_2)) \stackrel{\text{def}}{=} c \xleftarrow{\$} \{0,1\}^k; \text{ return } c$$

Assume without loss of generality that $(x_1, w) \in R_1$. In the second phase the prover constructs the challenge for the first protocol by xor-ing the challenge c of the OR protocol with the challenge used in the simulation of the second protocol in the first phase. It then runs the second phase of the prover of the first protocol to compute a reply. The result of the second phase is constructed from the challenges for each protocol and the prover replies (the ones coming from the simulated protocol are recovered from the state):

```
$\mathsf{P}_2((x_1, x_2), w, (state, c', s), c) \stackrel{\text{def}}{=}$
  if $(x_1, w) \in R_1$ then
    $state_1 \leftarrow state;\ c_2 \leftarrow c';\ s_2 \leftarrow s$;
    $c_1 \leftarrow c_2 \oplus c$;
    $s_1 \leftarrow \mathsf{P}^1_2(x_1, w, state_1, c_1)$
  else
    $state_2 \leftarrow state;\ c_1 \leftarrow c';\ s_1 \leftarrow s$;
    $c_2 \leftarrow c_1 \oplus c$;
    $s_2 \leftarrow \mathsf{P}^2_2(x_2, w, state_2, c_2)$
  return $((c_1, s_1), (c_2, s_2))$
```

The verifier accepts a conversation when the runs of both protocols are accepting and the challenge is the xor of the challenges used in each of the combined protocols,

```
$\mathsf{V}_2((x_1, x_2), (r_1, r_2), c, ((c_1, s_1), (c_2, s_2))) \stackrel{\text{def}}{=}$
$b_1 \leftarrow \mathsf{V}^1_2(x_1, r_1, c_1, s_1)$;
$b_2 \leftarrow \mathsf{V}^2_2(x_2, r_2, c_2, s_2)$;
return $(c = c_1 \oplus c_2 \wedge b_1 = \mathsf{true} \wedge b_2 = \mathsf{true})$
```

Completeness: The proof is slightly more involved than the proof for the AND combination, since only one of the protocols is run for real, while the other is just simulated, and this depends on the knowledge of the prover. Thus, the proof is split into two cases:

- **case** $(x_1, w) \in R_1$: the outline of the proof is as follows:

$$\begin{aligned}\mathsf{Protocol}((x_1, x_2), w) &\simeq \mathsf{Protocol}_1(x_1, w);\\ &\quad c_2 \xleftarrow{\$} \{0,1\}^k; (r_2, s_2) \leftarrow \mathsf{S}_2(x_2, c_2)\\ &\simeq \mathsf{Protocol}_1(x_1, w); \mathsf{Protocol}_2(x_2, w')\end{aligned}$$

  The first equivalence is immediate from inlining and simplification. The second equivalence follows from the fact that $\exists w', R_2(x_2, w')$ and the sHVZK property of the second protocol. The proof concludes by application of the completeness property of each of the sub-protocols.
- **case** $(x_2, w) \in R_2$: Idem.

sHVZK: The simulator for the OR combination is easily built from the simulators of the sub-protocols.

```
**Simulator** $\mathsf{S}((x_1, x_2), c)$ :
$c_2 \xleftarrow{\$} \{0,1\}^k$;
$c_1 \leftarrow c \oplus c_2$;
$(r_1, s_1) \leftarrow \mathsf{S}_1(x_1, c_1)$;
$(r_2, s_2) \leftarrow \mathsf{S}_2(x_2, c_2)$;
return $((r_1, r_2), ((c_1, s_1), (c_2, s_2)))$
```

As before, the proof is split into two cases.

- **case** $(x_1, w) \in R_1$:

$$\begin{aligned}\mathsf{Protocol}((x_1, x_2), w) &\simeq \mathsf{Protocol}_1(x_1, w);\ \mathsf{S}_2(x_2)\\ &\simeq \mathsf{S}_1(x_1);\ \mathsf{S}_2(x_2)\\ &\simeq \mathsf{S}((x_1, x_2), c)\end{aligned}$$

  Where the first and last steps are immediate from inlining, and simplification, whereas the second step is a direct application of the HVZK property of S_1 (which follows from sHVZK by Theorem 3).
- **case** $(x_2, w) \in R_2$: Idem.

Soundness: Unlike the AND combination, the OR combination does not have the property that runs with distinct challenges guarantee that the challenges used in the sub-protocols are distinct as well. This is not as problematic as in the case of the AND combination, since it suffices to compute a w such that either $(x_1, w) \in R_1$ or $(x_2, w) \in R_2$. Furthermore, from

$$c = c_1 \oplus c_2 \neq c' = c'_1 \oplus c'_2$$

we have either $c_1 \neq c'_1$ or else $c_1 = c'_1$, in which case necessarily $c_2 \neq c'_2$. Thus, the knowledge extractor simply needs to do a case analysis:

```
KE($(x_1, x_2), (r_1, r_2), c, c',$
   $((c_1, s_1), (c_2, s_2)), ((c'_1, s'_1), (c'_2, s'_2))$) :
if $c_1 \neq c'_1$ then
  $w \leftarrow \mathsf{KE}^1(x_1, r_1, c_1, c'_1, s_1, s'_1)$
else
  $w \leftarrow \mathsf{KE}^2(x_2, r_2, c_2, c'_2, s_2, s'_2)$
return $w$
```

Assume two accepting runs of the combined protocol with the same commitment and $c \neq c'$:

$$((x_1, x_2), (r_1, r_2), c, ((c_1, s_1), (c_2, s_2)))$$
$$((x_1, x_2), (r_1, r_2), c', ((c'_1, s'_1), (c'_2, s'_2)))$$

We have to establish that for an $i \in 1, 2$,

$$\Pr\left[w_i \leftarrow \mathsf{KE}^i(x_i, r_i, c_i, c'_i, s_i, s'_i) : (x_i, w_i) \in R_i\right] = 1$$

depending on whether $c_1 \neq c'_1$ or $c_1 = c'_1 \wedge c_2 \neq c'_2$,

- **case** $c_1 \neq c'_1$: From the special soundness of $\mathsf{Protocol}_1$,

$$\Pr\left[w_1 \leftarrow \mathsf{KE}^1(x_1, r_1, c_1, c'_1, s_1, s'_1) : (x_1, w_1) \in R_1\right] = 1$$

- **case** $c_1 = c'_1$ (and thus $c_2 \neq c'_2$): From the special soundness of $\mathsf{Protocol}_2$

$$\Pr\left[w_2 \leftarrow \mathsf{KE}^2(x_2, r_2, c_2, c'_2, s_2, s'_2) : (x_2, w_2) \in R_2\right] = 1$$

VII. Conclusion

In this article we have presented a formalization of Σ-protocols in CertiCrypt. The highlights of our formalization are its generic account of the class of Σ^ϕ-protocols and the detailed treatment of the AND/OR composition. Our work complements recent advances in the field, and takes a first but important step towards formalizing a rich theory of zero-knowledge proofs. In our opinion, and judging by the myriad of small variations in definitions we have found in the literature, this effort would be worth pursuing for a field that strives for definitional clarity and consistency.

Compared to other applications of CertiCrypt, like the verification of security proofs of encryption and signature schemes [5]–[7], the formalization presented here imposes challenges of a different nature to the user. In contrast to earlier case studies, for which we have developed a mature set of techniques that mechanize most of the reasoning patterns appearing in proofs, we found that the formalization of Σ-protocols does not require as much complex probabilistic reasoning, but is more demanding with respect to the compositionality of proofs. This led us to revise some design choices of CertiCrypt and has given us ideas on how to improve the framework so that results can be reused and composed more easily. For instance, when composing proofs of observational equivalence statements the user often needs to manually rename variables to match the names of the context where the proof is being reused; currently the user has to appeal to the `alloc` tactic to do this, but a simply heuristic may suffice in most cases.

We can build on the existing formalization to verify other important results about zero-knowledge proofs. These include other means of composing protocols: sequential [26] and concurrent [27], [28] composition; transforming public-coin zero-knowledge proofs in general zero-knowledge proofs [2], or different formulations like non-interactive zero-knowledge proofs [29] or properties, i.e. statistical zero-knowledge and computational zero-knowledge instead of perfect zero-knowledge. Moreover, Σ-protocols form the base for a number of important and intriguing protocols for electronic voting schemes [4], identity schemes [4], and commitment schemes [4], [30]. All are prime targets for future formalizations.

Acknowledgment. We would like to thank Dominique Unruh for pointing out that our original definition of soundness was weaker than the standard one, and that under the standard definition the OR combination can only be proved sound with respect to a bigger relation.

References

[1] S. Goldwasser, S. Micali, and C. Rackoff, “The knowledge complexity of interactive proof systems,” *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989.

[2] O. Goldreich, “Zero-knowledge twenty years after its invention,” Electronic Colloquium on Computational Complexity, Tech. Rep. TR02-063, 2002.

[3] M. Backes, M. P. Grochulla, C. Hritcu, and M. Maffei, “Achieving security despite compromise using zero-knowledge,” in *22nd IEEE Computer Security Foundations Symposium, CSF 2009.* IEEE Computer Society, 2009, pp. 308–323.

[4] R. Cramer, “Modular design of secure yet practical cryptographic protocols,” Ph.D. dissertation, CWI and Uni. of Amsterdam, 1996.

[5] G. Barthe, B. Grégoire, and S. Zanella Béguelin, “Formal certification of code-based cryptographic proofs,” in *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009.* New York: ACM, 2009, pp. 90–101.

[6] S. Zanella Béguelin, B. Grégoire, G. Barthe, and F. Olmedo, “Formally certifying the security of digital signature schemes,” in *30th IEEE Symposium on Security and Privacy, S&P 2009.* Los Alamitos, Calif.A: IEEE Computer Society, 2009, pp. 237–250.

[7] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin, "Formal certification of ElGamal encryption. A gentle introduction to CertiCrypt," in *5th International Workshop on Formal Aspects in Security and Trust, FAST 2008*, ser. Lecture Notes in Computer Science, vol. 5491. Berlin: Springer, 2009, pp. 1–19.

[8] The Coq development team, "The Coq Proof Assistant Reference Manual Version 8.2," Online – http://coq.inria.fr, 2009.

[9] U. Maurer, "Unifying zero-knowledge proofs of knowledge," in *Progress in Cryptology – AFRICACRYPT 2009*, ser. Lecture Notes in Computer Science, vol. 5580. Springer, 2009, pp. 272–286.

[10] E. Bangerter, J. Camenisch, S. Krenn, A.-R. Sadeghi, and T. Schneider, "Automatic generation of sound zero-knowledge protocols," Cryptology ePrint Archive, Report 2008/471, 2008.

[11] E. Bangerter, J. Camenisch, and S. Krenn, "Efficiency limitations for Sigma-protocols for group homomorphisms," in *7th Theory of Cryptography Conference, TCC 2010*, ser. Lecture Notes in Computer Science, vol. 5978. Springer, 2010, pp. 553–571.

[12] X. Leroy, "Formal certification of a compiler back-end, or: programming a compiler with a proof assistant," in *33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006*. New York: ACM, 2006, pp. 42–54.

[13] G. Barthe, B. Grégoire, C. Kunz, and T. Rezk, "Certificate translation for optimizing compilers," *ACM Trans. Program. Lang. Syst.*, vol. 31, no. 5, pp. 1–45, 2009.

[14] M. Backes, M. Maffei, and D. Unruh, "Zero-knowledge in the applied pi-calculus and automated verification of the Direct Anonymous Attestation protocol," in *29th IEEE Symposium on Security and Privacy, S&P 2008*. IEEE Computer Society, 2008, pp. 202 –215.

[15] M. Backes, C. Hritcu, and M. Maffei, "Type-checking zero-knowledge," in *15th ACM Conference on Computer and Communications Security, CCS 2008*. ACM, 2008, pp. 357–370.

[16] M. Abadi and P. Rogaway, "Reconciling two views of cryptography (The computational soundness of formal encryption)," *J. Cryptology*, vol. 15, no. 2, pp. 103–127, 2002.

[17] V. Cortier and B. Warinschi, "Computationally sound, automated proofs for security protocols," in *Programming Languages and Systems, 14th European Symposium on Programming, ESOP 2005*, ser. Lecture Notes in Computer Science, vol. 3444. Springer, 2005, pp. 157–171.

[18] P. Audebaud and C. Paulin-Mohring, "Proofs of randomized algorithms in Coq," *Sci. Comput. Program.*, vol. 74, no. 8, pp. 568–589, 2009.

[19] I. Damgård and B. Pfitzmann, "Sequential iteration of interactive arguments and an efficient zero-knowledge argument for NP," in *Automata, Languages and Programming, 25th International Colloquiumm, ICALP 1998*, ser. Lecture Notes in Computer Science, vol. 1443. Springer, 1998, pp. 772–783.

[20] C.-P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[21] T. Okamoto, "Provably secure and practical identification schemes and corresponding signature schemes," in *Advances in Cryptology – CRYPTO 1992*, ser. Lecture Notes in Computer Science, vol. 740. Springer, 1993, pp. 31–53.

[22] L. Guillou and J.-J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," in *Advances in Cryptology – EUROCRYPT 1988*, ser. Lecture Notes in Computer Science, vol. 330. Springer, 1988, pp. 123–128.

[23] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," in *Advances in Cryptology – CRYPTO 1986*. Springer, 1987, pp. 186–194.

[24] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *J. Cryptology*, vol. 1, no. 2, pp. 77–94, 1988.

[25] I. Damgård, "On sigma-protocols," Lecture Notes on Cryptologic Protocol Theory, 2010.

[26] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *J. Cryptology*, vol. 7, no. 1, pp. 1–32, 1994.

[27] J. A. Garay, P. MacKenzie, and K. Yang, "Strengthening zero-knowledge protocols using signatures," *J. Cryptology*, vol. 19, pp. 169–209, 2006.

[28] I. Damgård, "Efficient concurrent zero-knowledge in the auxiliary string model," in *Advances in Cryptology – EUROCRYPT 2000*, ser. Lecture Notes in Computer Science, vol. 1807. Springer, 2000, pp. 418–430.

[29] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *20th Annual ACM Symposium on Theory of computing, STOC 1988*. ACM, 1988, pp. 103–112.

[30] I. Damgård, "On the existence of bit commitment schemes and zero-knowledge proofs," in *Advances in Cryptology – CRYPTO 1989*, ser. Lecture Notes in Computer Science, vol. 435. Springer, 1990, pp. 17–27.

Appendix

A. Proofs of properties of the AND-combination

Completeness: We know that each of the sub-protocols is complete, i.e.

$$\begin{aligned} R_1(\mu(x_1), \mu(w_1)) &\implies \Pr[\mathsf{Protocol}_1, \mu : b_1 = \mathsf{true}] = 1 \\ R_2(\mu(x_2), \mu(w_2)) &\implies \Pr[\mathsf{Protocol}_2, \mu : b_2 = \mathsf{true}] = 1 \end{aligned}$$

and our goal is to prove that their AND-combination is complete,

$$R(\mu(x), \mu(w)) \implies \Pr[\mathsf{Protocol}, \mu : b = \mathsf{true}] = 1$$

We begin by inlining the definition on V_2 in the protocol, thus obtaining

$$\boxed{\begin{array}{l}(r, state) \leftarrow \mathsf{P}_1(x, w);\\ c \leftarrow \mathsf{V}_1(x, r);\\ s \leftarrow \mathsf{P}_2(x, w, state, c);\\ b_1 \leftarrow \mathsf{V}_2^1(x_1, r_1, c, s_1);\\ b_2 \leftarrow \mathsf{V}_2^2(x_2, r_2, c, s_2);\\ b \leftarrow b_1 = \mathsf{true} \wedge b_2 = \mathsf{true}\end{array}}$$

We then split the goal into two subgoals by applying the following lemma:

$$\frac{\Pr[p, \mu : b_1] = 1 \quad \Pr[p, \mu : b_2] = 1}{\Pr[p; b \leftarrow b_1 \wedge b_2, \mu : b] = 1}$$

We detail the proof of the first subgoal, the second is proven analogously. The subgoal has the form

$$\Pr[p, \mu : b_1 = \mathsf{true}] = 1$$

where p is the following program

$$\boxed{\begin{array}{l}(r, state) \leftarrow \mathsf{P}_1(x, w);\\ c \leftarrow \mathsf{V}_1(x, r);\\ s \leftarrow \mathsf{P}_2(x, w, state, c);\\ b_1 \leftarrow \mathsf{V}_2^1(x_1, r_1, c, s_1);\\ b_2 \leftarrow \mathsf{V}_2^2(x_2, r_2, c, s_2)\end{array}}$$

We restate the subgoal as an observational equivalence statement

$$p \simeq_{\{b\}}^{R(x,w)} b \leftarrow \mathsf{true}$$

We then simplify the program in the left hand side of the equivalence by applying the following sequence of tactics:

```
inline P1; inline P2; inline V1;
alloc_l c c1; ep; deadcode
```

$$\boxed{\begin{array}{l}(r_1, state_1) \leftarrow \mathsf{P}_1^1(x_1, w_1);\\ c_1 \xleftarrow{\$} \{0,1\}^k;\\ s_1 \leftarrow \mathsf{P}_2^1(x_1, w_1, state_1, c_1);\\ b_1 \leftarrow \mathsf{V}_2^1(x_1, r_1, c_1, s_1)\end{array}}$$

Observe that since protocol (P^1, V^1) is a public-coin protocol, the above program is semantically equivalent to the protocol. Since we know that $R(x, w)$ holds as precondition, we know that $R_1(x_1, w_1)$ also holds, and we can conclude the proof by appealing to the completeness of protocol (P^1, V^1).

Special Honest-Verifier Zero-Knowledge: We should prove the following observational equivalence statement:

$$\boxed{\begin{array}{l}\mathsf{Protocol}(x, w, c):\\ \ (r, state) \leftarrow \mathsf{P}_1(x, w);\\ \ s \leftarrow \mathsf{P}_2(x, w, state, c);\\ \ b \leftarrow \mathsf{V}_2(x, r, c, s)\end{array}} \simeq_{\{r,c,s\}}^{\{x,c\} \wedge R(x,w)} \mathsf{S}(x, c)$$

We introduce as an intermediate game the sequential composition of each of the sub-protocols, i.e.

$$p \stackrel{\text{def}}{=} \begin{array}{l}(x_1, x_2) \leftarrow x;\ (w_1, w_2) \leftarrow w;\\ \mathsf{Protocol}_1(x_1, w_1, c);\\ \mathsf{Protocol}_2(x_2, w_2, c);\\ r \leftarrow (r_1, r_2);\ s \leftarrow (s_1, s_2)\end{array}$$

We prove that the above game is observational equivalent to the main protocol using the following sequence of tactics:

```
inline_l P1; inline_l P2;
sinline_l V2;
swap; eqobs_in
```

We have then to prove that

$$p \simeq_{\{r,c,s\}}^{\{x,c\} \wedge R(x,w)} \mathsf{S}(x, c)$$

After inlining the definition of S and simplifying this becomes

$$\begin{array}{l}\mathsf{Protocol}_1(x_1, w_1, c);\\ \mathsf{Protocol}_2(x_2, w_2, c)\end{array} \simeq_{\{r_1,r_2,s_1,s_2\}}^{\Phi} \begin{array}{l}(r_1, s_1) \leftarrow \mathsf{S}_1(x_1, c);\\ (r_2, s_2) \leftarrow \mathsf{S}_2(x_2, c)\end{array}$$

where $\Phi = \{x_1, x_2, w_1, w_2, c\} \wedge R_1(x_1, w_1) \wedge R_2(x_2, w_2)$. This is proved in two steps by applying the rule of sequential composition of the Relational Hoare Logic,

$$\frac{p_1 \simeq_{\Theta}^{\Phi} p_1' \quad p_2 \simeq_{\Psi}^{\Theta} p_2'}{(p_1; p_2) \simeq_{\Psi}^{\Phi} (p_1'; p_2')}$$

with $\Theta = \{r_1, s_1, x_2, w_2, c\} \wedge R_2(x_2, w_2)$, which leads to two proof obligations. The first one has the form:

$$\mathsf{Protocol}_1(x_1, w_1, c) \simeq_{\Theta}^{\Phi} (r_1, s_1) \leftarrow \mathsf{S}_1(x_1, c)$$

and follows directly from the sHVZK property of protocol (P^1, V^1) and the fact that neither of the programs in the equivalence modify x_2, w_2 or c.

The second proof obligation is the equivalence

$$\mathsf{Protocol}_2(x_2, w_2, c) \simeq_{\{r_1,r_2,s_1,s_2\}}^{\Theta} (r_2, s_2) \leftarrow \mathsf{S}_2(x_2, c)$$

As above neither program modifies r_1 or s_1, and thus the equivalence follows from the sHVZK property of the second sub-protocol.

Soundness: We must prove that given two accepting runs

$$((r_1, r_2), c, (s_1, s_2)) \quad ((r_1, r_2), c', (s_1', s_2'))$$

of the main protocol with $c \neq c'$, the knowledge extractor KE proposed in Section VI-A succeeds in computing a witness for each of the sub-protocols. We begin by observing that each accepting run of the main protocol yields and accepting run for each of the sub-protocols. We illustrate the proof of this remark by showing how to extract an accepting run for protocol (P^1, V^1) from the first accepting conversation, the other cases are analogous.

Remark 14. (r_1, c, s_1) is an accepting conversation of protocol (P^1, V^1), i.e.

$$\Pr\left[b_1 \leftarrow \mathsf{V}_2^1(x_1, r_1, c, s_1), \mu : b_1 = \mathsf{true}\right] = 1$$

Proof: Because $((r_1, r_2), c, (s_1, s_2))$ is accepting,

$$\Pr\left[b \leftarrow \mathsf{V}_2((x_1, x_2), (r_1, r_2), c, (s_1, s_2)), \mu : b = \mathsf{true}\right] = 1$$

By expanding the definition of V_2, one gets

$$\Pr\left[\begin{matrix} b_1 \leftarrow \mathsf{V}_2^1(x_1, r_1, c', s_1'); \\ b_2 \leftarrow \mathsf{V}_2^2(x_2, r_2, c', s_2') \end{matrix}, \mu : b_1 = \mathsf{true} \wedge b_2 = \mathsf{true}\right] = 1$$

Observe then that for any program p,

$$\Pr\left[p, \mu : b_1 \wedge b_2\right] \leq \Pr\left[p, \mu : b_1\right]$$

which implies that

$$\Pr\left[\begin{matrix} b_1 \leftarrow \mathsf{V}_2^1(x_1, r_1, c', s_1'); \\ b_2 \leftarrow \mathsf{V}_2^2(x_2, r_2, c', s_2') \end{matrix}, \mu : b_1 = \mathsf{true}\right] = 1$$

The call to V_2^2 in the above program is dead code and can be eliminated, thus obtaining the needed result. ■

We now have to prove that

$$\Pr\left[(w_1, w_2) \leftarrow \mathsf{KE}(\ldots), \mu : R_1(x_1, w_1) \wedge R_2(x_2, w_2)\right] = 1$$

Observe that for any program p and events E_1, E_2,

$$\frac{\Pr\left[p, \mu : E_1\right] = 1 \quad \Pr\left[p, \mu : E_2\right] = 1}{\Pr\left[p, \mu : E_1 \wedge E_2\right] = 1}$$

so that after inlining the definition of KE and eliminating dead code, our goal boils down to proving

$$\begin{array}{l} \Pr\left[w \leftarrow \mathsf{KE}^1(x_1, r_1, c, c', s_1, s_1'), \mu : R_1(x_1, w_1)\right] = 1 \ \wedge \\ \Pr\left[w \leftarrow \mathsf{KE}^2(x_2, r_2, c, c', s_2, s_2'), \mu : R_2(x_2, w_2)\right] = 1 \end{array}$$

Both proof obligations follow directly from the soundness of the corresponding sub-protocol and the above remark that conversations (r_i, c, s_i) and (r_i, c', s_i') are accepting.