Implementing Security Protocol Monitors

Yannick Chevalier, Michaël Rusinowitch

Implementing Security Protocol Monitors

Yannick Chevalier
Irit, Université Paul Sabatier
Toulouse
France
ychevali@irit.fr

Michaël Rusinowitch
Lorraine University, Cnrs, Inria
Nancy
France
michael.rusinowitch@loria.fr

Cryptographic protocols are often specified by narrations, *i.e.*, finite sequences of message exchanges that show the intended execution of the protocol. Another use of narrations is to describe attacks. We propose in this paper to compile, when possible, attack describing narrations into a set of tests that honest participants can perform to exclude these executions. These tests can be implemented in monitors to protect existing implementations from rogue behaviour.

1 Introduction

Cryptographic protocols are designed to prescribe message exchanges between agents in a hostile environment in order to guarantee some security properties. In particular security properties such as confidentiality or authentication are violated when there exists an execution of the protocol in which they do not hold. However it has often been found that under certain circumstances, and after its deployment, a protocol failed to adequatly protect its participants. These circumstances usually involve one or more sessions, and the participation of a dishonest agent hereafter called the intruder. When the attack is on a specific implementation of a protocol, its mitigation usually amounts to fixing this implementation.

However, some attacks are related to the exchanges of messages prescribed by the protocol, and not in the actual handling of these messages by participants. In that case, the only recourse is—when available—to alter the sequence of acceptable messages. This can be implemented by changing the format of the messages exchanged or by stopping an execution once it has been detected that the attack may be under way. We consider in this paper only the second approach, in which the participants behaviour is altered in order to reject some possible executions of the protocol.

Let us consider for example the Needham-Schroder Public Key (*NSPK*) mutual authentication protocol [11] described by the following sequence of messages between roles A and B:

$$\begin{array}{rl} A \textbf{ knows} & A, B, K_A, K_B, K_A^{-1} \\ B \textbf{ knows} & A, B, K_A, K_B, K_B^{-1} \\ A \to B & : \mathrm{enc}(A, N_A, K_B) \\ B \to A & : \mathrm{enc}(N_A, N_B, K_A) \\ A \to B & : \mathrm{enc}(N_B, K_B) \end{array}$$

The attack on this protocol discovered by Lowe [10] is described as follows, where $I(A)$ denotes the

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
EPTCS 342, 2021, pp. 22–34, doi:10.4204/EPTCS.342.3

intruder impersonating the agent A:

$$\begin{aligned} A \to I &: \mathrm{enc}(A, N_A, K_I) \\ I(A) \to B &: \mathrm{enc}(A, N_A, K_B) \\ B \to I(A) &: \mathrm{enc}(N_A, N_B, K_A) \\ I \to A &: \mathrm{enc}(N_A, N_B, K_A) \\ A \to I &: \mathrm{enc}(N_B, K_I) \\ I(A) \to B &: \mathrm{enc}(N_B, K_B) \end{aligned}$$

This execution is an attack because B believes he has participated in a session with A whereas A never exchanged a message with B. As is the case in this attack narration, we assume from now on and without loss of generality [8] that in an attack, every message is sent to or received from the intruder. A fix, proposed in [10], consisting in altering the second message to include the name of the sender. The only drawback to such fixes is that implementations of the amended protocol are not interoperable with implementations of the original protocol. For widely deployed real-life protocols, interoperability must be maintained and thus the amended version coexists with the original one for years, leaving open an attack vector for attackers.

Our proposal aims at keeping the original version, but extended with additional tests. This extension involves the creation of a monitor for the actions of honest participants that furthermore may have access to some secret pieces of information held by these participants. In the case of Lowe's attack, this access is unnecessary, as it suffices for B to check that the message he receives at the third step is equal to the message sent by A.

We present in this paper an algorithm to implement a security protocol monitor. Given the input messages that participants are willing to share with the monitor, it basically amounts to computing the conditions to be checked in order to exclude a given narration from the possible executions of the protocol.

Related works This article is based on the refinement relation between traces introduced in [4]. An extension to the case where an attack can be excluded based on the information in only one session of a participant has been proposed in [9].

By contrast our approach stems from the line of work initiated in [3, 2] where the authors advocate for the prevention of attacks through detection and eventually retaliation against the attacker. Also, [7] presents in more details an architecture in which the analysis we present in this paper can be conducted with a better control on the messages, and also introduce the idea of applicative firewalls for security protocols.

Outline We recall in Sec. 2 how to represent protocols and roles and how to implement them as active frames. In Sec. 3 we formally introduce protocol monitors to control messages and manage knowledge shared by collaborating agents, in order to detect and block attacks. In Sec. 4 we show how to synthesize monitors from tests that can be derived automatically. We conclude in Sec. 6.

2 Role-based Protocol Specifications

2.1 Messages and basic operations

We consider an infinite set of free constants $\mathscr{C}$ and an infinite set of variables $\mathscr{X}$. For each signature $\mathscr{F}$ (*i.e.* a set of function symbols with arities), we denote by $\mathrm{T}(\mathscr{F})$ (resp. $\mathrm{T}(\mathscr{F}, \mathscr{X})$) the set of terms

over $\mathscr{F} \cup \mathscr{C}$ (resp. $\mathscr{F} \cup \mathscr{C} \cup \mathscr{X}$). The former is called the set of ground terms over $\mathscr{F}$, while the latter is simply called the set of terms over $\mathscr{F}$. Variables are denoted by x and decorations thereof, but for a distinguished subset $(v_i)_{i\in\mathbb{N}}$ employed to denote positions in a sequence. Terms are denoted by s, t, and finite sets of terms are written $E, F, \ldots$, and decorations thereof, respectively. In a signature $\mathscr{F}$ a *constant* is either a *free constant* or a function symbol of arity 0 in $\mathscr{F}$. Given a term t we denote by $\mathrm{Var}(t)$ the set of variables occurring in t and $\mathrm{Cons}(t)$ the set of free constants occurring in t. A (ground) substitution σ is an idempotent mapping from $\mathscr{X}$ to $\mathrm{T}(\mathscr{F},\mathscr{X})$ ($\mathrm{T}(\mathscr{F})$) and its support $\mathrm{Supp}(\sigma) = \{x \,|\, \sigma(x) \neq x\}$ is a finite set. The application of a substitution σ on a term t (resp. a set of terms E) is denoted $t\sigma$ (resp. $E\sigma$) and is equal to the term t (resp. the set of terms E) where all variables $x \in \mathrm{Supp}(\sigma)$ have been replaced by the term $x\sigma$.

Terms are manipulated by applying *operations* on them. These operations are defined by a subset of the signature $\mathscr{F}$ called the *set of public constructors*. A context $C[v_1,\ldots,v_n]$ is a term in which $\mathrm{Var}(C[v_1,\ldots,v_n]) \subseteq \{v_1,\ldots,v_n\}$, $\mathrm{Const}(C[v_1,\ldots,v_n]) = \emptyset$, and all non-variable symbols are public constructors, including possibly non-free constant. We will specify the effects of operations on the messages and the properties of messages by equations. When the index n is clear, we omit the possible variables list and denote contexts C. An *equational presentation* $\mathscr{E} = (\mathscr{F}, E)$ is defined by a set E of equations $u = v$ with $u, v \in \mathrm{T}(\mathscr{F},\mathscr{X})$. The *equational theory* generated by $(\mathscr{F}, E)$ on $\mathrm{T}(\mathscr{F},\mathscr{X})$ is the smallest congruence containing all instances of axioms in E (free constants can also be used for building instances) [6]. We write $s =_{\mathscr{E}} t$ as the congruence relation between two terms s and t. By abuse of terminology we also call $\mathscr{E}$ the equational theory generated by the presentation $\mathscr{E}$ when there is no ambiguity.

A *deduction system* is defined by a triple $(\mathscr{E}, \mathscr{F}, \mathscr{F}_p)$ where $\mathscr{E}$ is an equational presentation on a signature $\mathscr{F}$ and $\mathscr{F}_p$ a subset of *public* constructors in $\mathscr{F}$.

Example 1. Public key cryptography. For instance the following deduction system models public key cryptography:

$$\begin{array}{l}(\{\mathrm{dec}(\mathrm{enc}(x,y),y^{-1}) = x\},\\ \{\mathrm{dec}(\_,\_),\mathrm{enc}(\_,\_),\_^{-1}\},\\ \{\mathrm{dec}(\_,\_),\mathrm{enc}(\_,\_)\})\end{array}$$

The equational theory is reduced here to a single equation that expresses that one can decrypt a ciphertext when the inverse key is available. The inverse function $\_^{-1}$ is not public, as it cannot be computed in reasonable time by participants.

Example 2. Nonce generation. Nonces are random values that are critical to the analysis of cryptography and cryptographic protocols. To give an agent the capacity to generate new values, we assume the existence of an infinite set of constants $\mathscr{C}_{\mathscr{N}}$ away from $\mathscr{C}$ such that each value in this set can be generated:

$$\mathscr{N} = (\emptyset, \mathscr{C}_{\mathscr{N}}, \mathscr{C}_{\mathscr{N}})$$

Note this model makes sense only in the case where the attacker is only one agent, or a set of information sharing agents [13], as an agent cannot otherwise construct nonces generated by another, independent, agent.

Test systems. In order to express verifications performed by an agent on received messages we introduce test systems:

Definition 1. (Test systems) Let $\mathscr{D}$ be a deduction system with an equational theory $\mathscr{E}$. A *$\mathscr{D}$-test system* $S[v_1,\ldots,v_n]$ is a finite set of equations denoted by $(C_i \stackrel{?}{=} C'_i)_{i\in\{1,\ldots,n\}}$ with $\mathscr{D}$-contexts $C_i[v_1,\ldots,v_n], C'_i[v_1,\ldots,v_n]$. It is satisfied by a substitution σ, and we denote by $\sigma \models S[v_1,\ldots,v_n]$, if for all $i \in \{1,\ldots,n\}$ the equality $C[v_1,\ldots,v_n]_i\sigma =_{\mathscr{E}} C'_i[v_1,\ldots,v_n]\sigma$ holds.

As usual we simply denote a test system S if the maximal indice n is clear from the context.

2.2 Traces and active frames

We model messages with terms. The sequence of messages received and sent by a principal is a *trace*, and is thus a finite sequence of labeled messages:

Definition 2. (Trace) A *trace* is a finite sequence of messages each with label (or polarity) ! or ?.

Messages with label ! (resp. ?) are said to be "sent" (resp."received"). Given a trace $\Lambda = !/?t_1, \ldots, !/?t_n$ we write $?\Lambda$ (resp. $!\Lambda$) as a short-hand for $?m_1, \ldots, ?m_n$, (resp. $!m_1, \ldots, !m_n$). Given a trace $\Lambda = !/?m_1, \ldots, !/?m_n$ we denote by $\sigma_\Lambda = \{v_1 \mapsto m_1, v_n \mapsto m_n\}$ the substitution mapping each variable v_i to the ith message occurring in Λ. To simplify notation we also denote by $C[v_1, \ldots, v_n] \cdot \Lambda$, or more simply $C \cdot \Lambda$, the application of the substitution σ_Λ on the context $C[v_1, \ldots, v_n]$. Accordingly, we say that a trace Λ satisfies an equality $C_1 = C_2$, and denote it by $\Lambda \models C_1 \stackrel{?}{=} C_2$, whenever $C_1 \cdot \Lambda =_{\mathscr{E}} C_2 \cdot \Lambda$.

Operations on traces Let Λ be a trace. We say that a Λ is positive (*resp.* negative) if all its labels are ! (*resp.* ?). We denote $\Lambda^?$ (*resp.* $\Lambda^!$) the subsequence of Λ of terms labeled with ? (*resp.* !). We denote $-\Lambda$ the trace in which all the labels in Λ are inverted. Finally, we denote input(Λ) (resp. output(Λ)) the trace $-\Lambda^?$ (resp. $-\Lambda^!$).

Active frames An *active frame* represents the actions of a principal participating in a protocol. It is a sequence of steps, and at step i the principal either sends a message, constructed from the messages received at steps $j < i$, or receives and message, and accepts it if it satisfies some tests constructed from it and messages received at steps $j < i$. To simplify exposition, at a step i, we call these messages received at steps $j < i$ the messages *already known* at step i, or just *already known* if the step is clear from context. As in the case of traces, messages sent are labeled $!v_i$ (and v_i is an output variable) and those received are labeled $?v_i$ (and v_i is an input variable). Since the available contexts depend upon the deduction system, the notion of active frame is also parameterised by a deduction system.

Definition 3. Given a deduction system $\mathscr{D}$, a *$\mathscr{D}$-active frame* is a sequence $(T_i)_{1 \leq i \leq k}$ where

$$T_i = \begin{cases} !v_i \text{ with } v_i \stackrel{?}{=} C_i[v_1, \ldots, v_{i-1}] & \text{(send)} \\ \textbf{or} & \\ ?v_i \text{ with } S_i[v_1, \ldots, v_i] & \text{(receive)} \end{cases}$$

Without loss of generality and reusing the above notations, a simple recursion shows that we can assume that all variables in $C_i[v_1, \ldots, v_{i-1}]$ are labeled with ? at a step $j < i$, and that all variables in $S_i[v_1, \ldots, v_i]$ are labeled with ? at a step $j \leq i$. Without loss of generality, from now on we assume that this is the case for all the active frames we consider.

Example 3. The principal A in the description of the NSPK protocol can be modeled by an active frame as follows, with the caveat that we have renamed the v_i variables for more clarity:

$$\begin{array}{l} (?x_{N_A} \text{ with } \emptyset, ?x_A \text{ with } \emptyset, ?x_B \text{ with } \emptyset, ?x_{K_A} \text{ with } \emptyset, ?x_{K_B} \text{ with } \emptyset, ?x_{K_A^{-1}} \text{ with } \emptyset, \\ !x_{msg_1} \text{ with } x_{msg_1} \stackrel{?}{=} \mathrm{enc}(\langle x_A, x_{N_A} \rangle, x_{K_B}), \\ ?x_r \text{ with } \emptyset \\ !x_{msg_2} \text{ with } x_{msg_2} \stackrel{?}{=} \mathrm{enc}(\pi_2(\mathrm{dec}(x_r, x_{K_A^{-1}})), x_{K_B})) \end{array}$$

Algebraically, by describing a principal's actions, active frames are partial operations on the set of traces and map a sequence of messages sent by someone else and accepted to the sequence of received and sent messages by a principal. We formalize these notions as follows:

Definition 4. Let $\mathscr{D}$ be a deduction system with equational theory $\mathscr{E}$. Let $\varphi = (T_i)_{1\leq i\leq n}$ be an active frame, where the T_i's are as in Definition 3, and where the input variables are $?v_{\alpha_1},\ldots,?v_{\alpha_k}$. Let Λ be a positive trace of length k, θ be the renaming of variables $\{v_{\alpha_j}\mapsto v_j\}_{1\leq j\leq k}$, and S be the union of the test systems in φ. The *evaluation* of φ on Λ is denoted $\varphi\cdot\Lambda$. It is defined, and we say that φ *accepts* s, if $S\cdot s$ is satisfiable. In that case, it is the trace $(m_1,\ldots,m_n)$ where:

$$m_i = \begin{cases} !C_i\cdot\theta\sigma_\Lambda & \text{If } v_i \text{ has label ! in } T_i \\ ?v_i\cdot\theta\sigma_\Lambda & \text{If } v_i \text{ has label ? in } T_i \end{cases}$$

Example 4. Let Λ_A be the trace of the principal A in the the specification of the NSPK protocol in the introduction, $r=\mathrm{tr}(A)$, and ϕ_A be the active frame of Ex. 3. Let M be the message $\mathrm{msg}(B,\mathrm{enc}(\langle N_A,N_b\rangle,K_A))$. We have:

$$\mathrm{input}(\Lambda_A) = (!N_A,!A,!B,!K_A,!K_B,!K_A^{-1},!M)$$

and $\phi_A\cdot\mathrm{input}(\Lambda_A)$ is the trace:

$$\begin{array}{l}(?N_A,?A,?B,?K_A,?K_B,?K_A^{-1},\\ !\mathrm{msg}(B,\mathrm{enc}(\langle A,N_A\rangle,K_B)),\\ ?M,!\mathrm{msg}(B,\mathrm{enc}(\pi_2(\mathrm{dec}(\mathrm{payload}(M),K_A^{-1})),K_B))\end{array}$$

Modulo the equational theory, this trace is equal to:

$$\begin{array}{l}(?N_A,?A,?B,?K_A,?K_B,?K_A^{-1},\\ !\mathrm{msg}(B,\mathrm{enc}(\langle A,N_A\rangle,K_B)),?M,!\mathrm{msg}(B,\mathrm{enc}(N_b,K_B))\end{array}$$

It is not coincidental that in Ex. 4 the traces $\varphi_A\cdot\mathrm{input}(\Lambda_A)$ and Λ_A are equal as it means that within the active frame, the sent messages are composed from received ones in such a way that when someone sends the messages expected in Λ_A, the execution of A is described by Λ_A. This relation gives us a criterion to define what an implementation of a trace is.

Definition 5. An active frame φ is an *implementation* of a trace Λ if φ accepts $\mathrm{input}(\Lambda)$ and $\varphi\cdot\mathrm{input}(\Lambda)=_{\mathscr{E}}\Lambda$.

If a trace admits an implementation we say this trace is *executable*. Conversely we say that a trace t is an *execution* of an active frame φ whenever φ is an implementation of t.

2.3 Computation of an implementation

We present in this section a method, parameterised by the deduction system $\mathscr{D}$, to compute an active frame implementing an executable trace. To build such an implementation we need to compute, given a message t sent at step i, a $\mathscr{D}$-context C_i that evaluates to t when applied to the previously received messages. This reachability problem is unsolvable in general. Hence we have to consider systems that admit a reachability algorithm, formally defined below:

Definition 6. Given a deduction system $\mathscr{D}$ with equational theory $\mathscr{E}$, a *$\mathscr{D}$-reachability algorithm* $\mathscr{A}_\mathscr{D}$ computes, given a positive trace Λ of length n and a term t, a $\mathscr{D}$-context $\mathscr{A}_\mathscr{D}(s,t)=C[v_1,\ldots,v_n]$ such that $C\cdot\Lambda=_{\mathscr{E}}t$ iff there exists such a context and $\bot$ otherwise.

For the many theories that admit a reachability algorithm, it can be employed as an oracle to compute the contexts in sent messages and therefore to derive an implementation of a trace s. We thus have the following theorem (see a proof in [4]).

Theorem 1. *If a $\mathscr{D}$-reachability algorithm exists then it can be decided whether a trace s is executable and if so one can compute an implementation of s.*

2.4 Computation of a prudent implementation

An implementation does not necessarily checks the conformity of the messages with the intended patterns, *e.g.*, the active frame in Ex. 4 neither checks that x_r is really an encryption with the public key x_{K_A} of a pair, nor that the first argument of the encrypted pair has the same value as the nonce x_{N_A}.

Any of the algorithms proposed so far in the literature for the compilation of cryptographic protocols would require at least these tests. We now present an algorithm that computes these kinds of checks for an arbitrary deduction system. It formalizes a check as an equation between $\mathscr{D}$-contexts over messages received so far, including the initial knowledge. For example, and reusing the notations of Ex. 3 it computes that upon reception of the message the initiator must, among other tests, check the validity of the equation:

$$\pi_1(\mathrm{dec}(x_r, x_{K_A^{-1}})) \stackrel{?}{=} x_{N_A}$$

We formalize in the definition below which traces Λ' are acceptable by an agent expecting a trace Λ. We define the acceptable traces as the refinements of Λ, that is traces Λ' such that every test system accepting Λ also accepts Λ'.

Definition 7. Let Λ, Λ' be two positive traces of identical length. We say that Λ' *refines* Λ if, for any pair of $\mathscr{D}$-contexts (C_1, C_2) one has $C_1 \cdot \Lambda = C_2 \cdot \Lambda$ implies $C_1 \cdot \Lambda' = C_2 \cdot \Lambda'$.

Consider for example the following traces Λ and Λ':

$$\begin{cases} \Lambda' = (!\mathrm{enc}(a,k), !\mathrm{enc}(a,k'), !k, !\mathbf{k}'', !a) \\ \Lambda = (!\mathrm{enc}(a,k), !\mathrm{enc}(a,k'), !k, !\mathbf{k}', !a) \end{cases}$$

since all equalities that can be checked on σ can be checked on σ'. Two traces s, s' that refine one another are *equivalent*. This definition is an adaptation to our setting of the classic notion of static equivalence [1].

When the behaviour of a principal is defined by a trace Λ, we expect that any implementation of that principal accepts the trace input(Λ). Thus, and as long as only equality tests are considered, we expect any implementation of the trace Λ to also accept any refinement of input(Λ). We define a *prudent implementation* of Λ as an implementation that only accepts inputs that refine the inputs in Λ.

Definition 8. An active frame φ is a *prudent implementation* of a trace Λ if φ is an implementation of Λ and any trace Λ' accepted by φ is a refinement of input(Λ).

As already noted in [4], most deduction systems considered in the context of cryptographic protocols analysis have the property that it is possible to compute, given a positive trace, a finite set of context pairs that summarizes all possible equalities. Given a positive trace Λ we denote P_Λ the (infinite) set of context pairs (C_1, C_2) such that $C_1 \cdot s = C_2 \cdot s$.

Definition 9. A deduction system $\mathscr{D}$ has the *finite basis property* if for each positive trace Λ one can compute a finite set P_Λ^f of pairs of $\mathscr{D}$-contexts such that, for each positive trace Λ':

$$P_\Lambda \subseteq P_{\Lambda'} \text{ iff } P_\Lambda^f \subseteq P_{s'}$$

Let us now assume that a deduction system $\mathscr{D}$ has the finite basis property. There thus exists an algorithm $\mathscr{A}'_{\mathscr{D}}$ that takes a positive trace Λ as input, computes a finite set P_Λ^f of context pairs (C, C') and returns as a result the test system $S_\Lambda : \left\{ C \stackrel{?}{=} C' \mid (C, C') \in P_\Lambda^f \right\}$. For any positive trace Λ' of length n, by definition of S_Λ we have that $S_\Lambda \cdot \Lambda'$ is satisfiable if and only if s' is a refinement of s. We are now ready to present our algorithm for the compilation of strands into active frames.

Algorithm Given a trace Λ and assuming that the deduction system $\mathscr{D}$ has a reachability algorithm and the finite basis property, and let Λ be a trace of length n, and let us denote Λ^i for $1 \leq i \leq n$ the prefix of length i of Λ, and $\Lambda(i)$ the ith element of Λ. We construct a prudent implementation $\varphi_\Lambda = (T_i)_{i=1,\ldots,n}$ of Λ as follows:

$$T_i = \begin{cases} !v_i \text{ with } v_i \stackrel{?}{=} \mathscr{A}_{\mathscr{D}}(\Lambda^{i-1}, t_i) & \text{If } \Lambda(i) = !t_i \\ ?v_i \text{ with } \mathscr{A}'_{\mathscr{D}}(\Lambda^i) & \text{If } \Lambda(i) = ?t_i \end{cases}$$

By construction we have the following theorem[4]:

Theorem 2. *Let $\mathscr{D}$ be a deduction system that has a $\mathscr{D}$-ground reachability algorithm and has the finite basis property. Then for any executable trace Λ one can compute a prudent implementation φ_Λ of Λ.*

2.5 Protocol implementation and execution

It is customary to describe a protocol by giving its intended execution, either using a message sequence chart or an Alice&Bob notation. We note that the same notation is also employed to described *e.g.* attacks on that protocol. Beyond their syntax, the characteristic of such description is to associate to a generic principal (a rôle, in the case of a protocol specification, a participant in the case of an attack description) a trace describing its actions, and how these actions interact with the other principal actions. This association of a participant with a trace is formalised by a function mapping *strands* [14], *i.e.* principals, rôles, etc., to traces. We define a protocol to be just one such mapping.

Definition 10. (Protocol) A *protocol* is a couple $P = (\Xi_P, \mathrm{tr}_P)$ where Ξ_P is a finite set of strands and tr_P maps Ξ_P to the set of traces.

When a protocol is intended to be a protocol specification, we refer to strands as the rôles of that protocol (*e.g.* the rôles A and B in the NSPK protocol. A strand ξ is *positive* in a protocol P if $\mathrm{tr}_P(\xi)$ is a positive trace.

In the preceding definition the function tr_P prescribes for each role $\xi \in \Xi_P$ the sequence of actions to be performed by an agent playing this role in any protocol instance. In the following, when there is no ambiguity in the considered protocol, we identify a strand ξ with its trace $\mathrm{tr}(\xi)$.

We have worked so far on the implementation of the trace of a role in a protocol, but the definitions lift to the level of an implementation of a protocol as follows.

Definition 11. (Protocol implementation) An *implementation of a protocol* $P = (\Xi_P, \mathrm{tr}_P)$ is a couple (Ξ_P, Φ_P) where Φ maps each role $\xi \in \Xi_P$ to an active frame such that $\Phi(\xi)$ is an implementation of $\mathrm{tr}_P(\xi)$. It is *prudent* if moreover for each $\xi \in \Xi_P$, $\Phi(\xi)$ is a prudent implementation of $\mathrm{tr}_P(\xi)$.

From now on we consider only prudent implementations of protocols, *i.e.* implementations whose execution is a refinement of the protocol specification.

Definition 12. (Protocol execution) Let $P = (\Xi_P, \Phi_P)$ be a protocol implementation. A triple $E = (\Xi_E, \mathrm{tr}_E, R_E)$ where:

1. Ξ_E is a set of strands away from Ξ_P;
2. (Ξ_E, tr_E) is a protocol;
3. $R_E : \Xi_E \to \Xi_P \cup \{I\}$.

is a *protocol execution* of P if, for each $\xi \in \Xi_E$, if $R_E(\xi) \neq I$ then $\mathrm{tr}_E(\xi)$ is an execution of $\Phi_P(R_E(\xi))$.

The strand I denotes an *Intruder* who does not necessarily follows the directions prescribed by the protocol. A protocol execution is *honest* if $R_E(\Xi_E) \subseteq \Xi_P$. Strands in Ξ_E are called the *participants* of the protocol execution E. The function R_E maps each (honest) participant to its rôle in the protocol.

3 Protocol monitor

To mitigate an attack on a protocol, a monitor has to coordinate the participants to detect and stop an instance of a known flaw. This coordination is built according to data the participants are willing to share to prevent the attack. Our monitor construction relies on the description of the data the participants are willing to share, a description of the attack, and a description of the expected behaviour of the participants, and we compute tests (when possible) to distinguish an instance of the attack from the normal execution.

In Def. 13, for each participant A, $\mathrm{tr}_M(A)$ contains the same inputs as $\mathrm{tr}_P(A)$, and the messages sent in $\mathrm{tr}_M(A)$, are the pieces of data shared by A with the monitor.

Definition 13. (Protocol Monitor) Let $P = (\Xi_P, \mathrm{tr}_P)$ and $M = (\Xi_M, \mathrm{tr}_M)$ be two protocols. We say that M *is a monitor for* P if 1. $\Xi_M = \Xi_P$; 2. M is executable; and 3. For each $\xi \in \Xi_M$ we have $\mathrm{input}(\mathrm{tr}_M(\xi)) = \mathrm{input}(\mathrm{tr}_P(\xi))$.

Proposition 1. *Let $P = (\Xi_P, \mathrm{tr}_P)$ be a protocol, $M = (\Xi_P, \mathrm{tr}_M)$ be a monitor of P, $I_X = (\Xi_P, \Phi_X)$ be any implementation of $X \in \{P, M\}$, and $E = (\Xi_E, \Phi_E, R_E)$ be an honest execution of I_P.*

If I_P is prudent and $\Phi_M(R_E(\xi))$ accepts $\mathrm{input}(\mathrm{tr}_E(\xi))$, then $\Phi_M(R_E(\xi)) \cdot \mathrm{input}(\mathrm{tr}_E(\xi))$ is a refinement of $\mathrm{tr}_M(R_E(\xi))$.

Proof. Assume there exists $\xi_R \in \Xi_M$ and $\xi_e \in \Xi_E$ with $R_E(\xi_e) = \xi_R$ such that $\Phi_M(\xi_R) \cdot \mathrm{input}(\mathrm{tr}_E(\xi_e))$ is not a refinement of $\mathrm{tr}_M(\xi_R)$. That is, there exists pairs of contexts C_1, C_2 such that $\mathrm{tr}_M(\xi_r) \models C_1 = C_2$ but $\Phi_M(\xi_R) \cdot \mathrm{input}(\mathrm{tr}_E(\xi_e)) \not\models C_1 = C_2$. Without loss of generality we can assume that C_1, C_2 are built upon the input variables of $\Phi_M(R_E(\xi))$, that is, with $\theta : \left\{v_{i_j} \mapsto v_j\right\}_{1 \le j \le k}$, where i_j is the jth input step of $\mathrm{tr}_M(\xi_R)$:

$$\begin{cases} \mathrm{input}(\mathrm{tr}_M(\xi_R)) \models C_1\theta = C_2\theta \\ \mathrm{input}(\Phi_M(\xi_R) \cdot \mathrm{input}(\mathrm{tr}_E(\xi_e))) \not\models C_1\theta = C_2\theta \end{cases}$$

Since I_M is an implementation of M, by definition the second assertion is equal to $\mathrm{input}(\mathrm{tr}_E(\xi_e)) \not\models C_1\theta = C_2\theta$. By definition of a monitor, we have $\mathrm{input}(\mathrm{tr}_M(\xi_R)) = \mathrm{input}(\mathrm{tr}_P(\xi_R))$. Thus, we have:

$$\begin{cases} \mathrm{input}(\mathrm{tr}_P(\xi_R)) \models C_1\theta = C_2\theta \\ \mathrm{input}(\mathrm{tr}_E(\xi_R)) \not\models C_1\theta = C_2\theta \end{cases}$$

Hence $\mathrm{input}(\mathrm{tr}_E(\xi_R))$ is not a refinement of $\mathrm{input}(\mathrm{tr}_P(\xi_R))$, and thus $\Phi_P(\xi_R)$ cannot be a prudent implementation of $\mathrm{tr}_P(\xi_R)$. □

Definition 14. (Execution Log) Let $P = (\Xi_P, \mathrm{tr}_P)$ be a protocol, $I_P = (\Xi_P, \Phi_P)$ be an implementation of P, $E = (\Xi_E, \mathrm{tr}_E, R_E)$ be an execution of I_P, $<_E$ be an arbitrary total order on the participants, and $I_M = (\Sigma_P, \phi_M)$ be an implementation of a monitor M of P. The *execution log* of E for monitor M is the concatenation of the traces:

$$\mathrm{output}(\phi_M(R_E(\xi_e)) \cdot \mathrm{input}(\mathrm{tr}_E(\xi_e)))$$

for $\xi_e \in \Xi_E$ such that $R_E(\xi_e) \neq I$ in the increasing order with respect to $<_E$.

Proposition 2. *Let $P = (\Xi_P, \mathrm{tr}_P)$ be a protocol, $I_P = (\Xi_P, \Phi_P)$ be an implementation of P, $E = (\Xi_E, \mathrm{tr}_E, R_E)$ be an execution of I_P, $<_E$ be an arbitrary total order on the participants, and $I_M = (\Xi_P, \Phi_M)$ be an implementation of a monitor M of P. Then there exists a unique execution log of E for M.*

Proof. For each $\xi_e \in \Sigma_E$ let $\varphi_e = \Phi_M(R_E(\xi_e)$ be the active frame executed by ξ_e, and let $\text{in}_e = \text{input}(\text{tr}_E(\xi_e))$ denote the messages received by ξ_e. Since sent messages are built by a context over preceding messages an easy recurrence shows that the value of each message in $\varphi_e \cdot \text{in}_e$ is uniquely defined by the values in $\text{input}(\text{tr}_E(\xi_e))$. Thus $\text{output}(\varphi_e \cdot \text{in}_e)$ is uniquely defined for each participant $\xi_e \in \Xi_E$. Since the order $<_E$ is total the concatenation of these traces is unique. □

Since the ordering $<_E$ is arbitrary, we usually omit any reference to it. By Prop. 2 the execution log depends only on the monitor, not on its implementation. Accordingly we denote it $\log_{I_P,M}(E)$. Assuming there exists a $\mathscr{D}$-reachability algorithm, it is possible to compute an implementation of M whenever M is executable. Thus given a monitor M the function $\log_{I_P,M}(E)$ can be effectively computed.

4 Generating an attack-preventing monitor

4.1 Attack presentation

In our setting attacks are simply specified as protocol executions without reference to any violated security property. The flexibility entailed by this choice however implies that, in order to prevent the given execution, one also has to provide what should have been the correct execution for the subset of participants involved in the attack. This setting leads to the definition of an attack presentation sharing the same set of participants playing the same roles, but having different traces.

Definition 15. (Attack definition) Let $I_P = (\Xi_P, \Phi_P)$ be a protocol implementation. An *attack definition* on I_P is a tuple $(\Xi_E, \text{tr}_A, \text{tr}_N, R_E)$ such that $(\Xi_E, \text{tr}_A, R_E)$ is an execution of I_P and $(\Xi_E \setminus R_E^{-1}(I), \text{tr}_N, R_E)$ is an honest execution of I_P.

Given an attack definition $(\Sigma_E, \text{tr}_A, \text{tr}_N, R_E)$, $(\Sigma_E, \text{tr}_A, R_E)$ refers to the *attack execution* while $(\Sigma_E, \text{tr}_N, R_E)$ refers to the *normal execution* of the protocol expected for the honest participants involved. Though this is not enforced by the definition and not needed in the rest of this paper, it is expected that the initial segments of the traces corresponding to the initial knowledge and the generation of nonces should be the same for each participant in the two executions.

Definition 16. (Attack presentation) Let $P = (\Xi_P, \text{tr}_P)$ be a protocol, $I_P = (\Xi_P, \Phi_P)$ be an implementation of P, $M = (\Xi_P, \text{tr}_M)$ be a monitor of P, and $\mathscr{A} = (\Xi_E, \text{tr}_A, \text{tr}_N, R_E)$ be an attack definition on I_P. Then the *presentation of $\mathscr{A}$ to M* is the couple $(\log_{I_P,M}((\Xi_E \setminus R_E^{-1}(I), \text{tr}_A, R_E)), \log_{I_P,M}((\Xi_E \setminus R_E^{-1}(I), \text{tr}_N, R_E)))$

Detectable attacks. We note that the two traces in an attack presentation may be equivalent. In this case, no test performed by the monitor could enable it to distinguish between the normal and the attack execution, and the latter would not be preventable. We say that an attack $\mathscr{A}$ is *detectable* by the monitor M if its presentation (Λ, Λ') to M is such that Λ and Λ' are not equivalent.

This definition leads to the problem of deciding whether an attack is detectable by a monitor.

Decision Problem 1. $\mathit{AttackDetectability}_{\mathscr{D}}(s, s')$

Input: The presentation (Λ, Λ') of an attack $\mathscr{A}$ on the protocol implementation I_P with a monitor M;

Output: YES if $\mathscr{A}$ is detectable by M

This problem is related to the classic static equivalence problem by the following theorem, proved in the appendix.

Theorem 3. *Let $\mathscr{D}$ be a deduction system, and $\mathscr{N}$ be the deduction system of Ex. 2. Then $\mathit{AttackDetectability}_{\mathscr{D}\cup\mathscr{N}}$ on strands that do not contain symbols of $\mathscr{C}_{\mathscr{N}}$ is polynomial-time reducible to $\mathit{StaticEquivalence}_{\mathscr{D}}$.*

The latter $\mathit{StaticEquivalence}_{\mathscr{D}}$ decision problem is well-studied and in most cases of deduction systems of interest was found to be decidable, which implies that the $\mathit{AttackDetectability}_{\mathscr{D}\cup\mathscr{N}}$ problem is also decidable for most deduction systems of interest.

4.2 Monitor Synthesis

In our setting an attack definition relies on humans to specify also the intended execution, but this execution is not present when searching whether a concrete execution is an attack. Thus we need to synthesize tests that will detect whether an execution is an attack by relying solely on the contents of the actual execution.

Let (Λ,Λ') be a detectable attack presentation. By definition there exists at least one equation $C_1 \stackrel{?}{=} C_2$ either in P^f_Λ or in $P^f_{\Lambda'}$ that is not satisfied by the other trace. We add it to the tests of the monitor. If the equation is in P^f_s the monitor interrupts the protocol if it is not satisfied, whereas if it is in $P^f_{s'}$ the monitor interrupts the protocol if it is satisfied.

5 Attack detection in practice

We present in this section a simple example, the ISO/IEC 9797-1 protocol, especially its manual authentication mecanism 7a described in [12]. The normal run of the protocol is, after a human user sent D and R to the two devices A and B:

$$\begin{array}{rl} A \textbf{ knows} & A,B,D,R \\ B \textbf{ knows} & A,B,D,R \\ A \to B & : h(A,D,k_A,R) \\ B \to A & : h(B,D,k_B,R) \\ A \to B & : k_A \\ B \to A & : k_B \end{array}$$

A dishonest participant i can sent back the first message directly to a honest participant a willing to play the rôle A, and completely impersonate B during the session:

$$\begin{array}{rl} a \to I & : h(A,D,k_A,R) \\ I \to a & : h(A,D,k_A,R) \\ a \to I & : k_A \\ I \to a & : k_A \end{array}$$

We let $P = (\{A,B\},\mathrm{tr}_P)$ be the definition of the protocol, $E = (\{a,i\},\mathrm{tr}_E,\{a\mapsto A, i\mapsto I\})$ be the execution of the protocol P describing the attack, and $M = (\{A,B\},\mathrm{tr}_M)$ with:

$$\begin{cases} \mathrm{tr}_M(A) & = (?A,?B,?D,?R,?k_A,!h(A,D,k_A,R),?h(A,D,k_B,R),!h(A,D,k_B,R),?k_B) \\ \mathrm{tr}_M(B) & = \mathrm{input}(\mathrm{tr}_P(B)) \end{cases}$$

The implementation of this monitor would be:

$$\Phi_M(A) = \quad (?v_1;?v_2;?v_3;?v_4;!v_6 \text{ with } \left\{x_6 \stackrel{?}{=} h(x_1,x_3,x_5,x_4)\right\};?v_7,!v_8 \text{ with } \left\{x_8 \stackrel{?}{=} x_7\right\})$$

The two logs for the regular execution and the attack are respectively, with this implementation:

$$\begin{cases} (!h(A,D,k_A,R);!h(B,D,k_B,R)) & \text{(normal)} \\ (!h(A,D,k_A,R);!h(B,D,k_A,R)) & \text{(attack)} \end{cases}$$

and the test $x_1 \stackrel{?}{=} x_2$ is satisfied by the log of attack trace but not by the log of the normal execution. Thus the monitor can reject the attack from the log when this equality is satisfied. A more robust monitor would send the last two messages k_A and k_B as in that case we know of no other attack even when keys are guessable or the hash function is weak [5].

6 Conclusion

In future work we plan to generate monitor implementations from several roles, and to study test simplification techniques for efficiency. We also need to extend the monitor construction in order to protect a protocol from all the refinements of an attack.

References

[1] Martín Abadi & Véronique Cortier (2006): *Deciding knowledge in security protocols under equational theories*. *Theor. Comput. Sci.* 367(1-2), pp. 2–32, doi:10.1016/j.tcs.2006.08.032.

[2] Wihem Arsac, Giampaolo Bella, Xavier Chantry & Luca Compagna (2009): *Validating Security Protocols under the General Attacker*. In Pierpaolo Degano & Luca Viganò, editors: *Foundations and Applications of Security Analysis, ARSPA-WITS 2009, York, UK, March 28-29, 2009, Revised Selected Papers*, *Lecture Notes in Computer Science* 5511, Springer, pp. 34–51, doi:10.1007/978-3-642-03459-6_3.

[3] Giampaolo Bella, Stefano Bistarelli & Fabio Massacci (2003): *A Protocol's Life After Attacks...* In Bruce Christianson, Bruno Crispo, James A. Malcolm & Michael Roe, editors: *Security Protocols, 11th International Workshop, Cambridge, UK, April 2-4, 2003, Revised Selected Papers*, *Lecture Notes in Computer Science* 3364, Springer, pp. 3–10, doi:10.1007/11542322_2.

[4] Yannick Chevalier & Michaël Rusinowitch (2010): *Compiling and securing cryptographic protocols*. *Inf. Proc. Lett.* 110(3), pp. 116–122, doi:10.1016/j.ipl.2009.11.004.

[5] Stéphanie Delaune, Steve Kremer & Ludovic Robin (2017): *Formal Verification of Protocols Based on Short Authenticated Strings*. In: *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pp. 130–143, doi:10.1109/CSF.2017.26.

[6] Nachum Dershowitz & David A. Plaisted (2001): *Rewriting*. In: *Handbook of Automated Reasoning*, Elsevier and MIT Press, pp. 535–610, doi:10.1016/B978-044450813-3/50011-4.

[7] Maria-Camilla Fiazza, Michele Peroli & Luca Viganò (2015): *Defending Vulnerable Security Protocols by Means of Attack Interference in Non-Collaborative Scenarios*. *Front. ICT* 2015, doi:10.3389/fict.2015.00011.

[8] Mei Lin Hui & Gavin Lowe (1999): *Safe Simplifying Transformations for Security Protocols*. In: *Proceedings of the 12th IEEE Computer Security Foundations Workshop, CSFW 1999, Mordano, Italy, June 28-30, 1999*, IEEE Computer Society, pp. 32–43, doi:10.1109/CSFW.1999.779760.

[9] Zhiwei Li & Weichao Wang (2012): *Towards the attacker's view of protocol narrations (or, how to compile security protocols)*. In Heung Youl Youm & Yoojae Won, editors: *7th ACM Symposium on Information, Computer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012*, ACM, pp. 44–45, doi:10.1145/2414456.2414481.

[10] Gavin Lowe (1996): *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR*. *Software - Concepts and Tools* 17(3), pp. 93–102, doi:10.1007/3-540-61042-1_43.

[11] Roger M. Needham & Michael D. Schroeder (1978): *Using Encryption for Authentication in Large Networks of Computers*. Commun. ACM 21(12), pp. 993–999, doi:10.1145/359340.359342.

[12] Ludovic Robin (2018): *Vérification formelle de protocoles basés sur de courtes chaines authentifiées. (Formal verification of protocols based on short authenticated strings)*. Ph.D. thesis, University of Lorraine, Nancy, France. Available at https://tel.archives-ouvertes.fr/tel-01767989.

[13] Paul F. Syverson & Catherine A. Meadows (2000): *Dolev-Yao is no better than Machiavelli*. In: *Proceedings of the first* Workshop on Issues in the Theory of Security (WITS'00), doi:10.21236/ADA464936.

[14] F. Javier Thayer, Jonathan C. Herzog & Joshua D. Guttman (1998): *Strand Spaces: Why is a Security Protocol Correct?* In: *Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3-6, 1998, Proceedings*, IEEE Computer Society, pp. 160–171, doi:10.1109/SECPRI.1998.674832.

A Relation with the notion of static equivalence

The notions of equivalence *wrt* the refinement relation and static equivalence are strongly related. The different setting is justified by the different handling of nonces: in [1] contexts can contain any constant, so the secret constants in a trace have to be protected using π-calculus' ν operator, while we disallow non-public constants in contexts, which means that no constants can be used but the ones in the deduction system or those published (explicitly or implicitly) in the sequence of messages. We prove in Theo. 3 that the two notions are identical modulo the generation of new constants with the deduction system $\mathscr{N}$ of Ex. 2.

The *AttackDetectability* problem defined in this paper is new, but it is strongly related to the static equivalence problem. In order to show this relation, let us introduce *frames*, which are strands with a hidden set of constants.

Definition 17. (Frames, [1]) A *frame* is a couple $(\tilde{n}, s)$ where $\tilde{n}$ is a set of constants and s is a positive trace, and is usually denoted $\nu\tilde{n}.s$.

Technically the definition in [1] replaces *positive trace s* by a substitution of domain $x_1, \ldots, x_n$ that we have noted σ_s. The application of a context C on a frame $\nu\tilde{n}.s$ is equal to the application of C on s. We are now ready to define the static equivalence problem for a deduction system $\mathscr{D}$.

Decision Problem 2. *StaticEquivalence*$_{\mathscr{D}}(\varphi_1, \varphi_2)$

Input: Two frames $\varphi_i = \nu\tilde{n}_i.s_i$ for $i = 1, 2$

Output: YES if the frames have an equal length and for all pair C_1, C_2 of public contexts and all function $\theta : \mathrm{Var}(C_1) \cup \mathrm{Var}(C_2) \to \{x_1, \ldots, x_n\} \cup (\mathscr{C} \setminus (\tilde{n}_1 \cup \tilde{n}_2))$ we have $C_1\theta\sigma_{s_1} = C_2\theta\sigma_{s_1}$ if, and only if, $C_1\theta\sigma_{s_2} =_{\mathscr{E}} C_2\theta\sigma_{s_2}$.

Attack detectability is related to static equivalence with the following theorem:

Theorem 3. *Let $\mathscr{D}$ be a deduction system, and $\mathscr{N}$ be the deduction system of Ex. 2. Then AttackDetectability*$_{\mathscr{D}\cup\mathscr{N}}$ *on strands that do not contain symbols of $\mathscr{C}_{\mathscr{N}}$ is polynomial-time reducible to StaticEquivalence*$_{\mathscr{D}}$.

Proof. Given a trace s we let $\varphi_s = \nu\mathrm{Const}(s).[s_1, \ldots, s_n]$. This construction is clearly polynomial time. Let t_1, t_2 be the two traces in the presentation of the attack $\mathscr{A}$ on I_P to M.

First, if t_1 and t_2 are of different length or have different label sequence, one can respond to the *AttackDetectability* in polynomial time. So let us assume the two strands have the same length and the same label sequence. Also, we assume that t_1, t_2 do not contain the symbols of the $\mathscr{N}$ deduction system.

Let us prove that t_1 and t_2 are discernable wrt the deduction system $\mathscr{D} \cup \mathscr{N}$ if, and only if, the frames $\varphi_{s_1}, \varphi_{s_2}$ are not statically equivalent wrt the deduction system $\mathscr{D}$.

First let us assume that the attack presentation (t_1, t_2) is detectable, and wlog assume that t_1 does not refine t_1 for the deduction system $\mathscr{D} \cup \mathscr{N}$. Thus there exists two $\mathscr{D} \cup \mathscr{N}$-contexts such that $C_1 t_1 =_{\mathscr{E}} C_2 t_1$ but $C_1 t_2 \neq_{\mathscr{E}} C_2 t_2$. Since we assume that constants occurring in s_1, s_2 are away from $\mathscr{C}_{\mathscr{N}}$, we construct C_1', C_2' and θ' as follows. For each constant $c \in \mathscr{C}_{\mathscr{N}}$ occurring in C_1 or C_2:

- replace in the contexts c with a new variable x_c;
- define θ as follows:

$$\theta(x) = \begin{cases} x_i & \text{if } x \in \mathrm{Var}(C_1) \cup \mathrm{Var}(C_2) \\ c & \text{if } x = x_c \end{cases}$$

By construction C_1', C_2' are $\mathscr{D}$-public contexts and θ maps each variable of these contexts to either a variable or to a constant away from s_1, s_2. Thus C_1', C_2' and θ' are witnesses that the frames φ_s and $\varphi_{s'}$ are not $\mathscr{D}$-statically equivalent.

Conversely assume that the two frames $\varphi_{s_1}, \varphi_{s_2}$ are not statically equivalent. Then there exist $\mathscr{D}$ contexts C_1, C_2 and $\theta : \mathrm{Var}(C_1) \cup \mathrm{Var}(C_2) \to \mathscr{C} \setminus (\mathrm{Const}(s_1) \cup \mathrm{Const}(s_2))$ such that wlog $C_1 \theta \varphi_{s_1} =_{\mathscr{E}} C_2 \theta \varphi_{s_1}$. Replacing each free constant c by a constant in $\mathscr{C}_{\mathscr{N}}$ yields appropriate contexts for the $\mathscr{D} \cup \mathscr{N}$ attack detectability. □