INTERNSHIP REPORT

Supporting additional equational theories in Tamarin

Written by Charles Duménil

Supervised by Jannik Dreier and Steve Kremer

TAMARIN

Acknowledgements

Remerciements

J'adresse mes remerciements les plus sincères à ceux sans qui ce stage n'aurait pas était possible.

Premiers d'entre eux, Steve Kremer, qui m'a accepté au sein de son équipe. Steve m'a fait confiance malgré mon manque d'expérience dans le domaine des méthodes formelles. De par la clarté de ses explications, il a efficacement accéléré ma compréhension des notions mises en jeu dans le sujet. Il m'a fait part de ses connaissances, m'a intelligemment guidé dans les phases de recherche et surtout, m'a permis de réaliser un stage qui, de part les compétences qu'il requérait, m'a comblé.

Enfin, en me faisant entrer au Loria, Steve m'a ouvert les portes de la recherche, et si aujourd'hui j'ai obtenu un contrat doctoral, c'est en majeur partie grâce à lui.

Mes remerciements s'adressent aussi tout particulièrement à Jannik Dreier qui m'a co-encadré avec Steve. Jannik m'a chaleureusement accompagné tout au long du stage. Par sa patience et sa bienveillance, il m'a permis de surmonter les nombreuses difficultés auxquelles j'ai fait face, que ce soit dans l'apprentissage des méthodes formelles ou dans la compréhension de Tamarin dont il est l'un des développeurs.

Jannik et Steve ont tous deux pris beaucoup de leur du temps pour m'aider à rédiger ce rapport. Pour cela, je les remercie expressément.

Mes remerciements s'adressent également à mes tuteurs pédagogiques. Anne Gégout-Petit, responsable du Master que j'ai suivi, pour avoir compris ma volonté d'intégrer la recherche et donné son accord pour que je réalise ce stage bien qu'il ne fût pas dans la continuation logique du Master. Et Pierre-Jean Spaenlehauer, qui m'a enseigné les bases de la cryptologie, et qui, par sa sympathie, m'a donné envie d'approfondir mes connaissances dans la discipline.

Mes derniers remerciements s'adressent aux membres de l'équipe PESTO, qui m'ont accompagné pendant cinq mois et demi.

Charles Duménil, le 15 Septembre 2016.

Contents

1 Introduction

My internship took place in the Inria Nancy-Grand Est centre where I worked in the Pesto team and was supervised by Steve Kremer and Jannik Dreier.

1.1 Internship context

INRIA
The National Institute for Research in Computer Science and Control (Inria) is a public science and technology institution placed under the supervision of the ministries of research and industry. The institute consists in 8 research centres in France (Paris, Rennes, Sophia Antipolis, Grenoble, Nancy, Bordeaux, Lille and Saclay) and a head office in Rocquencourt. About 2600 people including 1800 scientists work at Inria.

The Inria Nancy-Grand Est centre was created in 1986. Divided into 25 project-teams, it welcomes around 170 scientists under the direction of Sylvain Petitjean. Its research is built around three fields: "Cognition: perception, language and knowledge", "Simulation, optimisation and control of complex systems" and "Safety and security of computer systems" [2].

The centre shares its premises with the Loria.

LORIA
The Lorraine Research Laboratory in Computer Science and its Applications (Loria) is a research unit, common to CNRS, the University of Lorraine and Inria. It was created in 1997 with the mission to conduct fundamental and applied research in computer science. About 500 people work at LORIA under the direction of Jean-Yves Marion. Loria contains 30 research teams divided into 5 departments named "Algorithms, Computation, Image and Geometry", "Formal methods", "Networks, Systems and Services", "Knowledge and Language Management" and "Complex Systems, Artificial Intelligence and Robotics" [4].

Project-team PESTO (Proof techniques for security protocols)
Pesto team is both part of Loria and Inria. As a team of the formal methods department, its aim is to build formal models and techniques for computer-aided analysis and design of security protocols. With the omnipresence of Internet and the development of new technologies, new protocols are created, designed to ensure new security properties like privacy for E-voting, anonymity for E-cash or again untracability for RFID. The team has the goal of developing tools that can prove that those protocols are secure [1].

It is composed of 10 permanent members, including Steve Kremer, the team leader and Jannik Dreier, inter alia, co-developer of the Tamarin prover.

The Tamarin Prover
Tamarin is a software initially developed by Simon Meier and Benedikt Schmidt at the Swiss Federal Institute of Technology in Zurich. It aims at automatically proving security properties of cryptographic protocols.

1.2 Research problem

Nowadays, security protocols can be complex and, even if the cryptographic primitives, like encryption, are supposed perfect, a protocol can contain flaws, logical mistakes can remain in the structure of a protocol. To verify if such a mistake exists, we perform a formal analysis of the protocol; because it can be long, the analysis is automated by computer-based tools. Tamarin is one these tools able to automatically verify security properties, like secrecy, on protocols.

In Tamarin, we model protocol messages by terms and algebraic properties of cryptographic primitives by rewriting rules. For example, if $enc(m, k)$ represents the symmetric encryption of the message m with the key k and $dec(c, k)$, the symmetric decryption of the cyphertext c, then we have the rewriting rule $dec(enc(m, k), k) \rightarrow m$. It means that the decryption with the key k of the encryption with the k of the message m can be rewritten as m itself. This rule is an example of an important class of rules called subterm rewriting rules where the right-hand side term, m in the example, is a subterm of the left-hand side term. This class is already treated by Tamarin.

Consider FOO protocol which is used for E-voting. Cryptographic primitives for FOO are blinding and signing functions. They lead to this kind of rewriting rule: $unblind(sign(blind(v, r), k), r) \rightarrow sign(v, k)$ which models the act of signing a blinded vote, and then, unblinding it to get only the signed vote. This rewriting rule is not a subterm rewriting rule, the message $sign(v, k)$ of the right-hand side is not contained in $unblind(sign(blind(v, r), k), r)$. It is part of more general class of rewriting rules, that we will call context subterm rules.

The objectives of my internship is to describe how we can model this class of rules in Tamarin and to implement it.

2 Background

Tamarin is a software developed to automatically analyse cryptographic protocols. The analysis is symbolic, which means that Tamarin considers that cryptographic primitives, like hashing or encryption functions, are perfect. But flaws can exist at the specification level of the protocol. One of the most known examples is the "man-in-the-middle" attack that can be executed on the Needham-Schroeder Public Key protocol (NSPK) by an active attacker [7].

2.1 Cryptographic protocol

As defined in [17], a protocol is a series of steps, involving two or more parties, designed to accomplish a task. Exchanging messages with somebody is one of these possible tasks. Nowadays a large part of exchanges are made via the Internet. This network is considered as public, which means anybody can, at least, read messages transiting on the Internet. We use cryptography to establish some security properties on the messages that are sent. Cryptographic protocols are protocols that use cryptography. They are designed to ensure security properties when parties exchange messages.

Example 1. Transport Layer Security (TLS) is a cryptographic protocol designed to secure web transactions over the Internet.

Cryptographic protocols are based on functions called cryptographic primitives. Three of the most usual primitives are encryption, decryption and hashing. Encryption is an operation that transforms a message said "plaintext", in another message, unintelligible, said "cyphertext", in order to ensure secrecy [3]. Decryption is the inverse operation that transforms back the cyphertext into the plaintext. A hash function maps a variable-length message into a fixed-length value called a hash code, designed in such a way that it provides a digest of the message.[18]

There exists two ways to encrypt, that are described in [18] as follows:

- Symmetric encryption is a cryptographic system in which encryption and decryption are performed using the same secret key, a value only known by the participants. We note $senc(m, k)$ and $sdec(m, k)$ for symmetric encryption and decryption of a message m with the key k.
- Asymmetric encryption is a cryptographic system in which encryption and decryption are performed using different keys, respectively a public key and a private (secret) key. It is also known as public-key encryption. We note $pk(k)$ for the public key associated to k, and $aenc(m, pk(k))$ and $adec(m, k)$ for asymmetric encryption and decryption of a message m with the keys $pk(k)$ and k. One of the most widely used public-key cryptographic system is RSA. It is based on the difficulty of finding the prime factors of a composite number.

In these two cryptographic systems, it must be assumed that the only secret lies in the secret key. For example, RSA is a public algorithm: confidentiality is ensured as long as the private key of the agent receiving the message is unknown by others.

In order to model cryptographic protocols, it is common to use the *Alice&Bob* notation. For instance, we write:

$$A \to B : \quad m$$

to express that Alice, represented by A sends the message m to Bob as B.

Supposing that agents A and B share a symmetric key, A sending a symmetrically encrypted message m to B will be denoted this way:

$$A \to B : \quad \{m\}_{A,B}$$

An agent A sending an asymmetrically encrypted message m to an agent B will be denoted this way:

$$A \to B : \quad \{m\}_{pk(B)}$$

Example 2. NSPK protocol (Needham–Schroeder Public Key) is designed for two agents A and B to ensure their authentication through a public channel. It can be represented as follow:

$$\begin{aligned} A &\to B : \ \{A, Na\}_{pk(B)} \\ B &\to A : \ \{Na, Nb\}_{pk(A)} \\ A &\to B : \ \{Nb\}_{pk(B)} \end{aligned}$$

where Na and Nb are nonces randomly generated (number only used once).

- First, agent A sends its identity with a nonce Na. The message is asymmetrically encrypted with the public key of agent B. The only person able to decrypt is therefore B.
- Second, B decrypts the received message with its private key. It sees that the communication is initiated by A. At this stage, Na is supposed to be known only by A and B. Agent B generates a new nonce Nb and sends to A the nonces Na and Nb encrypted with the public key of A.
- At last, A decrypts the message, reads the nonce Na which convinces him that the communication is indeed established with B, and, the same way as B, uses Nb paired with each message sent to ensure the authentication [7].

2.2 Formal verification of security protocols

Security protocols are designed to ensure security properties in a hostile environment [13]. Such a hostile environment is represented by the presence on the network of a dishonest third party that we call adversary. We can consider two kinds of adversaries. A passive adversary is a dishonest agent that can only tap the communication line and tries to decrypt the intercepted messages. Conversely, an active adversary can, in addition, impersonate another agent's identity and may alter or replay messages [8]. Since we consider cryptographic primitives as perfect, the possibilities of the adversary are formalized using a formal model based on a term algebra.

One particularly influential formal model for an active attacker is called the Dolev-Yao model. In this model, the protocol security problem is transformed into a search based on a term rewriting system [6]. Given a security protocol and a security property, Tamarin verifies if there exists an execution of this protocol that does not satisfy the property. To do so, it performs an exhaustive search under the form of graphs for such an execution, and consider the property as valid if no counter-example has been found. However, if it finds an attack, it shows it.

Example 3. Consider an active adversary I over a public network where an agent A starts a communication with B using NSPK to authenticate one another. Then, by impersonation of B, I is able to learn the shared nonce Nb used for authentication:

$$\begin{aligned} A \to I &: \{A, Na\}_{pk(I)} \\ I \to B &: \{A, Na\}_{pk(B)} \\ A \longleftarrow B &: \{Na, Nb\}_{pk(A)} \\ A \to I &: \{Nb\}_{pk(I)} \\ I \to B &: \{Nb\}_{pk(B)} \end{aligned}$$

This is called the "man-in-the-middle" attack. It has been discovered by G. Lowe in 1995 and is fixed by adding the identity B in the message sent by B resulting in the NSLPK protocol.

Even when considering a Dolev-Yao model with perfect cryptography, automatic verification is not always feasible. For example, the intruder can build an infinite number of messages, there may be an unbounded number of sessions, the equational theories manipulated by the adversary may add complications. In general, verification of security protocols is thus an undecidable problem.

Tamarin disposes of two methods of automatic verification. In the first one, it consider a security property as a trace property on a protocol, this will be defined later, and tests if this property is verified by all the possible executions of a protocol. It can be used to prove properties like secrecy, unforgeability or eligibility. In the second, Tamarin considers two different execution of a protocol and tries to distinguish them. If not, we say that the two executions are observationally equivalent. It can be used to prove properties like untracability, anonymity or privacy [9].

To model protocols and security properties we use notions from formal language theory. They are drawn from [11], [15] and [16].

2.3 Modelling messages

2.3.1 Term algebra

Definition 2.1. A signature Σ^0 is a set of function symbols, each having an arity (number of parameters) $n \geq 0$. We call function symbols of arity 0 constants.

Example 4. (Math) As a mathematical example, we can formalise natural numbers as defined by Peano. Consider the signature $\Sigma_P^0 = \{0, s, +\}$, where

the constant 0 represents the first number, s of arity 1 represents the successor function and + of arity 2 represents addition.

Example 5. (Crypto) As a cryptographic example, the NSPK protocol uses asymmetric encryption. We need a signature with functions to encrypt, decrypt and to associate a public key to a private key. Encryption and decryption functions will be of arity 2 because they have as parameters a message and a key, the public key function will be of arity 1. Additionally, to send multiple messages, we consider the pair function of arity 2, noted $\langle \_, \_ \rangle$ and the first and second functions of arity 1, noted *fst* and *snd*, to extract messages from the pair. For instance, we take:

$$\Sigma^0_{\mathcal{NSPK}} = \{\langle \_, \_ \rangle, fst, snd, aenc, adec, pk\}.$$

Definition 2.2. An order-sorted signature $\Sigma = (S, \leq, \Sigma^0)$ is a set of sorts S, a partial order $\leq$ and a signature Σ^0 associated with sorts with the following two properties. First, for every $s \in S$, the connected component C of s in $(S, \leq)$ has a top sort denoted $top(s)$ such that $c \leq top(s)$ for all $c \in C$. Second, for every $f : s_1 \times \cdots \times s_k \to s$ in Σ^0 with $k \geq 1$, there is an $f : top(s_1) \times \cdots \times top(s_k) \to s$ in Σ^0 called the top sort overloading of f .

Definition 2.3. Let Σ be a signature, we assume that, for each sort $s \in S$ there is V_s be a countably infinite set of variables with sorts s and define $\mathcal{V} = \uplus_{s \in S} \mathcal{V}_s$. Let $\mathcal{V}' \subset \mathcal{V}$ a term algebra $\mathcal{T}_\Sigma(\mathcal{V}')$ over Σ is the least set recursively defined by:

- $\mathcal{V}' \subseteq \mathcal{T}_\Sigma(\mathcal{V}')$,
- if $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{V})$ and $f \in \Sigma$ of arity n, then $f(t1, \ldots, t_n) \in \mathcal{T}_\Sigma(\mathcal{V}')$

$\mathcal{T}_\Sigma(\mathcal{V}')_s$ denote the terms of $\mathcal{T}_\Sigma(\mathcal{V}')$ with sort s. The set of ground terms $\mathcal{T}_\Sigma$ consists of terms built without variable ($\mathcal{T}_\Sigma = \mathcal{T}_\Sigma(\emptyset)$)

Example 6. (Math) Consider the order-sorted signature
$\Sigma_P = (\{N, EN, ON\}, \{EN \leq N, ON \leq N\}, \Sigma^0_P)$ where:

- every $n \in \mathcal{T}_{\Sigma_P}$ is with sort N,
- 0 is with sort EN as well as every $n = s(s(n'))$ where n' is with sort EN,
- $s(0)$ is with sort ON as well as every $n = s(s(n'))$ where n' is with sort ON.

This describes natural numbers, even numbers and odd numbers. The top sort is N. For instance, an element of $\mathcal{T}_{\Sigma_P}(\mathcal{V})$ is $s(s(0)) + s(x)$ where $x \in \mathcal{V}$. Notice we use $s(s(0)) + s(x)$ for $+(s(s(0)), s(x))$. This is syntactic sugar called infix notation.

Example 7. (Crypto) Consider the order-sorted signature
$\Sigma_{\mathcal{NSPK}} = (S, \leq, \Sigma^0_{\mathcal{NSPK}})$ where S describes sorts of messages. Let $m, k \in \mathcal{V}$, two elements of $\mathcal{T}_{\Sigma_{\mathcal{NSPK}}}(\mathcal{V})$ are $aenc(m, pk(k))$ and $adec(aenc(m, pk(k)), k)$.

These two terms correspond respectively to the asymmetric encryption of a message m with the public key associated to k and asymmetric decryption of the precedent term with the secret key k. We can notice that, to make sense, $adec(aenc(m, pk(k)), k) \in \mathcal{T}_{\Sigma_{\mathcal{NSPK}}}(\mathcal{V})$ should equal m, but since they are syntactically different, we need to introduce an equational theory to enforce the equality.

Definition 2.4. An equation over Σ is a pair of terms $l, r \in \mathcal{T}_\Sigma(\mathcal{V})$, written $l \simeq r$. Let E be a set of equations. The couple $\varepsilon = (\Sigma, E)$ defines an equational presentation that is identified to the equational theory $=_\varepsilon$, the smallest congruence over Σ containing all instances of equations of E.

Example 8. (Math) Equations over Σ_P are:

$$E_P = \left\{ \begin{array}{cc} x + 0 \simeq x & x + s(y) \simeq s(x + y) \end{array} \right\}$$

Example 9. (Crypto) Equations over Σ_{NSPK} are:

$$E_{\mathcal{NSPK}} = \left\{ \begin{array}{cc} fst(\langle x, y\rangle) \simeq x & snd(\langle x, y\rangle) \simeq y \\ adec(aenc(m, pk(k)), k) \simeq m & \end{array} \right\}$$

The first two equations describe the extraction of message from a pair while the third one means that decrypting a message cyphered with the public key, using the associated private key, gives back the initial message.

We define the equational theory used to model the NSPK protocol as

$$\mathcal{NSPK} = (\Sigma_{\mathcal{NSPK}}, E_{\mathcal{NSPK}})$$

Remark. By convention, we write $\langle \_, \_, \ldots, \_\rangle$ for $\langle \_, \langle \_, \ldots \langle \_, \_\rangle\rangle\rangle$. This leads for instance to the equation $y =_\varepsilon snd(fst(\langle x, y, z\rangle))$.

Finally we want Tamarin to retain m instead of $adec(aenc(m, pk(k)), k)$, to achieve this, we give an orientation to equations.

Definition 2.5. A rewriting rule over Σ is an oriented pair of terms $l, r \in \mathcal{T}_\Sigma(\mathcal{V})$, written $l \to r$. A rewriting system $\mathcal{R}$ is a set of rewriting rules.

Example 10. (Math) Rewriting system over Σ_P :

$$\mathcal{R}_P = \left\{ \begin{array}{cc} x + 0 \to x & x + s(y) \to s(x + y) \end{array} \right\}$$

We can now rewrite $s(s(0) + s(0))$:
$s(s(0)) + s(0) \to s(s(s(0) + 0)) \to s(s(s(0)))$

Example 11. (Crypto) Rewriting system over Σ_{NSPK}:

$$\mathcal{R}_{\mathcal{NSPK}} = \left\{ \begin{array}{cc} fst(\langle x, y\rangle) \to x & snd(\langle x, y\rangle) \to y \\ adec(aenc(m, pk(k)), k) \to m & \end{array} \right\}$$

Since the right-hand side term m of the rewriting rule $adec(aenc(m, pk(k)), k) \to m$, is a subterm of its left-hand side term, we say the rewriting rule is a subterm rewriting rule. Every rule of this rewriting system is a subterm rule.

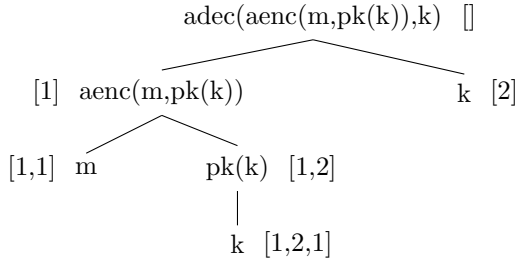
Definition 2.6. A position p is a sequence of positive integers. The subterm $t|_p$ of a term t at position p is obtained inductively as follows:

- if $p = []$, the empty sequence, then $t|_p = t$.
- if $p = [i] \cdot p'$, the concatenation of $[i]$ and p', for a positive integer i and a sequence p', and $t = f(t_1, \ldots, t_n)$ for $f \in \Sigma$ and $1 \leq i \leq n$ then $t|_p = t_i|_{p'}$.
- otherwise $t|_p$ is not defined and p is not a valid position.

We use $t[s]_p$ to denote the term t where the occurrence of the subterm $t|_p$ at position p has been replaced by s. The set $St(t)$ of syntactic subterms of a term t is defined as $\{t|_p \mid p \text{ valid position in } t\}$. For a term t, we define $vars(t) = St(t) \cap V$. We use $root(t)$ to denote f if $t = f(t_1, \ldots, t_k)$ for some $f \in \Sigma^0$ and t itself otherwise. A function symbol f is irreducible with respect to a rewriting system $\mathcal{R}$ if there is no $l \to r \in \mathcal{R}$ with $root(l) = f$.

Example 12. (Crypto) The term $t = adec(aenc(m, pk(k)), k)$ has six subterms:
$t|_{[]} = adec(aenc(m, pk(k)), k)$
$t|_{[1]} = aenc(m, pk(k))$
$t|_{[1,1]} = m$
$t|_{[1,2]} = pk(k)$
$t|_{[1,2,1]} = k$
$t|_{[2]} = k$.
The set of variables of t is $vars(t) = \{m, k\}$, $root(t) = adec$ and $adec$ is not irreducible in $\mathcal{R}_{\mathcal{NSPK}}$ because of the rule $adec(aenc(m, pk(k)), k) \to m$.

We can represent a term and its positions as a tree:



Definition 2.7. A substitution σ is a function from $\mathcal{V}$ to $\mathcal{T}_\Sigma(\mathcal{V})$ that corresponds to the identity function except on a finite set of variables which we denote with $dom(\sigma)$. We use $range(\sigma)$ to denote the image of $dom(\sigma)$ under σ and define $vrange(\sigma) = vars(range(\sigma))$.

Example 13. (Crypto) Consider the substitution
$\sigma = \{x \mapsto aenc(m, pk(k)), y \mapsto k\}$ and the term $t = adec(x, y)$, we have that $t\sigma = adec(aenc(m, pk(k)), k)$. Notice that the application of a substitution is written in postfix notation, and the application is homomorphically lifted from variables to terms.

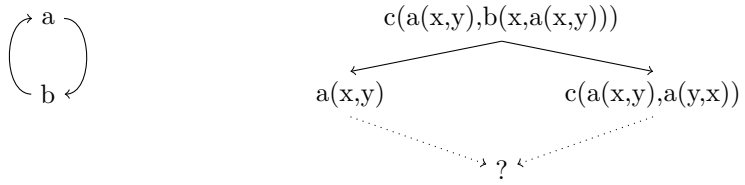
Definition 2.8. An equation $t \simeq u$ is regular if $vars(t) = vars(u)$ and sort-preserving if for all substitutions σ. It holds that $t\sigma \in \mathcal{T}_\Sigma(\mathcal{V})_s$ if and only if $u\sigma \in \mathcal{T}_\Sigma(\mathcal{V})_s$. An equational presentation is regular (respectively sort-preserving) if all its equations are.

2.3.2 Rewriting theory

Not all rewriting systems are allowed in Tamarin, they have to respect some properties: for instance if we take the rules $a \to b$ and $b \to a$, the system will turn forever from a to b then back to a etc... To illustrate another property, consider three functions a, b and c of arity 2, the rewriting rules: $c(y, b(x, y)) \to x$ and $b(x, a(x, y)) \to a(y, x)$ and the term $c(a(x, y), b(x, a(x, y)))$. Applying the first rewriting rule on this term gives $a(x, y)$ while applying the second one gives

$c(a(x,y),a(y,x))$ but nothing said that $c(a(x,y),a(y,x)) \to a(x,y)$. The first problem is about termination, the second one is about confluence. They can be represented the following way:

Non termination Non confluence



For the following definitions, we use $\to^+$ to denote the transitive closure and $\to^*$ to denote the transitive-reflexive closure of a relation $\to$.

Definition 2.9. A rewriting system $\mathcal{R}$ defines a rewriting relation $\to_\mathcal{R}$ with $s \to_\mathcal{R} t$ if there is a position p in s, a rule $l \to r \in \mathcal{R}$, and a substitution σ such that $s|_p = l\sigma$ and $s[r\sigma]_p = t$.

A rewriting system $\mathcal{R}$ is terminating if there is no infinite sequence $(t_i)_{i\in\mathbb{N}}$ of terms with $t_i \to_\mathcal{R} t_{i+1}$.

A rewriting system $\mathcal{R}$ is confluent if for all terms t_1, s_1, s_2 with $t_1 \to^*_\mathcal{R} s_1$ and $t_1 \to^*_\mathcal{R} s_2$, there is a term t_2 such that $s_1 \to^*_\mathcal{R} t_2$ and $s_2 \to^*_\mathcal{R} t_2$.

A rewriting system $\mathcal{R}$ is convergent if it is terminating and confluent. In this case, we use $t\downarrow_\mathcal{R}$ to denote the unique normal form of t.

A rewriting system $\mathcal{R}$ is subterm-convergent if it is convergent and for each rule $l \to r \in \mathcal{R}$, r is either a proper subterm of l or r is ground and in normal form.

A rewriting rule $l \to r$ is sort-decreasing if for all substitutions σ, $r\sigma \in \mathcal{T}_\Sigma(\mathcal{V})_s$ implies $l\sigma \in \mathcal{T}_\Sigma(\mathcal{V})_s$. A rewriting system is sort-decreasing if all its rules are.

We use $R^\simeq$ to denote the set of equations obtained from a rewriting system by replacing $\to$ by $\simeq$ in the rewriting rules.

Example 14. $\mathcal{R}_P$ and $\mathcal{R}_{NSPK}$ are subterm-convergent.

Definition 2.10. An ε-unifier of two terms s and t is a substitution σ such that $s\sigma =_\varepsilon t\sigma$. For $W \subseteq \mathcal{V}$, we use $unif_\varepsilon^W(s,t)$ to denote an ε-unification algorithm that returns a set of unifiers of s and t such that for all $\sigma \in unif_\varepsilon^W(s,t)$, $vrange(\sigma) \cap W = \emptyset$. The unification algorithm is complete if for all ε-unifier σ of s and t, there is $\tau \in unif_\varepsilon^W(s,t)$ and a substitution θ such that for all $x \in vars(s,t)$, $(x\tau)\theta =_\varepsilon x\sigma$. The unification algorithm is finitary if for all s and t, it terminates and returns a finite set of unifiers.

Analogously, an ε-matcher of two terms t and p is a substitution σ such that $t =_\varepsilon p\sigma$.

Example 15. (Crypto)
Two ε-unifiers (and also ε-matchers) of $adec(c,k)$ and m are:

- $\sigma_1 = \{m \mapsto adec(c,k)\}$, trivially, and
- $\sigma_2 = \{c \mapsto aenc(m, pk(k))\}$ because
  $adec(c,k)\sigma_2 = adec(aenc(m,pk(k)),k) =_\varepsilon m\sigma_2 = m$.

Proposition 2.1. *If the equations in ε can be oriented to obtain a convergent rewriting system $\mathcal{R}$, then $t =_\varepsilon s$ if and only if $t \downarrow_\mathcal{R} = s \downarrow_\mathcal{R}$*

A problem arises when we have an equational theory $\mathcal{AX}$ that is not orientable such as associative and commutative theories. We then use the notion of $\mathcal{R}, \mathcal{AX}$-rewriting for a rewriting system $\mathcal{R}$ and a not orientable equational theory $\mathcal{AX}$.

Definition 2.11. The rewriting relation $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is defined as $s \rightarrow_{\mathcal{R},\mathcal{AX}} t$ if there is a position p in s, a rewriting rule $l \rightarrow r \in \mathcal{R}$, and a substitution σ such that $s|_p =_{\mathcal{AX}} l\sigma$ and $s[r\sigma]_p = t$. If $\mathcal{AX}$-matching is decidable, then the relation $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is also decidable.

Definition 2.12. We say $\mathcal{R}, \mathcal{AX}$ is convergent if the relation $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is convergent. In this case, we denote the unique normal form of t with respect to $\mathcal{R}, \mathcal{AX}$-rewriting by $t \downarrow_{\mathcal{R},\mathcal{AX}}$. We say that $\mathcal{R}, \mathcal{AX}$ is coherent if for all t_1, t_2 and t_3, it holds that $t_1 \rightarrow_{\mathcal{R},\mathcal{AX}} t_2$ and $t_1 =_{\mathcal{AX}} t_3$ implies that there are t_4 and t_5 such that $t_2 \rightarrow^*_{\mathcal{R},\mathcal{AX}} t_4$, $t_3 \rightarrow^+_{\mathcal{R},\mathcal{AX}} t_5$, and $t_4 =_{\mathcal{AX}} t_5$. If $(\Sigma, \mathcal{R} \cup \mathcal{AX})$ is an equational presentation of $=_\varepsilon$ and $\mathcal{R}, \mathcal{AX}$ is convergent and coherent, then $t = \varepsilon s$ if and only if $t \downarrow_{\mathcal{R},\mathcal{AX}} = s \downarrow_{\mathcal{R},\mathcal{AX}}$. We call $(\Sigma, \mathcal{R}, \mathcal{AX})$ a decomposition of ε if the following holds:

- $(\Sigma, \mathcal{R}^\simeq \cup \mathcal{AX})$ is an equational presentation of $= \varepsilon$,
- $\mathcal{AX}$ is regular, sort-preserving and all equations contain only variables of top-sort,
- $\mathcal{R}$ is sort-decreasing and $\mathcal{R}, \mathcal{AX}$ is convergent and coherent and
- There is a complete and finitary $\mathcal{AX}$-unification algorithm.

Example 16. Consider the equational theory ε_{Xor} defined by $\Sigma_{Xor} = \{0, \oplus\}$ and $E = \mathcal{AC} \cup \mathcal{R}^\simeq_{Xor}$ where $\mathcal{AC} = \{x \oplus y \simeq y \oplus x, x \oplus (y \oplus z) \simeq (x \oplus y) \oplus z\}$ and $\mathcal{R}_{Xor} = \{x \oplus 0 \rightarrow x, x \oplus x \rightarrow 0\}$. We have that $x \oplus (x \oplus y) =_{\varepsilon_{Xor}} y$ but also:

$$\begin{array}{lll} x \oplus (x \oplus y) & =_{\mathcal{AC}} & (x \oplus x) \oplus y \\ \downarrow & & \downarrow \\ x \oplus (x \oplus y) & \neq_{\mathcal{AC}} & y \end{array}$$

showing that $\mathcal{R}_{Xor}, \mathcal{AC}$ is not coherent.

Definition 2.13. For an equational theory ε, we define the ε-instances of a term t as $insts_\varepsilon(t) = \{t' |\ \exists \sigma,\ t\sigma =_\varepsilon t'\}$. We use $ginsts_\varepsilon(t)$ to denote the set of ground ε-instances of t. To reason about ε-instances using a decomposition $(\Sigma, \mathcal{R}, \mathcal{AX})$ of ε, the finite variant property is often useful. A decomposition $(\Sigma, \mathcal{R}, \mathcal{AX})$ of an equational theory ε has the finite variant property if for all terms t, there is a finite set of substitutions$\{\tau_1, \ldots, \tau_k\}$ with $dom(\tau_i) \subseteq vars(t)$ such that for all substitutions σ, there is a substitution θ and $i \in \{1, \ldots, k\}$ with:

- $(t\sigma) \downarrow_{\mathcal{R},\mathcal{AX}} =_{\mathcal{AX}} (t\tau_i) \downarrow_{\mathcal{R},\mathcal{AX}} \theta$ and,
- $x\sigma \downarrow_{\mathcal{R},\mathcal{AX}} =_{\mathcal{AX}} (x\tau_i) \downarrow_{\mathcal{R},\mathcal{AX}} \theta$ for all $x \in vars(t)$.

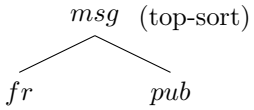
We call such a set of substitutions a complete set of $\mathcal{R}, \mathcal{AX}$-variants of t. For a decomposition with the finite variant property, we can use folding variant narrowing to compute a complete and minimal set of $\mathcal{R}, \mathcal{AX}$-variants of a term[11].

2.4 Modelling protocols

2.4.1 Cryptographic messages

Cryptographic messages are modelled under three ordered sorts of messages: msg, the top-sort of messages, and two incomparable subsorts fr and pub for fresh messages and public names. Their order can be represented as in the following graph:

Message sorts:



Fresh names are used in particular to model keys, nonces or other messages that supposed to be unique. Public names are suitable for messages that are not supposed to be secret, for instance an identity or a possible vote. fr and pub messages are supposed to be elements of the infinite sets FN and PN.

As a basis to model cryptographic primitives, we use an unsorted signature $\Sigma_{\mathcal{ST}}$ associated with a subterm-convergent rewriting system $\mathcal{R}_{ST}$ and define the equational theory $\mathcal{ST} = (\Sigma_{\mathcal{ST}}, \mathcal{R}^{\approx}_{\mathcal{ST}})$. It depends on the protocol that will be analysed. We note $\mathcal{T}$ for $\mathcal{T}_{\Sigma_{\mathcal{ST}} \cup FN \cup PN}(\mathcal{V})$ and $\mathcal{M}$ for ground terms in $\mathcal{T}$.

Example 17. We resume this using the NSPK example

$$\mathcal{NSPK} = (\Sigma_{\mathcal{NSPK}}, \mathcal{R}^{\approx}_{\mathcal{NSPK}})$$

$$\Sigma_{\mathcal{NSPK}} = \left\{ \begin{array}{ccc} \langle \_, \_ \rangle & fst(\_) & snd(\_) \\ aenc(\_, \_) & adec(\_, \_) & pk(\_) \end{array} \right\}$$

$$\mathcal{R}_{\mathcal{NSPK}} = \left\{ \begin{array}{c} fst(\langle x, y \rangle) \to x \quad snd(\langle x, y \rangle \to y \\ adec(aenc(m, pk(k)), k) \to m \end{array} \right\}$$

2.4.2 Labeled Multiset Rewriting

In Tamarin, the execution of a security protocol in the context of an adversary is modeled as an infinite labeled transition system, whose states consist of the adversary's knowledge, the messages on the network, freshly generated values, and the protocol's states. Those states are modelled as finite multisets of facts.

Definition 2.14. Consider a signature Σ_{Facts}, we define the set $\mathcal{F}$ of facts as $\mathcal{F} = \{F(t_1, \ldots, t_n) |\ F \in \Sigma_{Facts}, F \text{ is of arity } n|\}$. The set $\mathcal{F}$ of facts is partitioned into linear and persistent facts. Derived from linear logic, linear facts model resources that can only be consumed once, whereas persistent facts model inexhaustible resources that can be consumed arbitrarily often, in particular, the knowledge of the adversary. Persistent symbols are prefixed with ! .

Example 18. The following facts are predefined in Tamarin with the following meaning:

- $Fr(x : fr)$ to generate of fresh messages x.
- $Out(m : msg)$ to send a message m.
- $In(m : msg)$ to receive a message m.

- $!K(m : msg)$ when a message m is known by the adversary.

Note that we can write $t : type$ to emphasize that t is of sort $type$. For simplicity, Tamarin also uses the notations $\sim x$ and $\$A$ to denote that x is fresh and A is public.

Each role of the protocol uses its set of facts, called state facts. They are usually of the form $St\_R\_s(x_1, x_2, \ldots, x_n)$ to denote that, at step s, an agent with the role R has the knowledge of $x_1, x_2, \ldots, x_n$.

Example 19. Consider a step where:

- agent A in the role I knows its identity A, the one of its correspondent B a symmetric key k, a nonce Na and sends $senc(Na, k)$,
- agent B in the role R knows identities B and A and the key k
- and the adversary knows k

Then we should have the state:

$$\{St\_I\_1(A, B, k, Na), St\_R\_1(B, A, k\}, Out(senc(Na, k)), !K(k)\}$$

We model possible transitions of our system as labeled multiset rewriting rules.

Definition 2.15. A labeled multiset rewriting rule is a triple (l, a, r) where l, a and r are sequences over $\mathcal{F}$ and is denoted by $l -[a]\!\!\rightarrow r$ or with the inference rule notation:

$$\frac{l_1 \ldots l_k}{r_1 \ldots r_n}[a_1 \ldots a_m]$$

For $ru = l -[a]\!\!\rightarrow r$, we define the premises as $prems(ru) = l$, the actions as $acts(ru) = a$, the conclusions as $concs(ru) = r$, and the sequence of fact symbols occurring in ru with $fsyms(ru)$.

Example 20. For fresh name generation, we define the rule: $Fresh = -[]\!\!\rightarrow [Fr(x : fr)]$

For an adversary who knows $senc(Na, k)$ and k, we have the multiset rewriting rule:

$$sdec = [!K(senc(Na, k)), \quad !K(k)] -[]\!\!\rightarrow [!K(Na)]$$

which is derived from the rewriting rule: $sdec(senc(Na, k), k) \rightarrow Na$.

For an agent A in the role of an initiator I that knows a symmetric key k and receives the cyphertext $senc(m, k)$ which A can decrypt and then send m on the network, a protocol rule can be:

$$[St\_I\_1(A, k), In(senc(m, k))] -[Dec()]\!\!\rightarrow [St\_I\_2(A, k, m), Out(m)]$$

Definition 2.16. The labeled transition relation step $steps(R)$ of the rewriting multiset system R is the set of triples $(S, l -[a]\!\!\rightarrow r, S')$ where

- S and S' are multisets of ground facts,
- $l -[a]\!\!\rightarrow r$ is a ground instance of a rule from R or of the Fresh rule,
- the multiset of linear facts in l is included in S,

- the set of persistent facts in l is included in S,
- the successor state S' is obtained from S by removing the linear facts in l and adding the facts in r.

Finally we can model an execution of a protocol under the control of an adversary:

Definition 2.17. An execution of R is an alternating sequence

$$e = [S_0, (l_1 -[a_1]\!\!\rightarrow r_1), S_1, \ldots, S_{k-1}, (l_k -[a_k]\!\!\rightarrow r_k), S_k]$$

of states and multiset rewriting rule instances such that the following conditions hold:

E1 . $S_0 = \emptyset$ (in the sense of multisets),

E2 . For all $i \in 1, \ldots, k, (S_{i-1}, (l_i -[a_i]\!\!\rightarrow r_i), S_i) \in \mathit{steps}(R)$,

E3 . For all $i, j \in 1, \ldots, k$ and $n \in FN$ where $(l_i -[a_i]\!\!\rightarrow r_i) = (l_j -[a_j]\!\!\rightarrow r_j) = ([] -[]\!\!\rightarrow Fr(n))$, it holds that $i = j$.

We denote the set of executions of R with $\mathit{execs}(R)$. We define the trace of such an execution e as $\mathit{trace}(e) = [\mathit{set}(a_1), \ldots, \mathit{set}(a_k)]$, i.e., the trace is the sequence of sets of actions of the multiset rewriting rule instances. We define the observable trace tr of a trace tr as the subsequence of all actions in tr that are not equal to $\emptyset$.

$E1$ and $E2$ ensure that an execution starts with the empty multiset and each step is valid. $E3$ ensures that the same fresh name is never generated twice.

2.4.3 Protocol/Adversary rules

We divide the multiset rewriting system R into two disjoint multisets, the first one P which contains rules from the protocol, the second one MD for message deduction which contains the possibilities of the adversary.

Definition 2.18. A protocol rule is a multiset rewriting rule $l -[a]\!\!\rightarrow r$ such that:

P1 . l, a, and r do not contain fresh names,

P2 . l does not contain K and Out facts,

P3 . r does not contain K, In, and Fr facts,

P4 . the argument of a Fr fact is always of sort fr,

P5 . $l -[a]\!\!\rightarrow r$ satisfies (a) $\mathit{vars}(r) \subseteq \mathit{vars}(l) \cup \mathcal{V}_{pub}$ and (b) l only contains irreducible function symbols from Σ_{ST} or it is an instance of a rule that satisfies (a) and (b).

A protocol is a finite set of protocol rules.

Conditions $P1$ to $P4$ ensure the smooth functioning of transitions while $P5$ ensures that (a) no free variable appears in a conclusion rule except for public one and (b) that no variable disappears by rewriting.

Example 21. Two examples of rules contradicting $P5$: $[In(x)] -[]\rightarrow [Out(y)]$ because of (a) and $[In(fst(\langle x, y\rangle))] -[]\rightarrow [Out(y)]$, which is in fact the same, because of (b) since $fst()$ is reducible in $\mathcal{R}_{ST}$.

Definition 2.19. The set of message deduction rules is defined in inference notations as follows:

$$MD = \left\{ \begin{array}{llll} \dfrac{Out(x)}{K(x)} & \dfrac{K(x)}{In(x)}[K(x)] & \dfrac{Fr(x:fr)}{K(x:fr)} & \dfrac{}{K(x:pub)} \\ \dfrac{K(x_1)\ldots K(x_k)}{K(f(x_1,\ldots,x_k))} \text{ for all } f \in \Sigma_{\mathcal{ST}} & & & \end{array} \right\}$$

They describe from left to right then down that an adversary can

- learn every sent message,
- send every message it knows,
- generate fresh names,
- know every public name, and
- construct messages based on the signature functions and messages it already knows.

We can notice that the position of Out and In facts in protocol rules and in message deduction rules enforces that every message sent or received via the public network used, goes through the knowledge of the adversary.

Example 22. As a basic example, consider a protocol where agent A sends a freshly generated nonce m on the network and gets it back. We have

$$P_{basic} = \left\{ \frac{Fr(m:fr)}{St(A:pub,m) \quad Out(m:fr)}[Start()], \frac{St(A,m) \quad In(m)}{}[End()] \right\}$$

We have the following execution:

We can see that, first, the fresh name m is created, then it is consumed by the rule $ri.2$ that generates the trace fact $Start()$, the state fact $St\_A(A,m)$ and the $Out(m)$ fact to denote that agent A knowns m, and publishes it on the network. The rule $ri.3$ consumes the $Out(m)$ fact and generates the persistent fact $!K(m)$. It is then used, but not consumed, by $ri.4$ that creates both of the action fact $K(m)$ and the $In(m)$ fact. At this stage, we have the state $S_4 = \{St\_A(A,m), In(m), !K(m)\}$, so the rule at $ri.5$ can consume the two first steps of the multiset to lead to the action fact $End()$ and to the last state $S_5 = \{!K(m)\}$. The trace tr of this execution is $tr = \{Start(),\ K(m),\ End()\}$.

Remark. Notice that Figure 2 is an execution of the same multiset rewriting system. The fact that it is incomplete does not exclude it from the set of executions.

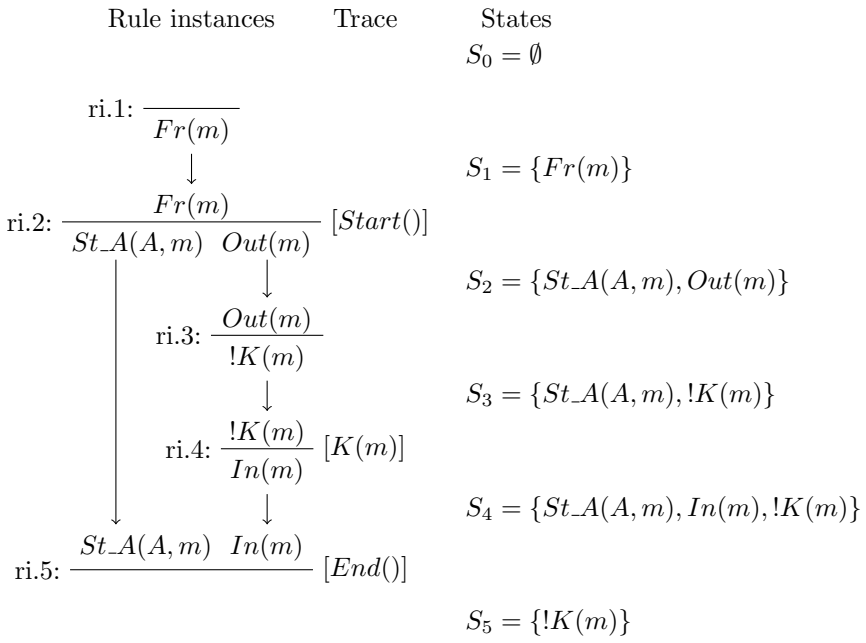


Figure 1: Example of execution of $(P_{basic} \cup MD)$ (arrows denote causal dependencies)

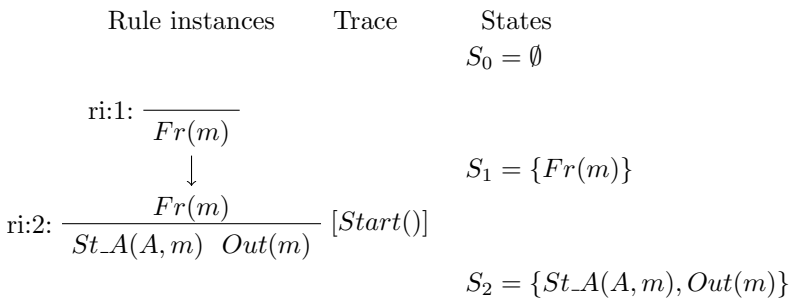


Figure 2: Other execution of $(P_{basic} \cup MD)$

Dependency graph

As we can see, the sequence of states can be deduced from the rule instances. To simplify the representation of a protocol execution, we only keep a graph over rule instances subject to some conditions that make the graph equivalent to an execution in a sense that will be defined later. To define those conditions, we first give the structure of such a graph.

Definition 2.20. From the sequence I of rule instances, consider the rule I_i, its sequence of conclusions $concs(I_i)$ and its sequence of premises $prems(I_i)$. For the graph over I, i is a node, (i, u), where $concs(I_i)_u$ is a conclusion fact, denotes a conclusion of the graph, and (i, v), where $prems(I_i)_v$ is a premise fact, denotes a premise of the graph. For a conclusion (i, u) and a premise (j, v), we denote the edge from (i, u) to (j, v) by $(i, u) \rightarrowtail (j, v)$

Example 23. In Figure 1 we have, among others, $(2,1) \rightarrowtail (5,1)$ and $(4,1) \rightarrowtail (5,2)$.

Definition 2.21. We say that the pair $dg = (I, D)$ is a dependency graph modulo ε for R if I is a sequence of ε-ground instances of rules from $R \cup Fresh$, $D \in \mathbb{N}^2 \times \mathbb{N}^2$, and dg satisfies the conditions:

DG1 . For every edge $(i,u) \rightarrowtail (j,v) \in D$ it holds that $i \leq j$ and the conclusion fact of (i,u) is syntactically equal to the premise fact of (j,v).

DG2 . Every premise of dg has exactly one incoming edge.

DG3 . Every linear conclusion of dg has at most one outgoing edge.

DG4 . The Fresh instances are unique.

We denote the set of all dependency graphs modulo ε for R by $dgraphs_\varepsilon(R)$

We say that the executions in the labeled transition system and the dependency graph are equivalent (or more precisely trace equivalent).

Lemma 2.1. *For all sets P of protocol rules,*

$$trace(execs(P \cup MD)) = trace(dgraphs_\varepsilon(P \cup MD))$$

.

Proof. We prove by induction that the sequences of rule instances of executions in labeled transition system and of dependency graphs coincide. □

Registering keys

If we want to approach cryptographic protocols with asymmetric primitives, we have to consider a protocol rule that generates and distributes keys to participants.

Consider $\Sigma_mathcalNSPK$ that contains a function symbol pk of arity 1 whose goal is to associate a secret key k with its public key $pk(k)$. In order to a associate private key and an identity, we use the persistent facts $!Ltk(A : pub,\ k : fr)$ and $!Pk(A : pub,\ x : msg)$. $!Ltk(A,\ k)$ denotes that agent A has k as its private key (Ltk stands for *Long term key*) and $!Pk(A,\ x)$ models that agent A has x as its public key. $!Pk(A, x)$ is used as $!Pk(A, pk(k))$ to link $!Ltk$ and $!Pk$ facts. They are used as persistent facts because a same agent can reuse its keys multiple times.

We can now consider the following protocol rule:

$$register\_Pk = \frac{Fr(k : fr)}{!Pk(A : pub,\ pk(k : fr))\ \ !Ltk(A : pub,\ k : fr)\ \ Out(pk(k : fr))}$$

The secret key k is modelled by a fresh name in order to be unknown and unique. Persistent facts $!Ltk$ and $!Pk$ ensure the relation with identity, public key and private key. The Out fact permits the adversary to know the public key.

From there, we can consider the NSPK protocol built on the above public key infrastructure. As a first step, we take the following multiset rewriting rule:

$$\frac{Fr(Na) \quad !Pk(B,\ pkB)}{St\_A(A,B,Na) \quad Out(aenc(\langle'1', A, Na\rangle, pkB))}$$

It means that agent A gets a fresh nonce and the public key of agent B and send to B the cyphertext of $\langle'1', A, Na\rangle$ encrypted with pkB. '1' is a tag added at the beginning of the message in order, for the recipient B to interpret the message as the first message.

For the second rule, we have:

$$\frac{Fr(Nb) \quad In(aenc(\langle'1', A, Na\rangle, pk(ltkB)) \quad !Pk(A,\ pkA) \quad !Ltk(B, ltkB)}{St\_B(B,A,Na,Nb) \quad Out(aenc(\langle'2', A, Na, Nb\rangle, pkA))}$$

Here we can better understand the need for a register_key rule : we use $pk(ltkB)$ in the In fact to denote that B can open a message addressed to him, and the link is done through the fact $!Ltk(B, ltkB)$. Let P_{NSPK} be the set of multiset rewriting rules used for the NSPK protocol, including the previous one. A dependency graph dg_0 from $dgraphs_{\mathcal{NSPK}}(P_{NSPK} \cup MD)$ is represented in Figure 3.

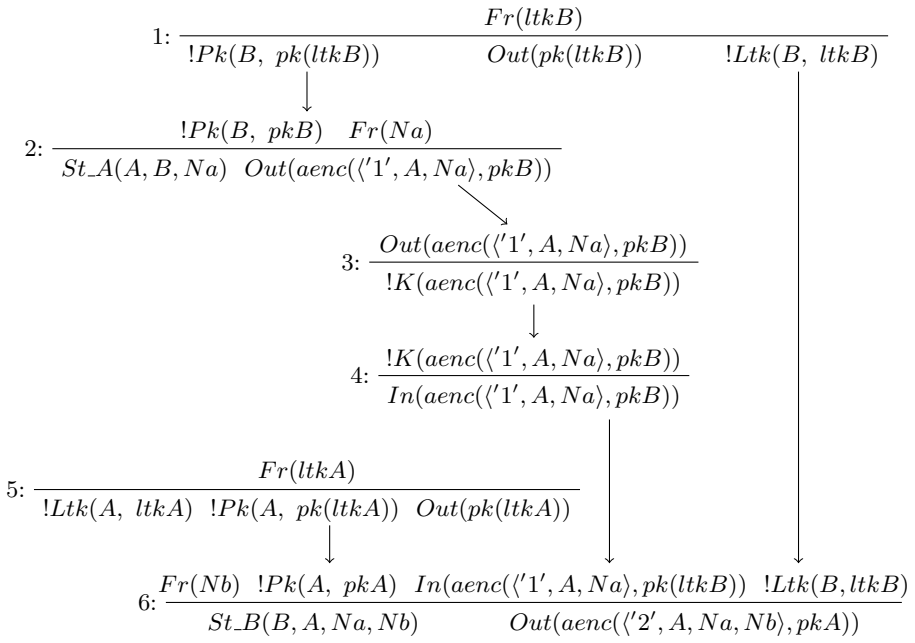


Figure 3: Representation of an element of $dgraphs_{\mathcal{NSPK}}(P_{NSPK} \cup MD)$

As we have seen before, at this step of NSPK protocol, it is possible for an adversary to impersonate B and to have knowledge of Na, but this does not deal with any rule of MD. In order to complete the adversary abilities, we add the *key_reveal* rule:

$$key\_reveal = \frac{!Ltk(A,\ k)}{Out(k)}[Rvl(A)]$$

It can both be used to corrupt an agent by making use of its private key or to impersonate an agent's identity.

If, for example, we want Tamarin to find an execution as described above, we have to consider a property that translates this condition and try to see if the property is true with respect to the protocol.

2.5 Modelling properties

A property is true with respect to the protocol if the trace of every execution of this protocol satisfies the property.

To identify the nonce Na contained in the cyphertext sent by A, we need to put trace facts in which Na appears. They are placed on the rule where A sends the cyphertext and on the rule where B receives it. We obtain these rules:

$$\frac{Fr(Na) \quad !Pk(B,\ pkB)}{St\_A(A,B,Na) \quad Out(aenc(\langle '1', A, Na\rangle, pkB))}[Send\_I(Na)]$$

and

$$\frac{Fr(Nb) \quad In(aenc(\langle '1', A, Na\rangle, pk(ltkB)) \quad !Pk(A,\ pkA) \quad !Ltk(B, ltkB)}{St\_B(B,A,Na,Nb) \quad Out(aenc(\langle '2', A, Na, Nb\rangle, pkA))}[Recv\_R(Na)]$$

We want to check whether if an adversary can know Na, which is represented by the event fact $K(Na)$. To do so, we specify a property ϕ_0 that states that for all executions where A sends Na to B, the event fact $K(Na)$ does not hold.

A execution where A sends Na is specified by the existence of a time point i where the event $Send\_I(Na)$ occurs. To take into account the time points in an execution, we introduce the sort *temp*. Informally, ϕ_0 say: *For all executions where $Send\_I(Na)$ occurs at a time point i and $Recv\_R(Na)$ occurs at a time point j, there is no time point k where $K(U)$ occurs.*

We can now rewrite ϕ_0 in first-order logic language (where we write $f@i$ to denote that the event f occurs at the time point i):

$$\forall\ i,\ j,\ Na, (Send\_I(Na)@i\ \wedge\ Recv\_R(Na)@j \Rightarrow \neg(\exists k, K(Na)@k))$$

To prove that the property ϕ_0 is true in the context of the multiset rewriting P, Tamarin will construct every execution of the protocol and verify if one of them has a trace in contradiction with ϕ_0. If Tamarin doesn't find any counter-example, we say that P satisfies ϕ_0, written $P \models \phi_0$.

To formulate a counter-example, we use the equivalence between $\forall x, \phi(x)$ and $\neg(\exists x, \neg\ \phi(x))$

Thus Tamarin will search for an execution where:

$$\exists\ i,\ j,\ k,\ Na, (Send\_I(Na)@i\ \wedge\ Recv\_R(Na)@j\ \wedge\ K(Na)@k)$$

Since we add the *key_reveal* rule, it easily finds the attack represented in Figure 4

This execution is a counter-example of the property. Indeed, it is an execution where we have all three facts $Send\_I(Na)$, $Recv\_R(Na)$ and $K(NA)$. This attack represents the corruption of agent B whose secret key has be revealed. It is not the "man-in-the-middle" attack we have seen before. To prevent from corruption of agent A or B, we can both add the properties that there are no *key_reveal* of those agents, or claim them as "honest" by an action fact and forbid corruption of honest agents.

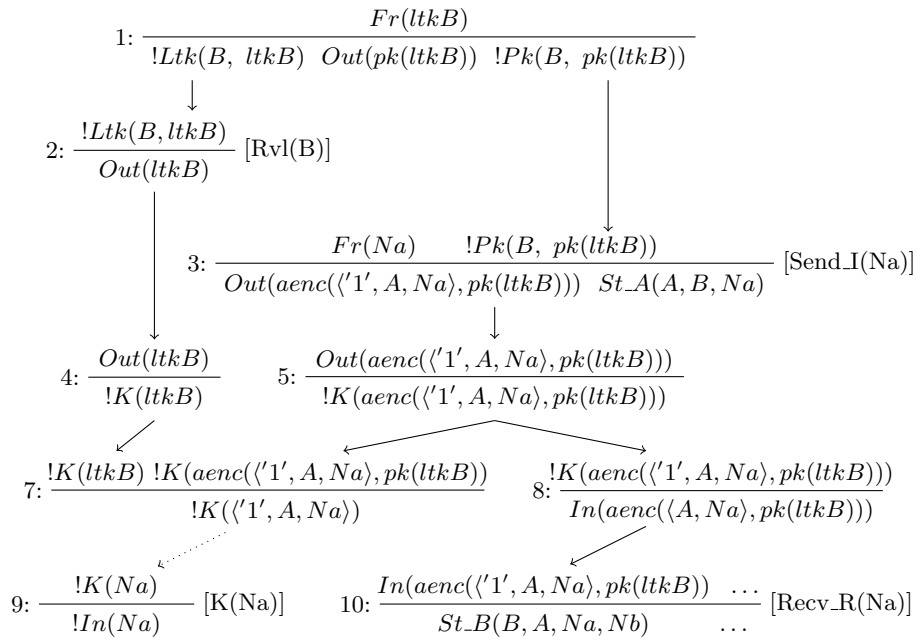


Figure 4: Dependency graph from $dgraphs_{\mathcal{NSPK}}(P_{NSPK} \cup MD)$ not satisfying ϕ_0 (dotted arrows denote hidden obvious rules)

We take the first method as example. To take into account their identities, we adapt the *Send_I* and *Recv_R* facts and make use of the *Rvl* fact:

We obtain the new ϕ_0:

$$\forall\ A,\ B,\ i,\ j,\ Na,\ \ Send\_I(A, Na)@i\ \wedge\ Recv\_R(B, Na)@j$$
$$\Rightarrow (\ \neg(\exists\ k, K(Na)@k)\ \vee (\exists\ l_1, Rvl(A)@l_1)\ \vee\ (\exists\ l_2, Rvl(B)@l_2)\ )$$

We get the expected attack represented in Figure 5

This graph represents the first step of the "man-in-the-middle" attack. At this step, neither A nor B can know that their communication is totally controlled by the adversary M which has given its public key instead of B's. If the conversation is continued this way, the authentication won't be satisfied.

In what follows, we show all the rules used to model $NSPK$, a complete trace formula for secrecy property and a first view of the representation of the attack in Tamarin.

2.6 Example: NSPK

We model $NSPK$ protocol:

$$A \to B:\ \{'1', A, Na\}_{pk(B)}$$
$$B \to A:\ \{'2', Na, Nb\}_{pk(A)}$$
$$A \to B:\ \{'3', Nb\}_{pk(B)}$$

designed, in theory, to ensure authentication between A and B.

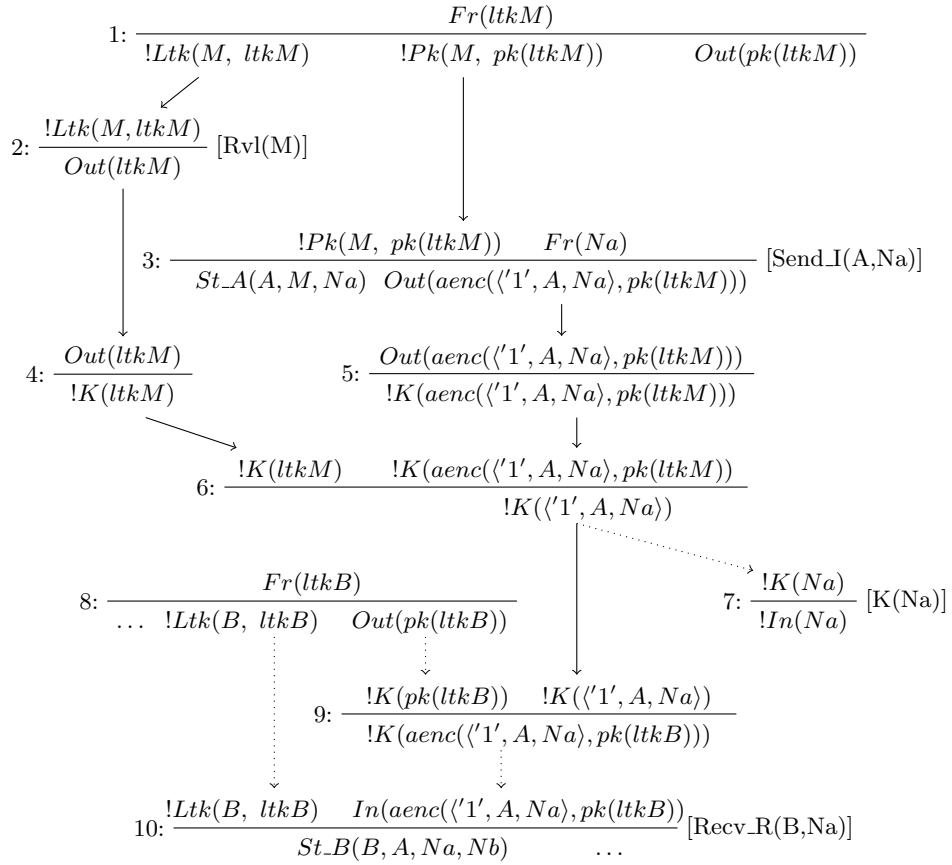


Figure 5: Dependency graph from $dgraphs_{\mathcal{NSPK}}(P_{NSPK} \cup MD)$ not satisfying ϕ_0

The adversary is given the set MD_{NSPK} of message deduction rules:

$$MD_{NSPK} = \left\{ \begin{array}{cccc} \frac{Out(x)}{K(x)} & \frac{K(x)}{In(x)}[K(x)] & \frac{Fr(x:fr)}{K(x:fr)} & \frac{}{K(x:pub)} \\ \frac{K(x)\ \ K(y)}{K(\langle x, y\rangle)} & \frac{K(p)}{K(fst(p))} & \frac{K(p)}{K(snd(p))} & \\ \frac{K(m)\ \ K(k)}{K(aenc(m,k))} & \frac{K(c)\ \ K(k)}{K(adec(c,k))} & \frac{K(k)}{K(pk(k))} & \end{array} \right\}$$

The protocol is described by the set P_{NSPK} of protocol rules:

$$\frac{Fr(Na)\ \ !Pk(B,\ pkB)}{St\_A(A, B, Na)\ \ Out(aenc(\langle'1', A, Na\rangle, pkB))}$$

$$\frac{Fr(Nb)\ \ In(aenc(\langle'1', A, Na\rangle, pk(ltkB))\ \ !Pk(A, pkA)\ \ !Ltk(B, ltkB)}{St\_B(B, A, Na, Nb)\ \ Out(aenc(\langle'2', Na, Nb\rangle, pkA))}$$

$$\frac{St\_A(A,B,Na)\ \ In(aenc(\langle'2',Na,Nb\rangle,pk(ltkA))\ \ !Pk(B,pkB)\ \ !Ltk(A,ltkA)}{Secret(A,B,Na)\ \ Secret(A,B,Nb)\ \ Out(aenc(\langle'3',Nb\rangle,pkB))}$$

$$\frac{St\_B(B,A,Na,Nb)\ \ In(aenc(\langle'3',Nb\rangle,pkB))\ \ !Pk(A,pkA)\ \ !Ltk(B,ltkB)}{Secret(B,A,Na)\ \ Secret(B,A,Nb)}$$

States of the form $Secret(A,B,m)$ are placed at the end of each role to denote that, A claims to share the message m with B and considers it as secret. We use the labeled multiset rewriting rule:

$$\frac{Secret(A,B,m)}{}[Secret(A,B,m)]$$

to use it in the property specification.

The previously defined *register_key* and *key_reveal* rules are added in the set of protocol rules.

Authentication is ensured by the fact that the nonces Na and Nb are supposed secret. Thus, we want to prove that nobody claims to have set up a shared secret, but the adversary knows it without having performed a long-term key reveal. In first-order logic, we can write it this way:

$$\begin{aligned}\neg\ (\exists\ A\ B\ s\ i.\ & secret(A,B,s)@i\\ & \&\ (\exists j.\ K(s)@j)\\ & \&\ \neg(\exists r.\ RevLtk(A)@r)\\ & \&\ \neg(\exists r.\ RevLtk(B)@r)\ )\end{aligned}$$

Tamarin will now try to find a trace where A and B think they share a secret but the adversary managed to knows it without corrupting neither A nor B.

Such a trace exists, it is represented by Tamarin as in Figure 6 and called the "man-in-the-middle" attack.

2.7 Dependency graphs modulo $\mathcal{AX}$

In order to generalise our results to equational theories with, among others, non-orientable equations like associativity or commutativity, we consider an equational theory $\mathcal{NO}$ such that $(\Sigma_{\mathcal{NO}}, \mathcal{R}_{NO}, \mathcal{AX})$ is a decomposition of $\mathcal{NO}$ with the finite variant property.

Example 24. $\mathcal{NO}$ can stand for the equational theory $\mathcal{DH}$ used for Diffie-Hellman cryptographic primitives. A decomposition of $\mathcal{DH}$ with the finite variant property is $(\Sigma_{\mathcal{DH}}, \mathcal{R}_{\mathcal{DH}}, \mathcal{AC})$ where:

$$\Sigma_{\mathcal{DH}} = \{\_\hat{}\_, \_*\_, 1, \_^{-1}\}$$

$$\mathcal{AC} = \{x*(y*z) \simeq (x*y)*z,\ x*y \simeq y*x\}$$

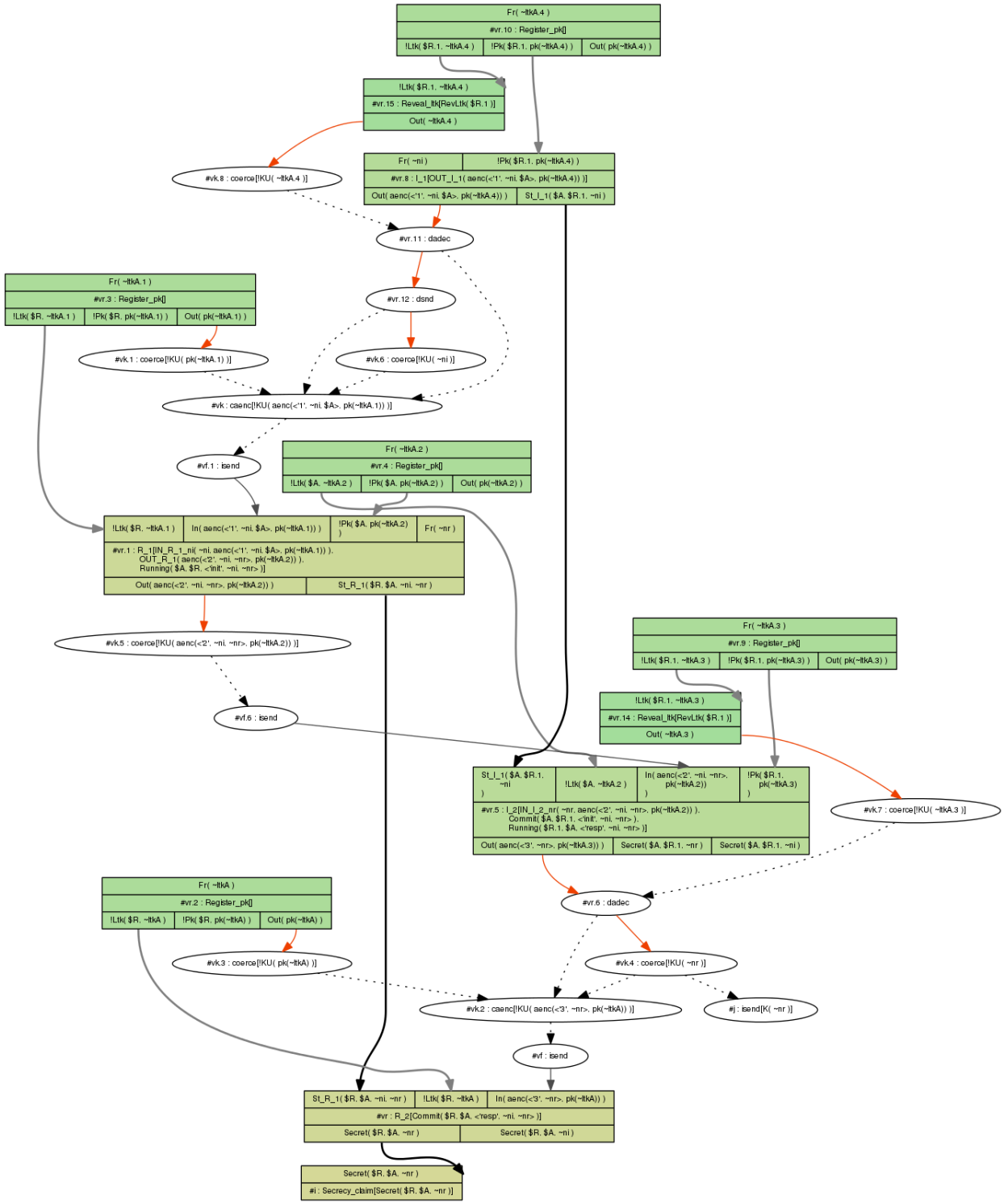


Figure 6: Man-in-the-middle attack on NSPK

$$\mathcal{R}_{\mathcal{DH}} = \left\{ \begin{array}{ccc} (x\hat{}y)\hat{}z \to x\hat{}(y*z) & & (x^{-1}*y)^{-1} \to x*y^{-1} \\ x^{-1}*(x*y)^{-1} \to y^{-1} & & x^{-1}*(y^{-1}*z) \to (x*y)^{-1}*z \\ (x*y)^{-1}*(y*z) \to (x^{-1}*y) & & x^{-1}*y^{-1} \to (x*y)^{-1} \\ 1^{-1} \to 1 & x*1 \to x & (x^{-1})^{-1} \to x \\ x*(x^{-1}*y) \to y & x\hat{}1 \to x & x*x^{-1} \to 1 \end{array} \right\}$$

To simplify reasoning about products and ensure that multiplication is not directly used, we add condition $P6$ to protocol rules, namely:

P6. a conclusion does not contain the function symbol $*$.

Thus, we consider $\Sigma_{\mathcal{NO}_e} = \Sigma_{\mathcal{NO}} \cup \Sigma_{\mathcal{ST}}$, $\mathcal{R}_{\mathcal{NO}_e} = \mathcal{R}^{\simeq}_{\mathcal{NO}} \cup \mathcal{R}^{\simeq}_{\mathcal{ST}}$ and the equational theory $\mathcal{NO}_e = (\Sigma_{\mathcal{NO}_e}, \mathcal{R}^{\simeq}_{\mathcal{NO}_e} \cup \mathcal{AX})$ for which $(\Sigma_{\mathcal{NO}_e}, \mathcal{R}_{\mathcal{NO}_e}, \mathcal{AX})$ is a decomposition with the finite variant property. Thanks to this property, we can compute, for a given term t, the set $\lceil t \rceil^{\mathcal{R}_{\mathcal{NO}_e}}_{substs}$ of $\mathcal{R}_{\mathcal{NO}_e}, \mathcal{AX}$-variants using a folding variant narrowing algorithm. Then we denote by $\lceil t \rceil^{\mathcal{R}_{\mathcal{NO}_e}}_{insts}$, the set $\{(t\sigma)\downarrow_{\mathcal{R}_{\mathcal{NO}_e}} \mid \sigma \in \lceil t \rceil^{\mathcal{R}_{\mathcal{NO}_e}}_{substs}\}$ of normalized instances corresponding to the variants. We say that t is $\downarrow_{\mathcal{R}_{\mathcal{NO}_e}}$-normal if $t =_{\mathcal{AX}} t\downarrow_{\mathcal{R}_{\mathcal{NO}_e}}$. We say a dependency graph $dg = (I, D)$ is $\downarrow_{\mathcal{R}_{\mathcal{NO}_e}}$-normal if all rule instances in I are.

Tamarin uses a folding variant narrowing algorithm on multiset rewriting rules to calculate their variants.

Example 25. $[K(x),\ K(y)] -[]\to [K(x\hat{}y)]$ has 47 $\mathcal{R}_{\mathcal{DH}_e}, \mathcal{AC}$-variants. For example we have

$$\left\{ \begin{array}{l} [K(x),\ K(y)] -[]\to [K(x\hat{}y)], \\ [K(u\hat{}v),\ K(v^{-1})] -[]\to [K(u)] \end{array} \right\} \subset \lceil\ [K(x),\ K(y)] -[]\to [K(x\hat{}y)]\ \rceil^{\mathcal{R}_{\mathcal{DH}_e}}_{insts}$$

The following lemma shows that we can use dependency graphs modulo $\mathcal{AX}$.

Lemma 2.2. *For all sets R of multiset rewriting rules,*

$$dgraphs_{\mathcal{NO}_e}(R)\downarrow_{\mathcal{R}_{\mathcal{NO}_e}} =_{\mathcal{AX}} \{dg \mid\ dg \in dgraphs_{\mathcal{AX}}(\lceil R \rceil^{\mathcal{R}_{\mathcal{NO}_e}}_{insts})\ \wedge\ dg\downarrow_{\mathcal{R}_{\mathcal{NO}_e}}\text{-normal}\}.$$

3 Resolution

In this section we introduce normal dependency graphs and explain what has already been proved for a subterm convergent theory $\mathcal{ST}$. Then we adapt the proof for context subterm rules and implement them for Tamarin.

3.1 Prior works

3.1.1 Construction and deconstruction rules

We can see, with simple examples, that dependency graphs are not sufficient to automatise the search for traces.

For instance, consider the case where the adversary deduces the first element a of a pair $\langle a, b\rangle$, then pairs it with an element c, then deduces a from the new pair to finally build the pair $\langle a, d\rangle$. It is a possible, but redundant subgraph of a dependency graph. Moreover, this could continue indefinably, but could be resumed in one step.

To gain in efficiency and avoid redundancy, we divide the adversary rules into two categories, deconstruction rules and construction rules. Deconstruction rules are used by the adversary just after protocol rules and made to deduce messages from what has been sent on the network. Construction rules are, conversely, used to build messages from the knowledge of the adversary and then send to them on the network. To achieve this, we provide K facts with an orientation "Up and Down" denoted $K^{\Uparrow}$ and $K^{\Downarrow}$. Deconstruction rules have premises with both $K^{\Downarrow}$ and $K^{\Uparrow}$ facts and conclusion with a $K^{\Downarrow}$ fact, while construction rules have premises with only $K^{\Uparrow}$ facts and conclusion with a $K^{\Uparrow}$ fact. This enforces deconstruction rules to be locally before construction rules. The transition from $K^{\Downarrow}$ to $K^{\Uparrow}$ is achieved by a special rule called "Coerce".

In the context of a subterm convergent theory $\mathcal{ST}$, the idea is to consider a construction rule for every function in $\Sigma_{\mathcal{ST}}$, and deconstruction rules for each rewriting rule. The process for deconstruction rules is not trivial and will be explained later. Additionally we consider construction rules for fresh and public name generation, and, to be coherent with the purpose of construction and deconstruction rules, we state that the $Out()$ rule has a $K^{\Downarrow}$ fact as conclusion while the $In()$ rule has a $K^{\Uparrow}$ facts as premise.

Example 26. If we consider theory from $\mathcal{NSPK}$, as defined previously, for subterm convergent theory, we get the following set of rules $ND_{\mathcal{NSPK}}$ (for normal message deduction):

$$ND_{\mathcal{NSPK}} = \left\{ \begin{array}{ccc} & \dfrac{Out(x)}{K^{\Downarrow}(x)} & \\ \dfrac{K^{\Downarrow}(\langle x,y\rangle)}{K^{\Downarrow}(x)} \quad \dfrac{K^{\Downarrow}(\langle x,y\rangle)}{K^{\Downarrow}(y)} & \dfrac{K^{\Downarrow}(aenc(m,pk(k))) \quad K^{\Uparrow}(k)}{K^{\Downarrow}(m)} & \\ \dfrac{K^{\Downarrow}(x)}{K^{\Uparrow}(x)} & \dfrac{Fr(x:fr)}{K^{\Uparrow}(x:fr)} & \dfrac{}{K^{\Uparrow}(x:pub)} \\ \dfrac{K^{\Uparrow}(x) \quad K^{\Uparrow}(y)}{K^{\Uparrow}(\langle x,y\rangle)} & \dfrac{K^{\Uparrow}(p)}{K^{\Uparrow}(fst(p))} & \dfrac{K^{\Uparrow}(p)}{K^{\Uparrow}(snd(p))} \\ \dfrac{K^{\Uparrow}(m) \quad K^{\Uparrow}(k)}{K^{\Uparrow}(aenc(m,k))} & \dfrac{K^{\Uparrow}(c) \quad K^{\Uparrow}(k)}{K^{\Uparrow}(adec(c,k))} & \dfrac{K^{\Uparrow}(k)}{K^{\Uparrow}(pk(k))} \\ & \dfrac{K^{\Uparrow}(x)}{In(x)}[K(x)] & \end{array} \right\}$$

The rules are arranged more or less following the way they can be used between protocol rules. We can notice that the deconstruction rule for decryption has $K^{\Uparrow}$ and $K^{\Downarrow}$ facts in premises.

With such rules, the adversary avoids cases of redundancy as shown in Figure 7

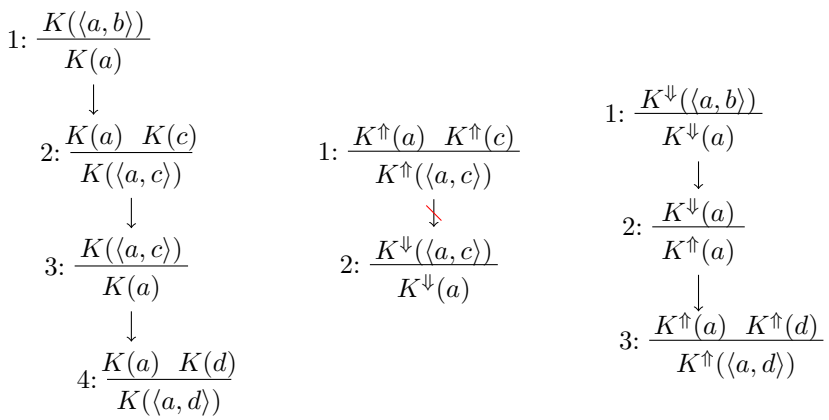


Figure 7: Message deduction subgraphs for pairing. The first one represents a redundant dependency subgraph, the second one an impossible deduction with Up and Down K-facts and the last one, a correct deduction and construction from pairing

Consider now that we have a subterm convergent rewriting system $\mathcal{ST}_0 = (\{1, a(\_,\_),\ b(\_,\_),\ c(\_,\_)\}, \{a(b(c(x,y),1),y) \to x\})$. We can notice first that, given $b(c(x,y),1)$ and y we can deduce x by applying the function $a(\_,\_)$, but, given $c(x,y)$, 1 which is ground, and y, we can also deduce x by applying successively functions b and a. We can deduce there should be at least two

deconstruction rules for the rewriting rule. A second problem is to know where to place the $K^{\Downarrow}$ and $K^{\Uparrow}$ facts in the premises.

For a subterm convergent rewriting system, a method to compute deconstruction rules is the following. Consider a subterm rewriting rule $l \to r$ where r is not ground. Since it is a subterm rewriting rule, there is a position p in l such that $l|_p = r$. Then, for each position $p' \neq []$ strictly above p, we compute a deconstruction rule for which the term $l|_{p'}$ is in a $K^{\Downarrow}$ fact and the terms $l|_{\tilde{p}}$, where $\tilde{p}$ has a sibling equal or above p', are used in a $K^{\Uparrow}$ fact.

Example 27. Consider the rewriting rule $l \to r = a(b(c(x, y), 1), y) \to x$, the only position p of l such that $l|_p = r$ is $[1, 1, 1]$, there are two positions different than $[]$ and strictly above p, $p'_1 = [1, 1]$ and $p'_2 = [1]$. For p'_1, we have $\tilde{p}_1 = [2]$ and $\tilde{p}_2 = [1, 2]$ as positions which have a sibling above or equal to p'_1. For p'_2, we have only $\tilde{p}_1 = [2]$ as positions which have a sibling above or equal to p'_2. We represent this in Figure 8:

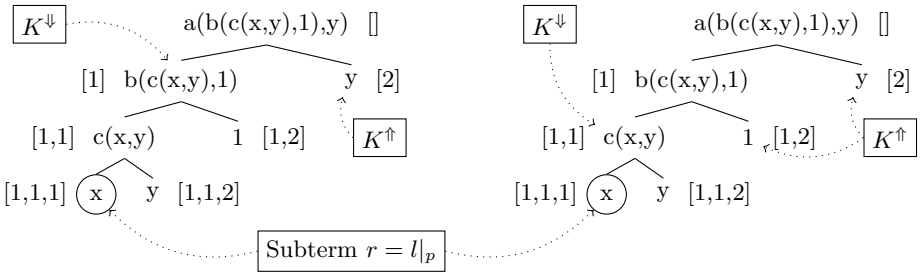


Figure 8: Different possible positions of K-facts for deconstruction rules associated with $a(b(c(x, y), 1), y) \to x$.

Thus, the two associated deconstruction rules are:

$$[K^{\Downarrow}(b(c(x, y), 1)),\ K^{\Uparrow}(y)] \text{ –}[]\rightarrow [K^{\Downarrow}(x)]$$

and

$$[K^{\Downarrow}(c(x, y)),\ K^{\Uparrow}(1),\ K^{\Uparrow}(y)] \text{ –}[]\rightarrow [K^{\Downarrow}(x)]$$

More formally, for each position p such that $l|_p = r$, we use the function *ctxtdrules* drawn from [16] to compute the corresponding deconstruction rules:

$$ctxtdrules(l, p) =$$
$$\{[K^{\Downarrow^d}(l|_{p'})] \cdot cprems(l, p') \text{ –}[]\rightarrow [K^{\Downarrow^d}(l|_p)] |\ p' \text{ strictly above } p \text{ and } p' \neq []\}$$

where $cprems(l, p')$ determines the sequence of $K^{\Uparrow}$ premises:

$$cprems(l, p') = seq(\{K^{\Uparrow}(l|_{\tilde{p}}) |\ \tilde{p} \neq [] \wedge \tilde{p} \text{ has a sibling above or equal to } p'\})$$

(We use *seq* to convert sets into sequences.)

We can notice that deconstruction rules from $ND_{\mathcal{NSPK}}$ are coherent with this construction.

Normal message deduction for $\mathcal{NO}$ theories

In order to be more general, we extend construction and deconstruction rules to non-orientable theories. Since such theories are not the main subject of the report, we only apply an instance already studied in [16] of non-orientable theory, namely, the bilinear-pairing theory $\mathcal{BP}$. It is an extension of $\mathcal{DH}$ used for the Joux Protocol, including in particular multisets.

To be brief:

$$\mathcal{BP} = (\Sigma_{\mathcal{BP}}, \mathcal{R}^{\simeq}_{\mathcal{BP}} \cup \mathcal{ACC})$$

where

$$\Sigma_{\mathcal{BP}} = \{\hat{e}(\_,\_), [\_]\_, \_\sharp\_\} \cup \Sigma_{\mathcal{DH}}$$

$$\mathcal{ACC} = \{x\sharp(y\sharp z) \simeq (x\sharp y)\sharp z,\ x\sharp y \simeq y\sharp x,\ \hat{e}(x,y) \simeq \hat{e}(y,x)\} \cup \mathcal{AC}$$

$$\mathcal{R}_{\mathcal{BP}} = \{[z]([y]x) \to [z*y]x,\ [1]x \to x,\ \hat{e}([y]x, z) \to \hat{e}(x,z)\hat{}y\} \cup \mathcal{R}_{\mathcal{DH}}$$

and then, $\mathcal{R}_{\mathcal{BP}_e} = \mathcal{R}_{\mathcal{DH}} \cup \mathcal{R}_{\mathcal{ST}}$ such that $(\Sigma_{\mathcal{BP}_e}, \mathcal{R}_{\mathcal{BP}_e}, \mathcal{ACC})$ is a decomposition of $\mathcal{BP}_e$ with the finite variant property. The associated set of normal message deduction rules $ND_{\mathcal{BP}}$ is described in [16]. It introduces two new symbols: $K^{\Downarrow d}$ and $K^{\Downarrow e}$ in the place of $K^{\Downarrow}$ symbol, and rules never have $K^{\Downarrow e}$-facts as premise.

This is done to avoid redundancy as in the following example:

$$1:\ \frac{K^{\Downarrow d}(a\hat{}b) \quad K^{\Uparrow}(b^{-1}*c)}{K^{\Downarrow e}(a\hat{}c)}$$

$$\not\downarrow$$

$$2:\ \frac{K^{\Downarrow d}(a\hat{}c) \quad K^{\Uparrow}(c^{-1}*d)}{K^{\Downarrow e}(a\hat{}d)}$$

which could have been possible without distinction between $K^{\Downarrow d}$-facts and $K^{\Downarrow e}$-facts.

We extend this notation to $ND_{\mathcal{ST}}$ by substitute each occurrence of $K^{\Downarrow}$ by $K^{\Downarrow d}$ except for the coerce rule that can both take in premise $K^{\Downarrow d}$ and $K^{\Downarrow e}$.

The rest of the report is done with $\mathcal{NO} = \mathcal{BP}$.

We introduce the notion of invertible function symbols and non inverse factors as:

Definition 3.1. A function symbol $f \in \Sigma_{\mathcal{BP}} \cup \Sigma_{\mathcal{ST}}$ of arity n is invertible if for arbitrary terms t_i and for all $1 \leq i \leq n$, $K^{\Downarrow}(f(t_1, \ldots, t_n)) \ -[]\to\ K^{\Downarrow}(t_i) \in ginsts_{\mathcal{ACC}}(ND)$.

For instance, the only invertible function symbol of Σ_{NSPK} is $\langle\_,\_\rangle$.

Definition 3.2. The set $nifactors(t)$ of non-inverse factors of a term t is defined as

$$nifactors(t) = \begin{cases} nifactors(a) \cup nifactors(a) \text{ if } t = a*b \\ nifactors(s) \text{ if } t = s^{-1} \\ \{t\} \text{ otherwise} \end{cases}$$

3.1.2 Normal Dependency graphs

We integrate now the concept of normal message deduction, with construction and deconstruction rules, to dependency graphs.

Definition 3.3. A normal dependency graph for a set of protocol rules P is a dependency graph dg such that $dg \in dgraphs_{\mathcal{ACC}}(\lceil P \rceil^{\mathcal{R}_{\mathcal{BP}_e}}_{insts} \cup ND)$ and the following conditions are satisfied:

NDG 1 . The dependency graph dg is $\downarrow_{\mathcal{R}_{\mathcal{BP}_e}}$-normal.

NDG 2 . There is no multiplication rule that has a premise fact of the form $K^{\Uparrow}(s*t)$ and all conclusion facts of the form $K^{d}(s*t)$ are conclusions of a multiplication rule.

NDG 3 . If there are two conclusions c and c' with conclusion facts $K^{d}(m)$ and $K^{d'}(m')$ such that $m =_{AC} m'$ and either $d{=}d'{=}\Uparrow$ or $d{=}\Downarrow y$ and $d'{=}\Downarrow y'$ for $y,\ y' \in \{d, e\}$, then $c = c'$.

NDG 4 . All conclusion facts $K^{\Uparrow}(f(t_1, \ldots, t_n))$ where f is an invertible function symbol are conclusions of the construction rule for f.

NDG 5 . If a node i has a conclusion $K^{\Downarrow y}(m)$ for $y \in \{d, e\}$ and a node j has a conclusion $K^{\Uparrow}(m')$ with $m =_{AC} m'$, then $i < j$ and either $root(m)$ is invertible or the node j is an instance of coerce.

NDG 6 . There is no node $[K^{\Downarrow d}(a),\ K^{\Uparrow}(b)] -[]\rightarrow K^{\Downarrow e}(c\hat{}d)$ where c does not contain any fresh names and $nifactors(d) \subseteq_{AC} nifactors(b)$.

NDG 7 . There is no construction rule for $\sharp$ that has a premise of the form $K^{\Uparrow}(s\sharp t)$ and all conclusion facts of the form $K^{\Uparrow}(s\sharp t)$ are conclusions of a construction rule for$\sharp$.

NDG 8 . The conclusion of a deconstruction rule for $\sharp$ is never of the form $K^{\Downarrow d}(s\sharp t)$.

NDG 9 . There is no node $[K^{\Downarrow d}(a), K^{\Uparrow}(b)] -[]\rightarrow K^{\Downarrow e}([d]c)$ such that c does not contain any fresh names and $nifactors(d) \subseteq_{ACC} nifactors(b)$.

NDG 10 . There is no node i labeled with $[K^{\Downarrow d}([t_1]p), K^{\Downarrow d}([t_2]q)] -[]\rightarrow K^{\Downarrow d}(\hat{e}(p,q)\hat{}c)$ such that there is a node j labeled with $[K^{\Downarrow d}(\hat{e}(p,q)\hat{}c), K^{\Uparrow}(d)] -[]\rightarrow K^{\Downarrow d}(\hat{e}(p,q)\hat{}e)$, an edge $(i,1) \rightarrowtail (j,1)$, $nifactors(ti) \subseteq_{ACC} nifactors(d)$ for $i = 1$ or $i = 2$, and $\hat{e}(p,q)$ does not contain any fresh names.

NDG 11 . There is no node $[K^{\Downarrow d}([a]p), K^{\Downarrow d}([b]q)] -[]\rightarrow K^{\Downarrow d}(\hat{e}(p,q)\hat{}(a*b))$ such that the send-nodes of the first and second premises are labeled with ru_1 and ru_2 and $fsyms(ru_2) <_{fs} fsyms(ru_1)$ where $<_{fs}$ is a total order on sequences of fact symbols.

We denote the set of all normal dependency graphs for P with $ndgraphs(P)$.

We do not explain those rules in detail, most of them ensure the coherence of the graph and avoid redundancy. NDG 2 and NDG 6 are for $\mathcal{DH}$ theory, NDG 7 to NDG 11 are for $\mathcal{BP}$. NDG 3 avoid multiple deduction of the same term. NDG 4 forbids deduction coerce rule on invertible function symbol like $\langle \_, \_ \rangle$ for example. NDG 5 forces $K^{\Downarrow d}$ facts to be before $K^{\Uparrow}$ facts.

Example 28. The graph represented in 6 is a normal dependency graph. Figure 9 is a raw representation of the first interception of the adversary, on which we can see that rules with $K^{\Downarrow}$-facts as premises are found above rules with $K^{\Uparrow}$ facts as premises.

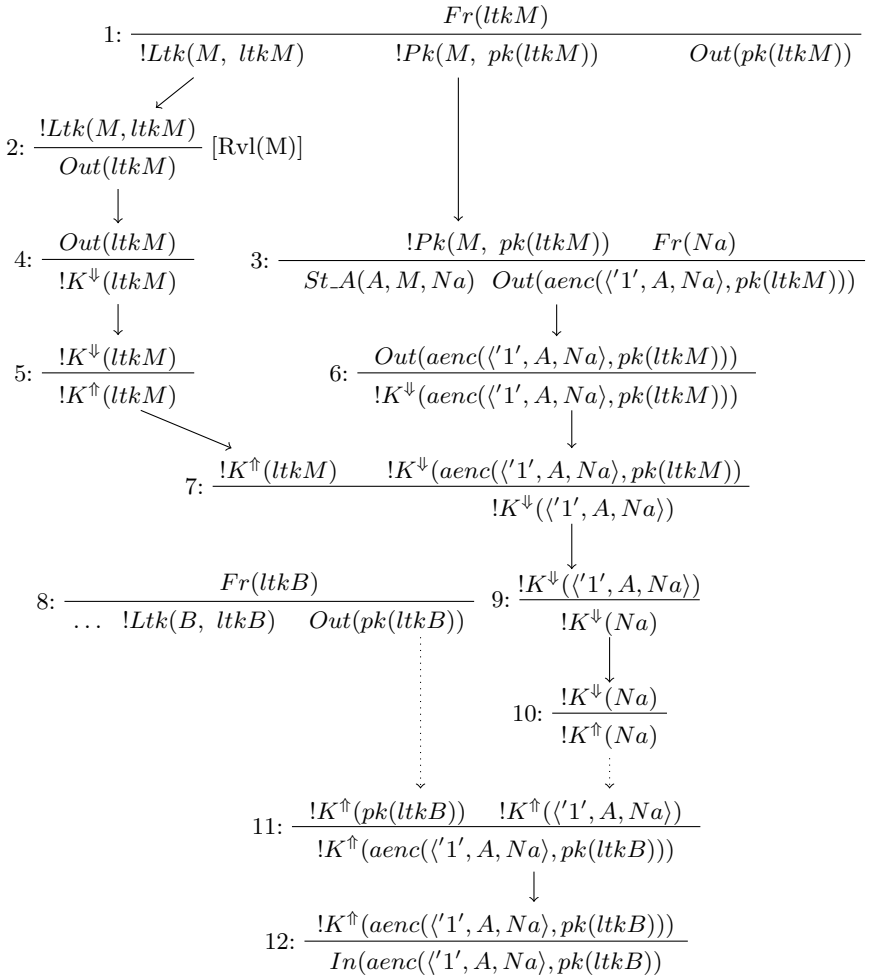


Figure 9: Subgraph of the normal dependency graph from Tamarin in Figure 6. Note that rule 9 does not exist: to get $K^{\Downarrow}(Na)$ we need in fact to apply two deconstruction rules associated with the rewriting rule $snd(\langle x, y\rangle) \to y$.

Considering such conditions for normal dependency graph and such deconstruction rules for rewriting system, we can deduce the following lemma:

Lemma 3.1. *For all protocols rules system P,*

$$\{\overline{trace(dg)} | dg \in dgraphs(\lceil P \cup MD \rceil_{insts}^{\mathcal{RBP}_e}) \wedge dg \downarrow_{RBP_e} \text{-}normal\} = \overline{trace(ndgraphs(P))}.$$

where *overlinetr* denotes the subsequence,called observable trace, of all actions in tr that are not equal to $\emptyset$.

We can find the proof of this lemma in [16], it is significantly reused to prove lemma 3.2.

Then, by applying successively lemmas 2.1, 2.2 and 3.1, we can prove the following corollary:

Corollary. *For all sets P of protocol rules,*

$$\overline{trace(execs(P \cup MD))} \downarrow_{\mathcal{R}_{\mathcal{BP}_e}} =_{\mathcal{ACC}} \overline{trace(ndgraphs(P))}.$$

This corollary says that normal dependency graphs and executions have the same observable traces.

3.2 Proof of Lemma 3.1 for context subterm rules

Now that we know that can use normal dependency graph for protocols involving a subterm convergent theory, we are going to adapt the proof of Lemma 3.1 for more general theories that are not subterm convergent namely, context subterm theories noted $\mathcal{CST}$. Such theories contain rewriting rules where the right-hand side term is not necessary a subterm of the right-hand side.

Definition 3.4. Formally a context subterm rule $l \to r$ verifies that there are k and $p_1, \ldots, p_k$ such that $r \in \mathcal{T}_{\Sigma_{\mathcal{CST}}}(l|_{p_1}, \ldots, l|_{p_k})$

As guideline for the following, we take the blind signature theory $\mathcal{BS}$ used in both the Chaum's online protocol for E-Cash and FOO or Okamoto protocols for E-Voting that we will study later.

Example 29. The blind signature permits to sign a blinded message with a secret key and then to unblind the signed blinded message to get the signed message without blinding. Materially, it is like putting a vote in an envelope that hides it, making somebody sign it with a pencil that write through the envelope and then getting back the signed vote. This primitive can be modelled this way.

$$\Sigma_{\mathcal{BS}} = \left\{ \begin{array}{cccc} \{blind(\_,\_), & unblind(\_,\_), & sign(\_,\_), & cheksign(\_,\_), \\ fst(\_), & snd(\_), & \langle \_,\_ \rangle, & pk(\_) \end{array} \right\}$$

and

$$\mathcal{R}_{\mathcal{BS}} = \left\{ \begin{array}{c} unblind(blind(m,r),r) \to m,\ checksign(sign(m,k),pk(k)) \to m, \\ unblind(sign(blind(m,r),k),r) \to sign(m,k), \\ fst(\langle x,y \rangle) \to x, \qquad snd(\langle x,y \rangle) \to y \end{array} \right\}$$

The first equation models the fact that, blinding then unblinding a message, with the same key, gives back the initial message, similar to symmetric encryption. The second permits to know a message under a signature, as the signature is not supposed to hide the message. The third one has already been explained, note that it is not a subterm rule. The last rules are the same as before, used to send multiple messages.

To gain in generality, we consider

$$\mathcal{R}_{\mathcal{CST}'} = \mathcal{R}_{\mathcal{CST}} \cup \mathcal{R}_{\mathcal{BP}}$$

and

$$\mathcal{CST}' = (\Sigma_{\mathcal{CST}} \cup \Sigma_{\mathcal{BP}}, \mathcal{R}^{\approx}_{\mathcal{CST}} \cup \mathcal{R}^{\approx}_{\mathcal{BP}} \cup \mathcal{ACC})$$

We admit that lemmas 2.1 and 2.2 are still true with respect to $\mathcal{CST}'$ since the property of subterm convergence is not involved in their demonstration.

We define the set of message deduction rules MD as:

$$MD = \left\{ \begin{array}{l} \frac{Out(x)}{K(x)} \quad \frac{K(x)}{In(x)}[K(x)] \quad \frac{Fr(x:fr)}{K(x:fr)} \quad \frac{}{K(x:pub)} \\ \frac{K(x_1) \ldots K(x_k)}{K(f(x_1,\ldots,x_k))} \text{ for all } f \in \Sigma_{\mathcal{CST}'} \end{array} \right\}$$

We prove this lemma:

Lemma 3.2. *For all sets P of protocol rules,*

$$\{\overline{trace(dg)} | dg \in dgraphs(\lceil P \cup MD \rceil^{\mathcal{CST}'}_{insts}) \wedge dg \downarrow_{\mathcal{CST}'} \text{-}normal\} = \overline{trace(ndgraphs(P))}.$$

In order to prove this lemma, we define these sets of messages:

Definition. The known messages of a dependency graph dg:

$$known(dg) = \{m | \text{ exists conclusion fact } K(m) \text{ in } dg\}$$

The known-d messages of a normal dependency graph ndg:

$$known^d(ndg) = \{m | \text{ exists conclusion fact } K^d(m) \text{ in } ndg\}$$

for $d \in \{\Uparrow, \Downarrow^d, \Downarrow^e\}$.

The known$\Downarrow$ messages of normal dependency graph ndg:

$$known \Downarrow (ndg) = known \Downarrow^e (ndg) \cup known \Downarrow^d (ndg)$$

The known messages of normal dependency graph ndg:

$$known \Updownarrow (ndg) = known \Downarrow (ndg) \cup known \Uparrow (ndg)$$

The available state-conclusions of a dependency graph dg:

$$stfacts(dg) = \{f \in cfact(dg) | \forall m, d. f \neq K(m) \wedge f \neq K^d(m)\}$$

where $cfact(dg)$ denotes the consumable facts in dg.

The created messages of a dependency graph dg:

$$created(dg) = \{n | \text{ exists conclusion fact } Fr(n) \text{ in } dg\}.$$

Finally we say a normal dependency graph $ndg' = (I', D')$ is a deduction extension $ndg = (I, D)$ if I is a prefix of I', $D \subseteq D'$, $trace(ndg) = trace(ndg')$, $stfacts(ndg) = stfacts(ndg')$, and $created(ndg) = created(ndg')$.

To prove Lemma 3.2, we use these two lemmas proved in [16]:

Lemma 3.3. *For all $ndg \in ndgraphs(P)$ and conclusion facts $K^{\Downarrow}(m)$,there is a deduction extension ndg' that contain a conclusion fact $K^{\Uparrow}(m')$ with $m =_{ACC} m'$.*

This lemma describes that the adversary can always convert a $K \Downarrow$ fact into a $K \Uparrow$ fact.

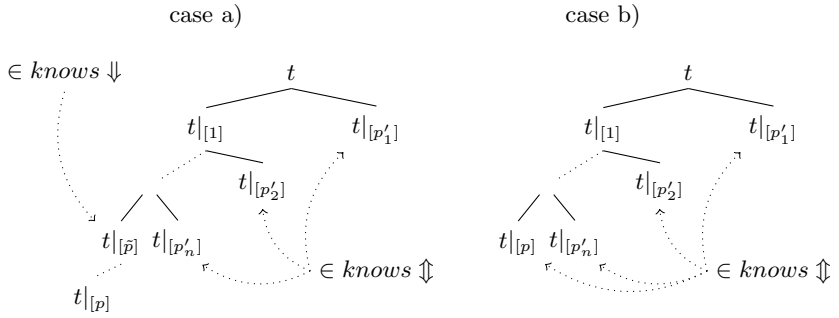
Lemma 3.4. *For all $ndg \in ndgraphs(P)$, $t \in known \Updownarrow (ndg)$ and valid positions p in t such that $root(t|_{p'}) \neq *$ for all p' above or equal to p, either*

a) there is a position $\tilde{p}$ strictly above p such that $t|_{\tilde{p}} \in_{ACC} known \Downarrow (ndg)$ and $t|_{p'} \in_{ACC} known \Updownarrow (ndg)$ for all valid positions p' in t that have a sibling above or equal to $\tilde{p}$, or

b) $t|_p \in_{ACC} known \Updownarrow (ndg)$ and $t|_{p'} \in known_{ACC} \Updownarrow (ndg)$ for all valid positions p' that have a sibling above or equal to p.

This lemmas divides terms known by the adversary into two categories illustrated in the following example:

Consider a term $t \in known_{ACC} \Updownarrow (ndg)$ and a valid position p then t can be in one of these situations:



(Note that we can have more that two siblings and that they can stop after more subterms.)

It shows in particular that, if, in a term $t \in known \Updownarrow (ndg)$, there is a valid position p where the subterm $t|_p \notin known \Updownarrow (ndg)$, then there is another position $\tilde{p}$, above p, where $t|_{\tilde{p}} \in known \Downarrow (ndg)$. This permits to establish deconstruction rules based on premises with subterms that are not in $known \Updownarrow (ndg)$.

We use common subterms to build deconstruction rules for context subterm theory.

Definition 3.5. A common subterm t of a rewriting rule $l \to r$ is a term such that there are p and q such that $t = l|_p = r|_q$.

A common maximal subterm t of a rewriting rule $l \to r$ is a common subterm of $l \to r$ such that there is no common subterm t' such that t is a subterm of t'.

For a given rewriting rule $l \to r$ where $vars(r) \neq \emptyset$, and for which there is a common maximal subterm $l|_p$, we use the function *ctxtdrules* to compute the corresponding deconstruction rules.

$$ctxtdrules(l, p, r) =$$
$$\{[K^{\Downarrow^d}(l|_{p'})] \cdot cprems(l, p') \,-[]\!\!\to [K^{\Downarrow^d}(r)] \mid p' \text{ strictly above } p \text{ and } p' \neq []\}$$

The set of deconstruction rules for a rewriting rule $l \to r$ is given by:

$$Ctxtdrules(l, P, r) = \bigcup_{p \in P} ctxtdrules(l, p, r)$$

where $P = \{p |\ \exists q, l|_p = r|_q\}$, the set of positions of common maximal subterms of $l \to r$.

The set DR_{CST} of deconstruction rules for $\mathcal{CST}$ is given by:

$$DR_{CST} = \bigcup_{(l,\ P,\ r)} Ctxtdrules(l, P, r)$$

where $l \to r$ is a rewriting rule of $\mathcal{R}_{\mathcal{CST}}$ with P for set of positions in l of common maximal subterms.

Thus, we have

$$ND = \left\{ \begin{array}{c} \dfrac{Out(x)}{K^{\Downarrow^d}(x)} \qquad \dfrac{K^{\Uparrow}(x)}{In(x)}[K(x)] \\ \\ \dfrac{K^{\Downarrow^d}(x)}{K^{\Uparrow}(x)} \quad \dfrac{K^{\Downarrow^e}(x)}{K^{\Uparrow}(x)} \quad \dfrac{Fr(x:fr)}{K^{\Uparrow}(x:fr)} \quad \dfrac{}{K^{\Uparrow}(x:pub)} \\ \\ \dfrac{K^{\Uparrow}(x_1) \quad \ldots \quad K^{\Uparrow}(x_k)}{K^{\Uparrow}(f(x_1, \ldots, x_k))} \text{ for all } f \in \Sigma_{\mathcal{CST}} \end{array} \right\} \cup DR_{CST}$$

Example 30. Consider the following signature

$$\Sigma_0 = \{a(\_,\_), b(\_,\_), c(\_,\_), d(\_,\_), g(\_,\_)\}$$

and the associated rewriting rule $\mathcal{R}_0 = \{a(b(c(x,y), d(z,w)), z) \to g(x,z)\}$.

To establish the deconstruction rules from ND, we look at the common maximal subterms in the rewriting rule, namely x and z. We have: $x = l|_{[1,1,1]}$, and $z = l|_{[1,2,1]} = l|_{[2]}$ so $P = \{[1,1,1], [1,2,1], [2]\}$ and

$$Ctxtrules(l, P, r) = ctxtdrules(l, [1,1,1], r) \cup ctxtdrules(l, [1,2,1], r) \cup ctxtdrules(l, [2], r)$$

We have

$$ctxtdrules(l, [1,1,1], r) = \left\{ \frac{K^{\Downarrow^d}(c(x,y)) \quad K^{\Uparrow}(d(z,w)) \quad K^{\Uparrow}(z)}{K^{\Downarrow^d}(g(x,z))}, \frac{K^{\Downarrow^d}(b(c(x,y), d(z,w))) \quad K^{\Uparrow}(z)}{K^{\Downarrow^d}(g(x,z))} \right\},$$

$$ctxtdrules(l, [1,2,1], r) = \left\{ \frac{K^{\Downarrow^d}(d(z,w)) \quad K^{\Uparrow}(c(x,y)) \quad K^{\Uparrow}(z)}{K^{\Downarrow^d}(g(x,z))}, \frac{K^{\Downarrow^d}(b(c(x,y), d(z,w))) \quad K^{\Uparrow}(z)}{K^{\Downarrow^d}(g(x,z))} \right\}$$

and $ctxtdrules(l, [2], r) = \emptyset$.

Then

$$Ctxtdrules(l, P, r) = \left\{ \begin{array}{c} \dfrac{K^{\Downarrow^d}(c(x,y))\ K^{\Uparrow}(d(z,w))\ K^{\Uparrow}(z)}{K^{\Downarrow^d}(g(x,z))}, \dfrac{K^{\Downarrow^d}(d(z,w))\ K^{\Uparrow}(c(x,y))\ K^{\Uparrow}(z)}{K^{\Downarrow d}(g(x,z))}, \\ \dfrac{K^{\Downarrow^d}(b(c(x,y),d(z,w)))\ \ K^{\Uparrow}(z)}{K^{\Downarrow^d}(g(x,z))} \end{array} \right\}$$

Since the rule $\frac{K^{\Downarrow^d}(b(c(x,y),d(z,v)))\ \ K^{\Uparrow}(w)}{K^{\Downarrow^d}(g(x,z))}$ appears in both *ctxtdrules*, we obtain only three deconstruction rules. Note that there is no need for a rule with two $K^{\Downarrow}$ facts.

Example 31. We apply this to the blind signature rewriting rule

$$unblind(sign(blind(m,r),k),r) \to sign(m,k).$$

We have m and k as common maximal subterms on respective positions $[1,1,1]$ and $[1,2]$. Then we consider the following deconstruction rules:

$$ctxtdrules(l,[1,1,1],r) = \left\{ \begin{array}{c} \dfrac{K^{\Downarrow^d}(blind(m,r))\ \ K^{\Uparrow}(k)\ \ K^{\Uparrow}(r)}{K^{\Downarrow^d}(sign(m,k))}, \dfrac{K^{\Downarrow^d}(sign(blind(m,r),k))\ \ K^{\Uparrow}(r)}{K^{\Downarrow^d}(sign(m,k))} \end{array} \right\}$$

and

$$ctxtdrules(l,[1,2],r) = \left\{ \begin{array}{c} \dfrac{K^{\Downarrow^d}(sign(blind(m,r),k))\ \ K^{\Uparrow}(r)}{K^{\Downarrow^d}(sign(m,k))} \end{array} \right\}.$$

Thus,

$$Ctxtdrules(l,\{[1,1,1],[1,2]\},r) = \left\{ \begin{array}{c} \dfrac{K^{\Downarrow^d}(blind(m,r))\ \ K^{\Uparrow}(k)\ \ K^{\Uparrow}(r)}{K^{\Downarrow^d}(sign(m,k))}, \dfrac{K^{\Downarrow^d}(sign(blind(m,r),k))\ \ K^{\Uparrow}(r)}{K^{\Downarrow^d}(sign(m,k))} \end{array} \right\}$$

Then we have set $ND_{\mathcal{BS}}$ of normal deduction message rules for BS:

$$ND_{\mathcal{BS}} = \left\{ \begin{array}{c} \dfrac{Out(x)}{K^{\Downarrow^d}(x)} \\[2ex] \dfrac{K^{\Downarrow^d}(blind(m,r)) \quad K^{\Uparrow}(r)}{K^{\Downarrow^d}(m)} \qquad \dfrac{K^{\Downarrow^d}(sign(m,k)) \quad K^{\Uparrow}(pk(k))}{K^{\Downarrow^d}(m)} \\[2ex] \dfrac{K^{\Downarrow^d}(\langle x,y\rangle)}{K^{\Downarrow^d}(x)} \qquad \dfrac{K^{\Downarrow^d}(blind(m,r)) \quad K^{\Uparrow}(k) \quad K^{\Uparrow}(r)}{K^{\Downarrow^d}(sign(m,k))} \\[2ex] \dfrac{K^{\Downarrow^d}(\langle x,y\rangle)}{K^{\Downarrow^d}(y)} \qquad \dfrac{K^{\Downarrow^d}(sign(blind(m,r),k)) \quad K^{\Uparrow}(r)}{K^{\Downarrow^d}(sign(m,k))} \\[2ex] \dfrac{K^{\Downarrow^d}(x)}{K^{\Uparrow}(x)} \qquad \dfrac{K^{\Downarrow^e}(x)}{K^{\Uparrow}(x)} \qquad \dfrac{Fr(x:fr)}{K^{\Uparrow}(x:fr)} \qquad \dfrac{}{K^{\Uparrow}(x:pub)} \\[2ex] \dfrac{K^{\Uparrow}(x) \quad K^{\Uparrow}(y)}{K^{\Uparrow}(\langle x,y\rangle)} \qquad \dfrac{K^{\Uparrow}(p)}{K^{\Uparrow}(fst(p))} \qquad \dfrac{K^{\Uparrow}(p)}{K^{\Uparrow}(snd(p))} \\[2ex] \dfrac{K^{\Uparrow}(m) \quad K^{\Uparrow}(k)}{K^{\Uparrow}(sign(m,k))} \qquad \dfrac{K^{\Uparrow}(s) \quad K^{\Uparrow}(pk)}{K^{\Uparrow}(checksign(s,pk))} \qquad \dfrac{K^{\Uparrow}(k)}{K^{\Uparrow}(pk(k))} \\[2ex] \dfrac{K^{\Uparrow}(m) \quad K^{\Uparrow}(r)}{K^{\Uparrow}(blind(m,r))} \qquad \dfrac{K^{\Uparrow}(b) \quad K^{\Uparrow}(r)}{K^{\Uparrow}(unblind(b,r))} \\[2ex] \dfrac{K^{\Uparrow}(x)}{In(x)}[K(x)] \end{array} \right\}$$

We can now prove Lemma 3.2 by adapting the proof we find in [16]. The proof considers a dependency graph on which we add a rule instance and see if it is convertible into a possible rule instance from $ginst(\lceil P \cup ND \rceil^{\mathcal{R}_{\mathcal{CSP}'}}_{insts} \cup \{Fresh\})$ to complete an equivalent normal dependency graph.

Proof of Lemma 3.2. We prove both inclusions by induction on graphs.

$\subseteq_{\mathcal{ACC}}$: We prove that, for all $dg \in dgraphs_{\mathcal{ACC}}(\lceil P \cup MD \rceil^{\mathcal{R}_{\mathcal{CSP}'}}_{insts})$ with $dg \downarrow_{\mathcal{R}_{\mathcal{CSP}'}}$ -normal, there is $ndg \in ndgraphs(P)$ such that:

$$known(dg) \subseteq_{\mathcal{ACC}} known \Updownarrow (ndg) \tag{1}$$
$$stfacts(dg) \subseteq_{\mathcal{ACC}} stfacts(ndg) \tag{2}$$
$$created(dg) =_{\mathcal{ACC}} created(ndg) \tag{3}$$
$$\overline{trace(dg)} =_{\mathcal{ACC}} \overline{trace(ndg)} \tag{4}$$

Notice that the inclusion of (2) applies to multisets.

The four properties hold for $dg = ([], \emptyset)$. Let $dg = (I, D) \in dgraphs_{ACC}(\lceil P \cup MD \rceil^{\mathcal{R}_{\mathcal{CSP}'}}_{insts})$ with $dg \downarrow_{\mathcal{R}_{\mathcal{CSP}'}}$ -normal, and $ndg = (\tilde{I}, \tilde{D}) \in ndgraphs(P)$ such that (1)-(4) hold. Let $ri \in ginst_{ACC}(\lceil P \cup MD \rceil^{\mathcal{R}_{\mathcal{CSP}'}}_{insts} \cup \{Fresh\})$ such that $dg' = (I \cdot ri, D \biguplus D') \in dgraphs_{ACC}(\lceil P \cup MD \rceil^{\mathcal{R}_{\mathcal{CSP}'}}_{insts})$. We must show that

there is a deduction extension ndg' of ndg that satisfies (1)-(4) with respect to dg'.

We perform a case distinction on ri:

- for $ri \in ginst_{\mathcal{ACC}}(\lceil P \rceil^{\mathcal{R}_{CST'}}_{insts} \cup \{Fresh\})$, we can extend ndg to $ndg' = (\tilde{I} \cdot ri, \tilde{D} \biguplus \tilde{D}')$ in accordance with conditions (2) and (3) ,
- for $ri \in \{Out(m) \ -[]\rightarrow \ K(m), K(m) \ -[K(m)]\rightarrow \ In(m), Fr(n) \ -[]\rightarrow \ K(n), \ -[]\rightarrow K(c)\} \subset ginst_{\mathcal{ACC}}(\lceil MD \rceil^{\mathcal{R}_{CST'}}_{insts}$, there are corresponding rules in ND that can complete $\tilde{I}$,
- for the rules

  $$[K(m_1), \ldots, K(m_n)] \ -[]\rightarrow [K(f(m_1, \ldots, m_n))] \in ginst_{\mathcal{ACC}}(\lceil MD \rceil^{\mathcal{R}_{CST'}}_{insts}),$$

  we can associate a construction rule because, according to lemma b) we can have $m_i \in known \Uparrow (ndg')$,
- for the rules $[K(m_1), \ldots, K(m_k)] \ -[]\rightarrow [K(m)]$ with $m = f(m_1, \ldots, m_k) \downarrow_{\mathcal{R}_{CST'}}$ where $f \in \Sigma_{CST'}$ and $m \neq_{ACC} f(m_1, \cdots, m_k)$. We can assume that $m \notin_{ACC} known \Updownarrow (ndg)$. Then, there is a rule $l \rightarrow r$ in $\mathcal{R}_{CST'}$ such that $f(m_1, \ldots, m_k)$ is an instance of l and, either $m = r \in \mathcal{T}_{\Sigma_{CST'}}(\emptyset)$ and in normal form, or, there are k', $j_1, \ldots, j_{k'}$ and $p_1, \ldots, p_{k'}$ such that $m \in \mathcal{T}_{\Sigma_{CST'}}(m_{j_1}|_{p_1}, \ldots, m_{j_{k'}}|_{p_{k'}})$.

  For instance, if $l = unblind(m_1, m_2)$ where $m_1 = sign(blind(x, m_2), k)$, then $x = m_1|_{[1,1]}$ and $k = m_1|_{[2]}$, so
  $m = sign(x, k) \in \mathcal{T}_{\Sigma_{CST'}}(m_1|_{[1,1]}, m_1|_{[2]})$. If $r \in \mathcal{T}_{\Sigma_{CST'}}(\emptyset)$, we can use construction rules to build m. In the other case, we can assume that there is an $m_{j_i}|_{p_i}$, namely $m_{j'}|_{p'}$, that is not in $known \Updownarrow (ndg)$ otherwise we could use a construction rule to build m, thus $m_{j'}|_{p'}$ satisfies the case a) of Lemma 3.4, and there is a valid position $\tilde{p}$ strictly above p' such that $m_{j'}|_{\tilde{p}} \in known \Downarrow (ndg)$. Moreover $m_{j'}|_{\tilde{p}} \in known \Downarrow^d (ndg)$ because, since we apply $l \rightarrow r$, then $m_{j'}|_{\tilde{p}}$, which is in a position strictly above $m_{j'}|_{p'}$, verifying $root(m_{j'}|_{\tilde{p}}) \in \Sigma_{CST'}$ and so $m_{j'}|_{p'}$ can't be in $known \Downarrow^e (ndg)$. Thus we can use the corresponding rule from $Ctxtdrules$, namely $[K^{\Downarrow^d}(l|_{[j']\cdot\tilde{p}})] \cdot cprems(l, \tilde{p}) \ -[]\rightarrow [K^{\Downarrow^d}(l \downarrow_{\mathcal{R}_{CST'}})]$.
- Other rules only deal with the bilinear-paring theory, the fact that CST' is not necessarily subterm convergent does not interfere with the original proof.

$\supseteq_{\mathcal{ACC}}$: For the other inclusion, we show, by an analogous way, that for all $ndg \in ndgraphs(P)$,there is a $dg \in dgraphs_{\mathcal{ACC}}(\lceil P \cup MD \rceil^{\mathcal{R}_{CST'}}_{insts})$ with $dg \downarrow_{\mathcal{R}_{CST'}}$ -normal, such that:

$$known(ndg) \subseteq_{\mathcal{ACC}} known \Updownarrow (dg) \quad (1)$$

$$stfacts(ndg) \subseteq_{\mathcal{ACC}} stfacts(dg) \quad (2)$$

$$created(ndg) =_{\mathcal{ACC}} created(dg) \quad (3)$$

$$\overline{trace(ndg)} =_{\mathcal{ACC}} \overline{trace(dg)} \quad (4)$$

It holds for $ndg = ([], \emptyset)$. Let $ndg = (I, D) \in ndgraphs(P)$, and $dg = (\tilde{I}, \tilde{D}) \in dgraphs_{ACC}(\lceil P \cup MD \rceil^{\mathcal{R}_{cst'}}_{insts})$ with $dg \downarrow_{\mathcal{R}_{cst'}}$ -normal such that (1)-(4) hold. Let $ri \in ginst_{ACC}(\lceil P \rceil^{\mathcal{R}_{cst'}}_{insts} \cup ND \cup \{Fresh\})$ such that $ndg' = (I \cdot ri, D \biguplus D') \in ndgraphs_{ACC}(P)$. We must show that there is a deduction extension dg' of dg that satisfies (1)-(4) with respect to ndg'. We still perform a case distinction on ri

- for $ri \in ginst_{\mathcal{ACC}}(\lceil P \rceil^{\mathcal{R}_{cst'}}_{insts} \cup \{Fresh\})$: it is analogous to the other inclusion,
- for $ri \in ginst_{\mathcal{ACC}}(ND)$: there is a corresponding rule in $\lceil MD \rceil^{\mathcal{R}_{cst'}}_{insts}$ for most of rules instance ri. For the others, we consider variants of rules for function symbols. For example, if $ri = \dfrac{K^{\Downarrow^d}(blind(m,r)) \quad K^{\Uparrow}(k) \quad K^{\Uparrow}(r)}{K^{\Downarrow^d}(sign(m,k))}$, we consider the trivial variant of the signing rule: $\dfrac{K(blind(m,r)) \quad K(k)}{K(sign(blind(m,r),k))}$ followed by the variant of the unblinding rule: $\dfrac{K(sign(blind(m,r),k)) \quad K(k)}{K(sign(m,k))}$

□

3.3 Implementation of context subterm rules in Tamarin

Now, we are able to perform the implementation of context subterm rules in Tamarin. Since subterm rules have already been implemented, only a few changes will be necessary.

3.3.1 Haskell

The source code of Tamarin is written in Haskell. It is a strongly typed functional programming language.

A functional language consider calculus like evaluation of mathematical functions by opposition with imperative languages where operations are described by sequences of operations like affectations or loops.

A language is strongly typed if each data is associated with a type that must be respected without implicit conversion.

A classic example of Haskell is the quick-sort algorithm that is easily written in such a functional language:

Example 32. The quick-sort algorithm *qs* rearranges a list of ordered elements by ascending order. It is one of the fastest sort algorithm. In Haskell, it is written:

```
1   qs :: Ord a ⇒ [a] → [a]
2   qs (x:xs) =
3     qs (filter (<x) xs )
4     ++ [x] ++
5     qs (filter (>=x) xs )
```

The first line gives and imposes informations on the function *qs*, **Ord** *a* denotes that the type *a* is of typeclass **Ord** (for ordered); such a type can be **Integral** for instance. $[a] \to [a]$ denote that it takes in argument a list, denoted by [ ], of elements of type *a* and returns a list of the same type.

The second line $qs\ (x : xs) =$ stand for "qs applied to $(x : xs)$ equals" where $(x : xs)$ is pattern matching for a list starting with x and followed by the list xs. Note that x is of type a but xs is of type $[a]$.

The following three lines denote the structure of the application of qs on $(x : xs)$. It places x between the elements greater and lesser than it by concatenation, denoted by $++$, of three lists: $qs\ (filter\ (< x)\ xs)$, the list of sorted elements lesser than x; $[x]$, the list containing x, and $qs\ (filter\ (>= x)\ xs)$, the list of sorted elements greater than x.

It can be useful to introduce other notions: [0..] denotes the infinite arithmetic list starting by 0 and 1. *elem* is a function of arity 2, often used in infix position., it tests the belonging. 4 '*elem*' [0..] returns *true*. Comprehension lists are convenient to define list: $[x * 2 |\ x \leftarrow [1..10],\ x * 2 >= 12]$ returns $[12, 14, 16, 18, 20]$. _ is used to denote indifferent argument, We can define function *second* on triplets by $second(\_, b, \_) = b$. The arguments are written after a function without parenthesis if they are not necessary. *map* permits to apply a function on each element of a list. And finally, here are three examples that define factorial function:

```
f :: (Integral a) => a -> a
f 0 = 1                 | f n                          | f n = case n of
f n = n * f (n-1)       |   | n ==0 = 1                |     0 -> 1
                        |   | otherwise = n * f (n-1)  |     _ -> n * f (n - 1)
```

Haskell runs by filtering patterns starting with the top one.

3.3.2 Implementation

Two major modifications have been done on the source code: they correspond to the generalisation from subterm rules to context subterm rules, and to an adaptation from $drules(l, p)$ to $Ctxtdrules(l, P, r)$.

In a first part we describe how the existing implementation of subterm rules worked then we adapt it to context subterm rules in the second part.

Former implementation

First we describe the three major types used to model subterm rules:

```
data RRule a = RRule a a
     deriving (Show, Ord, Eq)

data StRhs = RhsGround LNTerm | RhsPosition Position
    deriving (Show,Ord,Eq)

data StRule = StRule LNTerm StRhs
    deriving (Show,Ord,Eq)
```

Type **RRule** stands for a rewriting rule. It means that they are composed of two elements of the same type a. This type has the properties of typeclasses **Show**, **Ord** and **Eq** for elements that are displayable, ordered, and testable with equality.

Type **StRhs** denotes that a right-hand side term can be either a ground term or a position in the left-hand side term.

Finally, type **StRule** models subterm rewriting rules as the composition of a left-hand side term, denoted by the type **LNTerm**, and an element of type **StRhs** (ground term or a position).

Since the types used for modelling rewriting rules and subterm rules, we explain the algorithm *rRuleToStRule* that converts rewriting rules to subterm rules:

```
1  rRuleToStRule :: RRule LNTerm -> Maybe StRule
2  rRuleToStRule (lhs `RRule` rhs)
3      | frees rhs == [ ] = Just $ StRule lhs (RhsGround rhs)
4      | otherwise = case findSubterm lhs [ ] of
5          [ ]:_      -> Nothing
6          pos:_      -> Just $ StRule lhs (RhsPosition (reverse pos))
7          [ ]        -> Nothing
8      where
9        findSubterm t rpos | t == rhs = [rpos]
10       findSubterm (viewTerm -> FApp _ args) rpos =
11         concat $ zipWith (\ t i -> findSubterm t (i:rpos)) args [0..]
12       findSubterm (viewTerm -> Lit _) _ = [ ]
```

This algorithm takes as argument a rewriting rule defined by two terms denoted in infix notation by (lhs '**RRule**' rhs) of type $LNTerm$ and converts it, if it is possible (**Maybe**), into a subterm rule.

In the two following lines (3 and 4), we see vertical bars |, they stand for "if then elseif ... until otherwise". Line 3 tests if the set of variables of rhs is empty: $frees\ rhs\ ==\ [\ ]$ and in this case admits that rhs is ground. Line 5 is divided in three cases according to the value of $findSubterm\ lhs\ [\ ]$. Its description appears after the *where* according to its arguments.

Line 9 denotes that, applied to a term t that is equal to the right-hand side rhs, and, a position $rpos$, $findSubterm$ returns the list containing this position $[rpos]$. Line 10 considers the right-hand side terms lhs of the form $viewTerm\ ->\ \mathbf{FApp}\ \_\ args$, an undefined function denoted by _ applied on arguments $args$. $viewTerm$ permits the extraction of the term. For a function f with arguments $a_1, \ldots, a_k$, $findSubterm\ f(a_0, \ldots, a_k)\ rpos$ concatenates the results of $findSubterm\ a_i\ (i : rpos)$ for each i, $0 < i < k$. Notice that positions start from 0 and not 1, and are in reverse order. Informally, for a rewriting rule $l \to r$, this represents the fact of running down the tree of subterms of l and extract the positions of r in l. If it reaches a constant or variable different than r (line 12), it sends $[\ ]$ that disappears with the concatenation. Notice that we use type **Lit** for literals, ie, constants or variables.

Then, we can explain lines 5, 6 and 7:

- if $findSubterm\ lhs\ [\ ]$ returns a list starting with the empty position, denoted by $[\ ] : \_$, it means that the rewriting rule is of the kind $r \to r$ and therefore not considered,
- if it returns a list starting with a different position than $[\ ]$, the rewriting rule is converted into a subterm rule with the first position (reversed) of the list as position of the subterm,
- otherwise it does not convert the rewriting rule.

Now, we have to convert these subterm rules into deconstruction rules. We use the *destructionRules* algorithm for which we only explain the following part:

```
1  destructionRules :: Bool -> StRule -> [IntrRuleAC]
2  destructionRules _ (StRule lhs@(viewTerm -> FApp (NoEq (f,_)) _)
                      (RhsPosition pos)) =
3        go [ ] lhs pos
4     where
5        rhs = lhs `atPos` pos
6        go _ _ [ ] = [ ]
7        go _ (viewTerm -> FApp _ _) (_:[ ]) = [ ]
8        go uprems (viewTerm -> FApp _ as) (i:p) =
9             irule ++ go uprems' t' p
10          where
11             uprems' = uprems ++ [ t | (j, t) <- zip [0..] as, i /= j ]
12             t' = as!!i
13             irule = if (t' /= rhs && rhs `notElem` uprems')
14                        then [ Rule (DestrRule f)
15                                    ((kdFact t'):(map kuFact uprems'))
16                                    [kdFact rhs] [ ] ]
17                        else [ ]
18       go _ (viewTerm -> Lit _) (_:_) =
19            error "IntruderRules.destructionRules: impossible,
                     position, invalid"
```

This code is for subterm rules where the left-hand side term *lhs* has f for root function and its right-hand term is found in *lhs* at position *pos*. The boolean argument is not involved in our problem (it stands for private or public functions).

This function returns the result of a function *go* applied on arguments [ ], *lhs* and *pos*. The structure of *go* is described between lines 8 and 17, given a list *uprems* of terms, a function applied on a list *as* of arguments, and a position $(i : p)$ starting with i, it computes:

- a new list, $uprems'$, that corresponds to the old one concatenated with the list of arguments from *as* that are not at position i. it represents terms in $K^{\Uparrow}$ facts,
- t' as the i-th argument, it represents the term in $K^{\Downarrow}$ fact.
- and *irule*, the associated deconstruction rules while t' has not reached the right-hand side term *rhs* and if *rhs* is not in the list of newly computed $K^{\Uparrow}$ facts.

Moreover, *go* adds *irule* to the result of *go* at the next position p. When it reaches the last position $(\_:[])$, *go* returns the empty list (line 7) as well as when *go* is applied on an empty position (line 6).

Informally, the algorithm runs, from root to leaves, the tree representing *lhs*, on the branch where we find the subterm *rhs* and computes premises and deconstruction rule at each position until it reaches *rhs*

Line 18 describes an error message sent when a constant or a variable different than *rhs* is found on the branch.

Now that we understand how the former implementation with subterm rules worked, we adapt it to context subterm rules.

New implementation

First we modify the type for the right-hand side of a rewriting rule without excluding subterm rules:

```
data StRhs = StRhs [Position] LNTerm
    deriving (Show,Ord,Eq)
```

The term of type **LNTerm** contains the right-hand side of the rule, and the list of type [**Position**] contains the maximal subterms common with the left-hand side.

Example 33. As an example, for the rewriting rule $unblind(sign(blind(m,r),k),r) \rightarrow sign(m,k)$, the right-hand side term will be defined as **StRhs** $sign(m,k)$, $[[0,0,0],[0,1]]$. Remember that, to simplify algorithms, position starts at 0.

Then we describe context subterm rules as the composition of its left-hand side term and its right-hand side term.

```
data CtxtStRule = CtxtStRule LNTerm StRhs
    deriving (Show,Ord,Eq)
```

From this, we can build an algorithm *rRuleToCtxtStRule* that extracts positions of common maximal subterms. It is an extension of the previous one. We consider the left-hand side *lhs* and the right-hand side *rhs* as trees that we explore from root to leaves. For each subterm of *rhs*, we test its presence in *lhs*. We still use *findSubterm* to find explore *lhs* and create *findAllSubterms* to explore *rhs*.

```
1  rRuleToCtxtStRule :: RRule LNTerm -> Maybe CtxtStRule
2  rRuleToCtxtStRule (lhs `RRule` rhs)
3    | frees rhs == [ ] = Just $ CtxtStRule lhs (StRhs [ ] rhs)
4    | otherwise = case findAllSubterms lhs rhs of
5                    [ ]:_    -> Nothing
6                    [ ]      -> Nothing
7                    pos      -> Just $ CtxtStRule lhs (StRhs pos rhs)
8    where
9      findSubterm lst r rpos | lst == r = [reverse rpos]
10     findSubterm (viewTerm -> FApp _ args) r rpos =
11       concat $ zipWith ( \ lst i -> findSubterm lst r (i:rpos)) args [0..]
12     findSubterm (viewTerm -> Lit _) _ _ = [ ]
13     findAllSubterms l r@(viewTerm -> FApp _ args)
14       | fSt == [ ] = concat $ map (\ rst -> findAllSubterms l rst) args
15       | otherwise = fSt
16         where fSt = findSubterm l r [ ]
17     findAllSubterms l r@(viewTerm -> Lit _) = findSubterm l r [ ]
```

Example 34. We apply the function on the rewriting rule $unblind(sign(blind(m,r),k),r) \rightarrow sign(m,k)$ as rhs '**RRule**' lhs. First, we have $frees\ rhs = [m,k] \neq [\,]$ so we look at the *otherwise* case. This calls the computation of $findAllSubterms\ lhs\ rhs$. findAllSubterms is defined by pattern matching in lines (13)-(17). rhs is of form $(viewTerm{-}¿\mathbf{FApp}\_args)$, namely the function is $sign$ and the list of arguments is $[m,k]]$, thus we are in the case of line 13. This calls the computation of $findSubterm\ lhs\ rhs$. The function browses lhs to find rhs as a subterm, since $sign(m,k)$ is not a subterm of $unblind(sign(blind(m,r),k),r)$, it returns the concatenation of the application of $findAllSubterms$ on lhs and the different arguments of rhs, namely, m and k. Finally, $findAllSubterms$ returns [[0,0,0],[0,1]] so:

rRuleToCtxtStRule(lhs '**RRule**' rhs) =
Just \$ **CtxtStRule** lhs (**StRhs** $[[0,0,0],[0,1]]$ rhs)

with $lhs = unblind(sign(blind(m,r),k),r)$ and $rhs = sign(m,k)$

Explicitly (but not exhaustively) we have:

$$
\begin{aligned}
&findAllSubterms\ lhs\ sign(m,k)\\
&\quad = (findAllSubterms\ lhs\ m\ ) +\!+ (findAllSubterms\ lhs\ k\ )\\
&\quad = (findSubterm\ lhs\ m\ [\,]) +\!+ (findSubterm\ lhs\ k\ [\,])\\
&\quad = (findSubterm\ m\ m\ [0,0,0]) +\!+ (findSubterm\ r\ m\ [1,0,0])\\
&\quad +\!+ (findSubterm\ k\ m\ [1,0]) +\!+ (findSubterm\ r\ m\ [1])\\
&\quad +\!+ (findSubterm\ m\ k\ [0,0,0]) +\!+ (findSubterm\ r\ k\ [1,0,0])\\
&\quad +\!+ (findSubterm\ k\ k\ [1,0]) +\!+ (findSubterm\ r\ k\ [1])\\
&\quad = [reverse[0,0,0]] +\!+ [reverse[1,0]]\\
&\quad = [[0,0,0],[0,1]]
\end{aligned}
$$

Because of the efficiency of the former implementation and of the adaptability of Haskell to mathematical definitions, the addition of deconstruction rules for context subterm rules is contained in one line that takes the role of *Ctxtdrules*:

```
1  destructionRules :: Bool -> CtxtStRule -> [IntrRuleAC]
2  destructionRules bool (CtxtStRule lhs (StRhs (pos:posit) rhs)) =
3    destructionRules bool (CtxtStRule lhs (StRhs [pos] rhs))
     ++ destructionRules bool (CtxtStRule lhs (StRhs posit rhs))
```

where the application of *destructionRules* for each position is done by

```
1  destructionRules _ (CtxtStRule lhs@(viewTerm -> FApp (NoEq (f,_)) _)
                       (StRhs (pos:[ ]) rhs)) =
2        go [ ] lhs pos
3    where
4        go _ _
5    ...
```

This algorithm continues the same way as the old one, we just adapt types and input (pos : [ ]) as parameter to compute deconstruction rules for each position. Note that we remove $rhs = lhs$ '$atPos$' pos since rhs is now given as a parameter.

4 Cases Studies

We apply our new theory and implementation on three protocols: Chaum's online protocol, which is an E-cash protocol on which we prove unforgeability and anonymity properties, the FOO protocol, which is an E-voting protocol on which we prove eligibility and vote privacy, and finally the Okamoto protocol which is also an E-voting protocol on which we will face a problem to prove the receipt-freeness property.

4.1 Chaum's Online protocol

Chaum's Online protocol allows a client to withdraw a coin blindly from the bank, and then spend it later in a payment without being traced even by the bank. The protocol is "on-line" in the sense that the seller does not accept the payment before contacting the bank to verify that the coin has not been deposited before, to prevent double spending [10].

We consider three roles, the client C, the bank B and the seller S.

In a first phase, the withdrawal phase, the client blinds a coin x and sends it to the bank. The bank signs blindly the coin and sends the signature to the client. Then, in a second phase, the client unblinds the signature, and sends the coin x and the signature of x to the seller that check if the signature is correct. Then it sends it to the bank that sends back on a private channel the approval of the payment if the coin hasn't been already deposited. Then the seller accepts the coin.

4.1.1 Modelisation

The protocol can be schematised the following way:

$$\begin{aligned}
&C \longrightarrow B : blind(x, r)\\
&B \longrightarrow C : sign(blind(x, r), skB)\\
&C \longrightarrow S : \langle x, sign(x, skB)\rangle\\
&S \longrightarrow B : \langle x, sign(x, skB)\rangle\\
&B \longrightarrow S : x_{privately}
\end{aligned}$$

Thus we use the previous equational theory $\mathcal{BS}$, and $ND_{\mathcal{BS}}$ as deduction rules for the adversary.

Then we convert the exchanged message into protocol rules:

$$C_1 = \frac{Fr(x) \quad Fr(r)}{Out(blind(x, r)) \quad St\_C\_1(\$C, x, r)}$$

$$B_1 = \frac{In(blind(x, r)) \quad !Bank\_Ltk(\$B, ltkB)}{Out(sign(blind(x, r), ltkB))}[Withdraw(x)]$$

$$C_2 = \frac{St\_C\_1(C, x, r) \quad In(sign(blind(x, r), ltkB)) \quad !Bank\_Pk(B, pk(ltkB))}{Out(\langle x, sign(x, ltkB)\rangle)}$$

$$S_1 = \frac{In(\langle x, sign(x,\ ltkB)\rangle) \quad !Bank\_Pk(B, pk(ltkB))}{Out(\langle x,\ sign(x, ltkB)\rangle) \quad St\_S\_1(\$B, x)}$$

$$B_2 = \frac{In(\langle x,\ sign(x, ltkB)\rangle) \quad !Bank\_Pk(B, pk(ltkB))}{Private\_Ch(x)}[Deposited(x)]$$

$$S_2 = \frac{Private\_Ch(x) \quad St\_S\_1(B, x)}{}[Spend(x)]$$

Notice that, for rule C_2 for example, we impose that the received message is of the form $sign(blind(x, r), ltkB)$, this is called pattern matching. We could also have put the fact $In(m)$ and in the action facts added an event that verifies if m is actually the signature by the bank on the initial coin. Since the private channel can't be controlled by the adversary, it is modelled by the state facts $Private\_Ch(\_)$ placed in conclusion of the sender rule and in premise of recipient rule.

Additionally, to ensure that a bank does not accept a coin already deposited, we imposed the event $Deposited(x)$ to be unique. This is translated in Tamarin by an axiom:

$$axiom\ BankOnlyAcceptsOnce :$$
$$"All\ x\ \#j\ \#k.\ Deposited(x)@j\ \&\ Deposited(x)@k\ ==>\ \#j = \#k"$$

where *All* stands for "for all", # for position type and $f@i$ for "event f at position i"

We model the public key infrastructure for the bank as:

$$regiser\_pk\_Bank =$$
$$\frac{Fr(ltkB)}{!Bank\_Ltk(\$B, ltkB)\ !Bank\_Pk(\$B, pk(ltkB))\ Out(pk(ltkB))}[OnlyOnce()]$$

We consider that there is only one bank, this is modelled by the axiom that takes effect:

$$axiom\ OnlyOnce :$$
$$"All\ \#j\ \#k.\ OnlyOnce()@j\ \&\ OnlyOnce()@k\ ==>\ \#j = \#k"$$

Now the structure of protocol is established, we verify if it is executable by testing if it exists a trace where a coin is spent. Formally we prove the lemma:

$$lemma\ exec :$$
$$exists-trace\ "Ex\ x\ \#i.\ Spend(x)@i"$$

where Ex stands for "There exists" and "exists-trace" informs Tamarin that the property is provable by finding an example.

There are many executions that Tamarin can find. One interesting of them is the "classic" one where the protocol proceed correctly without other action from the adversary than reading public messages. Figure 10 illustrates it.

We can see on such an execution that every protocol rule is used and adversary actions are limited to intercept and send public messages without modifying them. The existence of such a trace shows that the protocol is probably correctly modelled. However, Tamarin indicates the presence of a rule for which

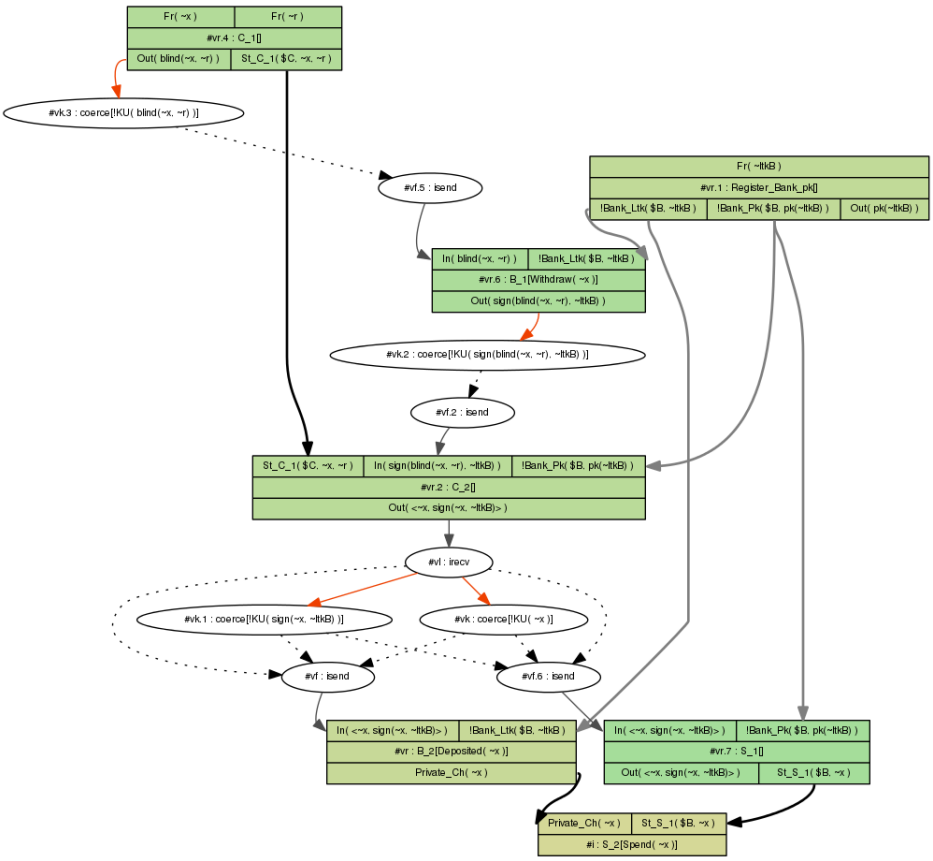


Figure 10: Execution of Chaum's Online protocol

it does not know what can be the result.We call this an " open chain ". Some open chains can be treated by induction using a lemma, called typing lemma, that restricts the possible results. Others, like the following one, are linked to an attack on the protocol.

The open chain we are facing is articulated around rule B_1. Indeed, a dishonest client can send a twice blinded message $blind(blind(m, r_1)r_2)$ to the bank that signs it leading to $sign(blind(blind(m, r_1)r_2), k_B$. Then by applying unblinding, the client gets $sign(blind(m, r_1), k_B)$ and $sign(m, k_B)$ which consists in two signed coins although only one message has been signed. Since the adversary has the possibility to blind message an unbound number of times, it leads to an open chains. For the moment, we keep this as it is.

4.1.2 Unforgeability

We now prove that Chaum's Online protocol guaranties the property of unforgeability. This property ensures that, in an e-cash protocol, a client must not be able to create a coin without involving the bank, resulting in a fake coin, or to double spend a valid coin he withdrew from the bank [10]. Since the attack may comes from the client part, we don't consider rules C_1 and C_2 in the protocol rules. It leaves to the adversary the possibility to act like a honest client.

We model unforgeability with the following lemma:

$$
\begin{aligned}
&lemma\ unforgeability: \\
&\quad "All\ x\ \#j.\ Spend(x)@j\ ==> \\
&\quad (\ Ex\ \#i.\ Withdraw(x)@i\ \&\ \#i < \#j \\
&\quad \&\ not(\ Ex\ \#l.\ Spend(x)@l\ \&\ not(\ \#l = \#j\ )\ )\ )"
\end{aligned}
$$

Tamarin returns the expected attack we can see it in Figure 11.

Indeed we can see the spent coin is x, even though the withdrawn coin is $blind(x, r)$. The attack is not considered pertinent because the bank is supposed to verify that the blinded message has a correct structure. To do this verification, the bank performs a procedure called "Cut and Choose" that we model by imposing with pattern matching, that the received message is of the form $blind(\sim x, r)$. If x is fresh, ensured by the prefix $\sim$, then it cannot be a blind message.

Once this correction made, we can notice that open chains have also been solved.

By launching again the automatic verification, Tamarin finds another not pertinent attack where the seller accepts a blind coin whereas he should also verify the form of the received message (Figure 12). We correct this the same way. Under this last model, Tamarin manages to proof unforgeability.

Proposition 4.1. *Chaum's Online protocol ensures unforgeability.*

4.1.3 Anonymity

Anonymity is a property that we can prove using observational equivalence. Before defining what is anonymity, it is important to remark that in the context of observational equivalence, the adversary is given an additional rule:

$$equality = \frac{K^{\Downarrow}(m)\quad K^{\Uparrow}(m)}{}$$

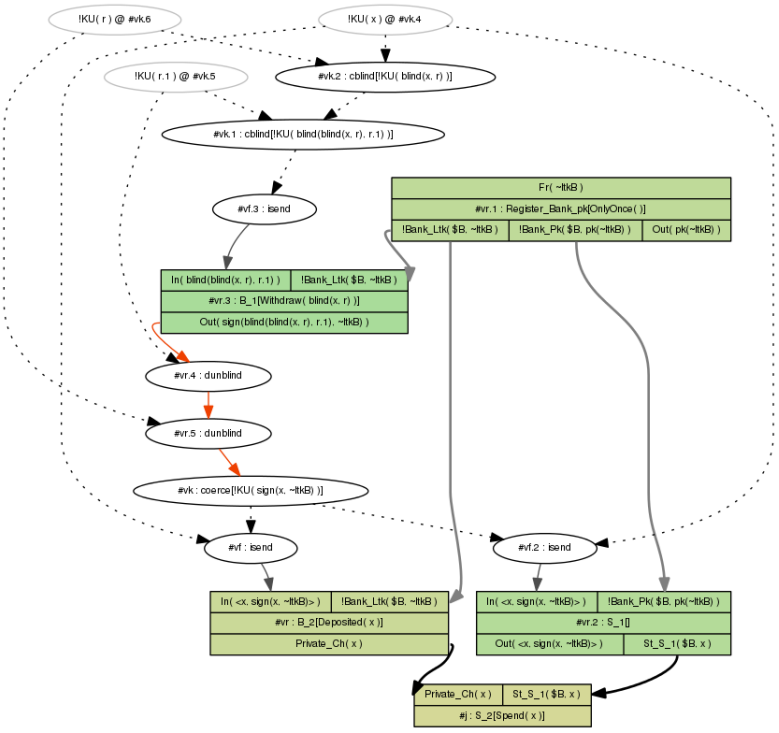


Figure 11: Attack on Chaum's Online protocol

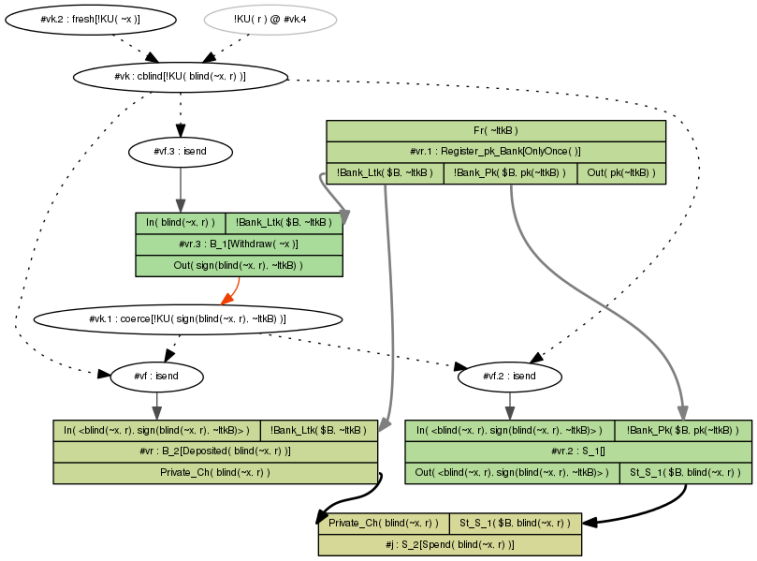


Figure 12: Attack on Chaum's Online protocol

It permits the adversary to compare messages.

We distinguish two kinds of anonymity: weak anonymity and strong anonymity. To define weak anonymity, we consider two clients C_1 and C_2 and the case where both of them, withdraw a coin in the same bank, but only one of them makes a purchase. Weak anonymity is the property guaranteeing that neither the bank nor the seller are able to distinguish the case where C_1 makes the purchase from the case where it is C_2 that makes it. Strong anonymity guarantees the same indistinguishability regardless of the number of clients and coins withdrawn or spent.

Weak anonymity is considered in a context where two clients are involved and the dishonest part is supposed to come from the bank or the seller. Thus, we model the identity of the clients by the constants $'c1'$ and $'c2'$ and consider a starting protocol rule for each one:

$$C_1^1 = \frac{Fr(\sim x) \quad Fr(\sim r)}{Out(blind(\sim x, \sim r)) \quad St\_C\_1(\ 'c1',\ \sim x,\ \sim r\ )}$$

and

$$C_1^2 = \frac{Fr(\sim x) \quad Fr(\sim r)}{Out(blind(\sim x, \sim r)) \quad St\_C\_2('\ c2',\ \sim x,\ \sim r\ )}$$

Then, we consider the rule C_2' representing that a client get his coin back signed by the bank. It is the same as C_2, used for unforgeability, except that the *Out* fact in conclusion is replaced by a state fact that stores the signature.

$$C_2' = \frac{St\_C\_1(\ C,\ x,\ r\ ) \quad In(sign(blind(x,r), ltkB)) \quad !Bank\_Pk(B, pk(ltkB))}{St\_C\_2(\ C,\ x,\ sign(x, ltkB)\ )}.$$

At this state, if a client makes a purchase while the other has not yet withdrawn any coin, it is trivial that one can identify the purchaser as the only client that made a withdraw. We need therefore, to perform a synchronisation step to ensure they both perform a withdrawing but also, consider separately the case where C_1 makes the purchase, and the case where C_2 makes it.

Then we specify the two rules C_2sync_n:

$$C_2sync_n = \frac{St\_C\_2(\ 'c1',\ x_1,\ s_1\ ) \quad St\_C\_2(\ 'c2',\ x_2,\ s_2\ )}{Out(<\ x_n,\ s_n\ >)}$$

where $n \in \{1, 2\}$

The two states facts in premises with identities as constants ensure the synchronisation in both rules C_2sync_1 and C_2sync_2 while the conclusion distinguishes the cases.

Since the bank and the seller are supposed to be dishonest, we do not involve any protocol rules involving them except for the key registration rule for which we publish the secret key:

$$\begin{aligned} &Reg\_Corrupted\_Bank\_Pk = \\ &\quad [Fr(\sim ltkB)] -[OnlyOnce]\rightarrow \\ &\quad [!Bank\_Ltk(\$B,\ \sim ltkB)\ ,\ !Bank\_Pk(\$B,\ pk(\sim ltkB))\ ,\ Out(\sim ltkB)] \end{aligned}$$

Then we consider the two protocol models P_1 and P_2:

$$P_1 = \left\{ \begin{array}{cccc} Reg\_Corrupted\_Bank\_Pk \\ C_1^1 \quad C_1^2 \quad C_2' \quad C_2 sync_1 \end{array} \right\}$$

and

$$P_2 = \left\{ \begin{array}{cccc} Reg\_Corrupted\_Bank\_Pk \\ C_1^1 \quad C_1^2 \quad C_2' \quad C_2 sync_2 \end{array} \right\}$$

The anonymity is considered proved if P_1 and P_2 are observationally equivalent [5].

The only difference between P_1 and P_2 is the $C_2 sync_n$ rule. To simplify the model, Tamarin disposes of the $diff(\_,\_)$ operator that permits to specify two cases in one rule. As first arguments of $diff(\_,\_)$, we put terms involved in P_1 and as second argument, we put those of P_2. Then we consider the rule:

$$C_2 sync = \frac{St\_C\_2(\ 'c1',\ x_1,\ s_1\ )\ \ St\_C\_2(\ 'c2',\ x_2,\ s_2\ )}{Out(diff(<\ x_1,\ s_1\ >,<\ x_2,\ s_2\ >))}.$$

System with the *diff* operator are called bi-system. Then we obtain the multiset rewriting bi-system P_{Chaum_Anonym} analysed by Tamarin as:

$$P_{Chaum\_Anonym} = \left\{ \begin{array}{cccc} Reg\_Corrupted\_Bank\_Pk \\ C_1^1 \quad C_1^2 \quad C_2' \quad C_2 sync \end{array} \right\}$$

which correspond to $P1$ and $P2$.

Tamarin verifies observational equivalence for P_{Chaum_Anonym}.

Proposition 4.2. *Chaum's Online protocol ensures weak anonymity.*

Remark. We do not prove strong anonymity but we can prove variants of weak anonymity by completing the $C_2 sync$ rule. For example, we can consider that C_1 and C_2 withdraw two coins each, and we distinguish the case where C_1 spends its coins while C_2 spends only one of them and the reverse case. This corresponds to the rule:

$$C_2' sync = \left[\begin{array}{c} St\_C\_2(\ 'c1',\ x_1,\ s_1\ ), \\ St\_C\_2(\ 'c1',\ x_2,\ s_2\ ), \\ St\_C\_2(\ 'c2',\ x_3,\ s_3\ ), \\ St\_C\_2(\ 'c2',\ x_4,\ s_4\ ) \end{array} \right] -[]\!\!\rightarrow \left[\begin{array}{c} Out(<\ x_1,\ s_1\ >), \\ Out(<\ x_3,\ s_3\ >), \\ Out(diff(<\ x_2,\ s_2\ >,<\ x_4,\ s_4\ >)) \end{array} \right]$$

4.2 The FOO protocol

The FOO protocol [12] allows a voter to publish a vote signed by the administration without being identified even by the administration. The protocol is designed to ensure that each published vote has been signed by the bank guaranteeing eligibility, and at the same time ensuring anonymity.

We consider three roles, the voter V, the administration A, and the collector C. The protocol is split into three phases described in [9].

- In the first phase the administrator signs the voter's commitment to his vote: Voter V chooses his vote v and computes a commitment $x = commit(v, r)$ for a random key r. He blinds the commitment using a random value b and obtains $e = blind(x, b)$. Then he signs e and sends the signature $sb_V = sign(e, LtkV)$ together with e and his identity to the administrator. The administrator checks if V has the right to vote and has not yet voted, and if the signature sb_V is correct. If all tests succeed, he signs $sb_A = sign(e, ltkA)$ and sends it back to V. V checks the signature, and unblinds it to obtain $s_A = unblind(sb_A, b) = sign(x, ltkA)$.
- In the second phase, the voter submits his ballot: Voter V sends (x, s_A) to the collector C through an anonymous channel. The collector checks the administrator's signature and enters (x, s_A) as the l-th entry into a list.
- When all ballots are cast the counting phase begins: The collector publishes the list of correct ballots. V verifies that his commitment appears on the list and sends (l, r) to C using an anonymous channel. The collector C opens the l-th ballot using r and publishes the vote.

4.2.1 Modelisation

The protocol can be schematised the following way:

$$\begin{aligned}
V &\longrightarrow A \quad : \langle e, sb_V \rangle \\
A &\longrightarrow V \quad : \langle e, sb_A \rangle \\
V &\longrightarrow C \quad : \langle x, s_A \rangle_{anonymously} \\
C &\longrightarrow Pub : \langle x, s_A \rangle_l \\
V &\longrightarrow C \quad : \langle r, l \rangle_{anonymously} \\
C &\longrightarrow Pub : v
\end{aligned}$$

To model commitment, we use the equational theory $\mathcal{BSC} = (\Sigma_{\mathcal{BSC}}, \mathcal{R}^{\approx}_{\mathcal{BSC}})$ where $\Sigma_{\mathcal{BSC}} = \Sigma_{\mathcal{BS}} \cup \{commit(\_,\_), open(\_,\_)\}$ and $\mathcal{R}_{\mathcal{BSC}} = \mathcal{R}_{\mathcal{BS}} \cup \{open(commit(m, r), r) \rightarrow m\}$. Then Tamarin computes the set $ND_{\mathcal{BSC}}$ of deduction rules.

We formalize the protocol under the following rules.

$$V_1 = \frac{Fr(x) \quad Fr(r) \quad !Ltk(V, ltkV)}{Out(\ \langle e,\ sign(e, ltkV)\rangle\ ) \quad St\_V\_1(V, \$vote, r, b)}$$

with $e = blind(commit(\$vote, r), b)$.

$$A_1 = \frac{In(\ \langle e,\ sign(e,\ ltkV)\rangle\ ) \quad !AdminLtk(A, ltkA) \quad !Pk(V, pkV)}{Out(\ \langle e,\ sign(e,\ ltkA)\rangle\ )}[Registered(e)]$$

$$V_2 = \frac{In(\ \langle e,\ sign(e, ltkA)\rangle\ ) \quad St\_V\_1(V, vote, r, b) \quad !AdminPk(A, pkA(ltkA))}{Out(\ \langle x,\ sign(x, ltkA)\rangle\ ) \quad St\_V\_2(V, A, vote, r)}$$

with $x = commit(vote, r)$ and $e = blind(x, b)$.

$$C_1 = \frac{In(\ \langle x,\ sign(x, ltkA)\rangle\ ) \quad !AdminPk(A, pkA(ltkA))}{St\_C\_1(A, x) \quad Out(\ \langle x,\ sign(x, ltkA),\ l\rangle\ )}$$

$$V_3 = \frac{In(\ \langle commit(vote, r),\ l\rangle\ )\ \ St\_V\_2(V, A, vote, r)}{Out(\ \langle r,\ l\rangle\ )}$$

$$C_2 = \frac{In(\ \langle r,\ l\rangle\ )\ \ St\_C\_1(A, x)}{Out(v)}[VotePublished(v)]$$

We model public key infrastructure for voters and administration.

$$register\_Voter\_pk = \frac{Fr(ltkV)}{!Ltk(\$V, ltkV)\ !Pk(\$V, pk(ltkV))\ Out(pk(ltkV))}$$

$$register\_Admin\_pk = \frac{Fr(ltkA)}{!Admin\_Ltk(\$A, ltkA)\ !Admin\_Pk(\$A, pk(ltkA))\ Out(pk(ltkA))}$$

In a first approach, we do not model the anonymous channel, it is not necessary to prove eligibility. Now the structure of the protocol is established, we verify if it is executable by proving the following lemma:

$$lemma\ exec:$$
$$exists-trace\ "Ex\ v\ \#i.\ VotePublished(v)@i"$$

Tamarin managed to find an execution whose trace satisfies the above lemma. It is illustrated in Figure 13.

It also detects 20 open chains that we managed to solve with a typing lemma.

4.2.2 Eligibility

We now prove that the FOO protocol guarantees the property of eligibility. This property ensures that, if a vote is published then its commitment has been signed by the administration.

We model eligibility with the following lemma:

$$lemma\ eligibility:$$
$$"All\ v\ \#j.\ VotePublished(v)@j\ \ ==>$$
$$(\ Ex\ b\ r\ \#i.\ Registered(blind(commit(v, r), b))@i\ \&\ \#i < \#j\ )"$$

It is successfully proved by Tamarin.

Proposition 4.3. *The FOO protocol ensures eligibility.*

4.2.3 Vote privacy

To define vote privacy, we consider two clients V_1 and V_2 and the case where, both of them commit a different vote among *yes* and *no* (for example). Vote privacy is the property guaranteeing that neither the administration nor the collector can distinguish the case where V_1 votes for *yes* from the case where he votes for *no*.

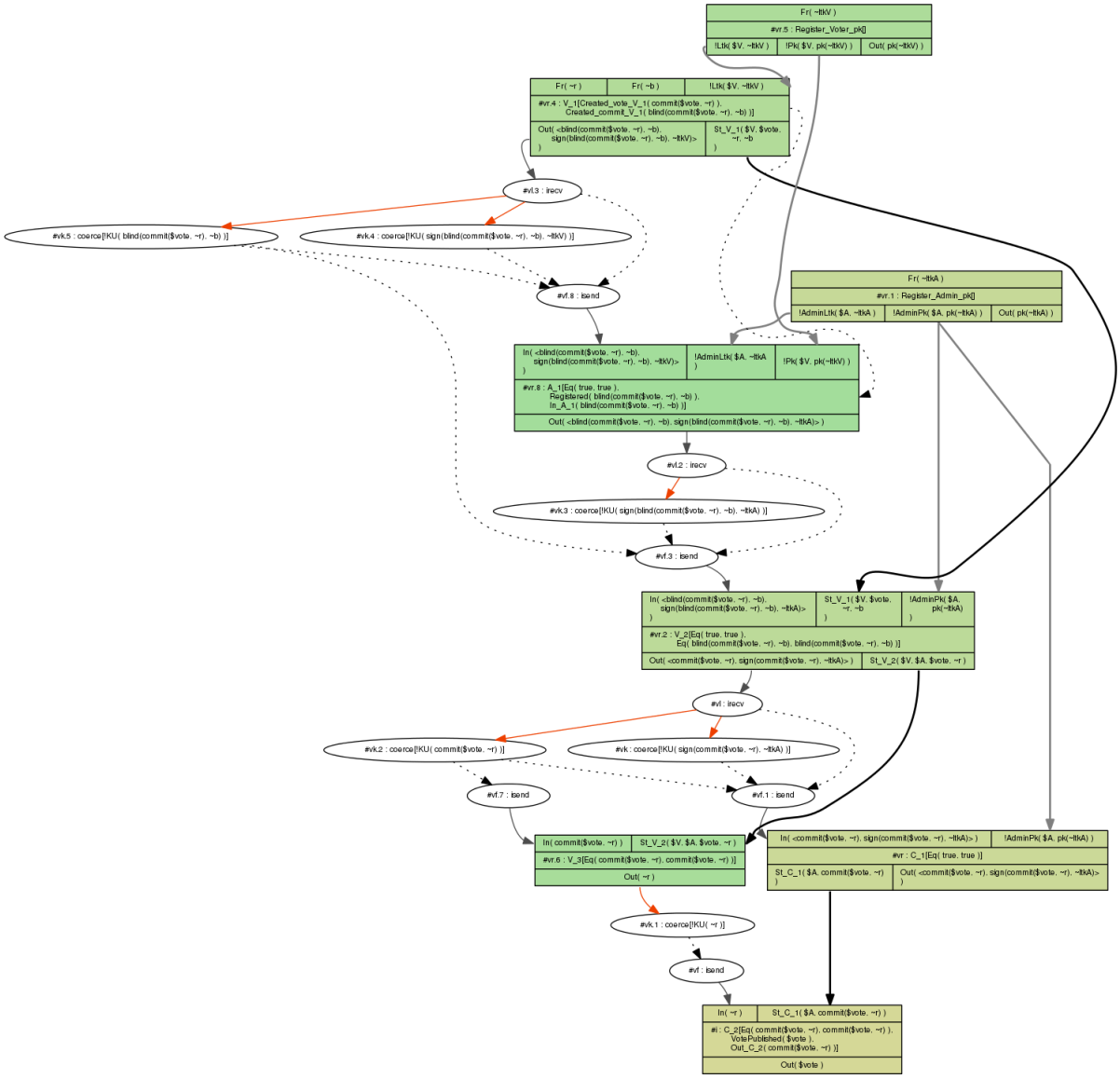


Figure 13: Execution of the FOO protocol

To model the bi-system, we consider the following rule that separates the two possible case.

$$setup = \frac{}{Vote1(diff(\$yes,\ \$no))\ \ Vote2(diff(\$no,\ \$yes))}[OnlyOnce()]$$

Votes are considered as public messages and, the *diff* operator and the state facts $Vote1$ and $Vote2$ ensure the distinction of the two cases.

As for Chaum's online protocol, honest participant's identity, here the voters, is modelled by constants. We consider the two following rules to introduce the voters:

$$V\_1\_1 = \frac{Fr(x)\quad Fr(r)\quad Vote1(vote)\quad !Ltk('v1', ltkV)}{Out(\ \langle e,\ sign(e, ltkV)\rangle\ )\quad St\_V\_1('v1', vote, r, b)}$$

$$V\_1\_2 = \frac{Fr(x)\quad Fr(r)\quad Vote2(vote)\quad !Ltk('v1', ltkV)}{Out(\ \langle e,\ sign(e, ltkV)\rangle\ )\quad St\_V\_1('v2', vote, r, b)}$$

with $e = blind(commit(vote, r), b)$.

Then we consider the second rule for voters. Since they send their commitment to the collector that is supposed to be dishonest, modelling the an anonymous channel is necessary. To do so, we consider the associative commutative theory multiset $\mathcal{AC}_m$. Its signature is only composed of the function symbol $+$. We use the fact that $a + b =_{\mathcal{AC}_m} b + a$ to model anonymous channel. Materially it can correspond to ballot box where ballots have been shuffled. To guarantee its effectiveness, it is preceded by a synchronisation phase We model this the following way:

$$V\_2 = \frac{In(\ \langle e,\ sign(e, ltkA)\rangle\ )\quad St\_V\_1(V, vote, r, b)\quad !AdminPk(A, pkA(ltkA))}{St\_V\_2\_sync(< V, pk(ltkA), \$vote, r >)}$$

with $x = commit(vote, r)$ and $e = blind(x, b)$,

$$V\_2\_sync = \frac{St\_V\_2\_sync(m1)\quad St\_V\_2\_sync(m2)}{Shuffle(\ m1 + m2\ )}$$

$$shuffle\_votes = \frac{Shuffle(\ m1 + m2\ )}{St\_V\_2(m1)\quad St\_V\_2(m2)}$$

$$V\_2\_out = \frac{St\_V\_2(< V, pk(\ ltkA), \$vote,\ r >)\quad !AdminPk(A, pk(\ ltkA))}{Out(< x, y >), St\_V\_3(V, \$vote,\ r)}$$

with $x = commit(\$vote, r)$ and $y = sign(x, ltkA)$

Finally, we model the rules for sending the key that opens commitment.

$$V_4 = \frac{In(commit(\$vote,\ r)), St\_V\_3(V, \$vote, r)}{Out(r)}$$

Since that the bank is supposed dishonest, we publish its long term key in the rule that registers bank keys.

Then we consider the associated bi-system $P_{FOO\_privacy}$

Tamarin verifies observational equivalence for $P_{FOO\_privacy}$.

Proposition 4.4. *The FOO protocol ensures vote privacy.*

4.3 The Okamoto protocol

The Okamoto protocol [14] is similar to the FOO protocol, but it uses trapdoor commitments and it involves a timeliness member to achieve the receipt-freeness property The first phase, during which the voter obtains a signature on his commitment x, is the same as for the FOO protocol, except that x is a trapdoor-commitment.

- In the second phase the vote is submitted; the voter V sends the signed trap-door commitment to the collector through an anonymous channel. The collector checks the administrators signature and enters (x, s_A) into a list. The voter sends (v, r, x) to the timeliness member T through an untappable anonymous channel.
- When all ballots are cast the counting phase begins: the collector publishes the list of correct ballots. V verifies that his commitment appears on the list. The timeliness member publishes the randomly shuffled list of votes [9].

4.3.1 Modelisation

The protocol can be shematised the following way:

$$\begin{aligned}
V &\longrightarrow A \quad : \langle e, sb_V \rangle \\
A &\longrightarrow V \quad : \langle e, sb_A \rangle \\
V &\longrightarrow C \quad : \langle x, s_A \rangle_{anonymously} \\
V &\longrightarrow T \quad : \langle v, r, x \rangle_{privatly} \\
C &\longrightarrow Pub : \langle x, s_A \rangle \\
T &\longrightarrow Pub : v
\end{aligned}$$

To model trap-door commitment algebraic properties, we can use the signature $\mathcal{BSTDC}_0 = (\Sigma_{\mathcal{BSTDC}}, \mathcal{R}_{\widetilde{\mathcal{BSTDC}}_0})$ where

$$\Sigma_{\mathcal{BSTDC}} = \Sigma_{\mathcal{BS}} \cup \{tdcommit(\_,\_,\_), open(\_,\_), f(\_,\_,\_,\_)\}$$

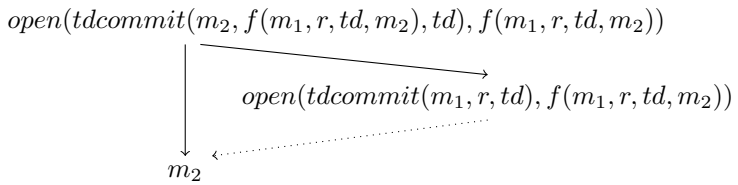
and

$$\mathcal{R}_{\mathcal{BSTDC}_0} = \mathcal{R}_{\mathcal{BS}} \cup \left\{ \begin{array}{c} open(tdcommit(m, r, td), r) \to m, \\ tdcommit(m_2, f(m_1, r, td, m_2), td) \to tdcommit(m_1, r, td) \end{array} \right\}.$$

This denotes that the voter is able to permute m_1 and m_2 in his commitment.

Remark. $\mathcal{R}_{\mathcal{BSTDC}_0}$ is not a convergent rewriting system.

Indeed, consider the term
$open(tdcommit(m_2, f(m_1, r, td, m_2), td), f(m_1, r, td, m_2))$. If we apply the first new rule, we directly obtain the term m_2. However, if we apply the second new rule, we obtain the term $open(tdcommit(m_1, r, td), f(m_1, r, td, m_2))$. But nothing says that $open(tdcommit(m_1, r, td), f(m_1, r, td, m_2))$ can be rewritten into m_2.



To obtain a convergent system, we have to consider $\mathcal{R}_{\mathcal{BSTDC}}$ as:

$$\mathcal{R}_{\mathcal{BSTDC}} = \mathcal{R}_{\mathcal{BSTDC}_0} \cup \left\{ \begin{array}{c} open(tdcommit(m_1, r, td), f(m_1, r, td, m_2)) \rightarrow m_2, \\ f(m_1, f(m, r, td, m_1), td, m_2) \rightarrow f(m, r, td, m_2) \end{array} \right\}.$$

And the associated equational theory $\mathcal{BSTDC}$.

We model the protocol with the following rules:

$$V\_1 = \frac{Fr(\ r)\ \ Fr(\ b)\ \ Fr(\ td)\ \ !Ltk(V,\ ltkV)}{Out(< e, s = sign(e,\ ltkV) >)\ \ St\_V\_1(V, \$vote,\ r,\ b,\ td)}$$

with $x = tdcommit(\$vote,\ r,\ td),\ e = blind(x,\ b)$.

$$A\_1 = \frac{In(< e, sign(e,\ ltkV) >)\ \ !AdminLtk(A,\ ltkA)\ \ !Pk(V, pkV)}{Out(< e, sign(e,\ ltkA) >)}[Registered(e)]$$

$$V\_2 = \frac{In(< e, sign(e,\ ltkA) >)\ \ St\_V\_1(V, \$vote,\ r,\ b,\ td)\ \ !AdminPk(A, pkA)}{Out(< x, sign(x,\ ltkA) >)\ \ P\_Ch\_Timeliness(\$vote,\ r, x)}$$

with $x = tdcommit(\$vote,\ r,\ td)$

$$C\_1 = \frac{In(< x, y >)\ \ !AdminPk(A, pkA)}{Out(< x, y >)}$$

$$T\_1 = \frac{P\_Ch\_Timeliness(\$vote,\ r, x)}{Out(\$vote)}[VotePublished(x)]$$

4.3.2 Eligibility and Vote Privacy

We now prove that the Okamoto protocol guarantees the property of eligibility. For the same reason as for the FOO protocol, we can not consider the adversary totally active.

We model eligibility with the following lemma:

lemma eligibility :

"*All v #j. VotePublished(v)@j ==>*

(Ex b r td #i. Registered(blind(tdcommit(v, r, td), b))@i & #i < #j)"

It is successfully proved by Tamarin.

Proposition 4.5. *The Okamoto protocol ensures eligibility.*

We do not explicit the model used to prove vote privacy since it is very similar.

However, Tamarin manages to prove vote privacy.

Proposition 4.6. *The Okamoto protocol ensures vote privacy.*

4.3.3 Receipt-freeness

Receipt-freeness is a property that guarantee that a voter cannot construct a receipt which allows him to prove to a third party that he voted for a certain candidate. This is to prevent vote-buying [9].

To model it, we a bi-system containing rule *setup* found in the FOO protocol model accompanied by the first rule for each voter:

$$setup = \frac{}{Vote1(diff(\$yes,\ \$no))\ \ Vote2(diff(\$no,\ \$yes))}[OnlyOnce()]$$

$$V\_1\_1 = \frac{Fr(\ r)\ Fr(\ b)\ Fr(\ td)\ Vote1(vote)\ !Ltk('v1',\ ltkV)}{Out(< e, s >)\ St\_V\_1('v1', vote,\ r,\ b,\ td)\ Out(< \$yes,\ f,\ td >)}$$

$$V\_1\_2 = \frac{Fr(\ r)\ Fr(\ b)\ Fr(\ td)\ Vote2(vote)\ \ !Ltk('v2',\ ltkV)}{Out(< e, s >)\ St\_V\_1('v2', vote,\ r,\ b,\ td)}$$

with $x = tdcommit(vote,\ r,\ td)$, $e = blind(x,\ b)$, $s = sign(e,\ ltkV)$ and $f = f(vote,\ r,\ td, \$yes)$.

This permits to consider a case where the voter $V1$ votes for *yes* and sends $< \$yes,\ f(\$yes,\ r,\ td, \$yes),\ td >$ as a receipt from the case he votes *no* and sends $< \$yes,\ f(\$no,\ r,\ td, \$yes),\ td >$ as a fake receipt.

In both cases, we have that:

$$open(tdcommit(yes,\ f(vote,\ r,\ td, \$yes), td),\ f(vote,\ r,\ td, \$yes)) = vote$$

According to this, an adversary could buy any without and be sure that he voted as the adversary asked.

In theory, the Okamoto protocol guarantees receipt-freeness against a Dolev-Yao adversary.

With Tamarin it is difficult to establish such a proof because Tamarin creates redundancy around the deconstruction rule for f:

$$\frac{K^{\Downarrow^d}(f(m, r, td, m_1))\ \ K^{\Uparrow}(m_1)\ \ K^{\Uparrow}(td)\ \ K^{\Uparrow}(m_2)}{K^{\Downarrow^d}(f(m, r, td, m_2))}$$

This is illustrated in Figure 14.

For now, we do not managed to solve this problem.

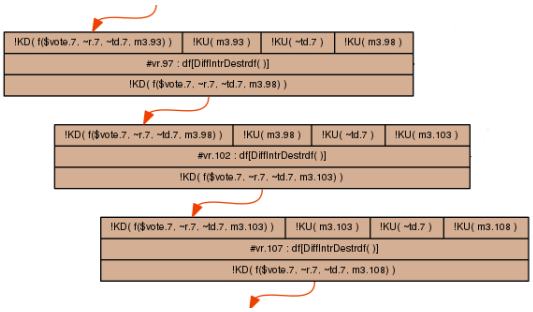


Figure 14: Redundancy around f

5 Conclusion

The goal of my internship was to adapt the Tamarin prover for equational theories more general than subterm convergent theories. In particular, we wanted to find all deconstruction rules associated to a rewriting rule. To achieve this, we managed to find how deconstruction rules should be determined so that Tamarin still construct proofs correctly. The proof of this correctness has been established, it is largely inspired the precedent proof found in [16].

Next, we have implemented such deconstruction rules in the source code of Tamarin. Since the former implementation was very clear and clean, it was really easy to adapt it.

Then we have tested the new implementation by running Tamarin. We observed that he correctly computed deconstruction rules. Thus we were able to make Tamarin prove security properties on three protocol involving blind signature. That was not possible before.

Formalising such protocols on Tamarin was not always an easy task, and most of the models used should be questioned or optimized. For instance the model for synchronisation and anonymous channel used for FOO and Okamoto protocols should be perfectible using rules like the following:

$$Anonymous\_Ch\_Out = \frac{Anonym\_Ch\_Out(m1 + m2)}{Out(m1) \quad Out(m2)}$$

and

$$Anonymous\_Ch\_In = \frac{In(m1) \quad In(m2)}{Anonym\_Ch\_In(m1 + m2)}$$

but the properties that involve anonymous channels required too much time to be proved. Moreover, it would have been possible to gain such a time by looking for lemmas that Tamarin could reuse.

All of the cases studies are findable on the github for Tamarin by following the path "tamarin-prover/examples/features/equational_theories" with other examples for testing

The adaptation of context subterm rules for Tamarin has progressed but it is not yet totally achieved.

The model of the protocol of Okamoto showed some rewriting rules are difficult to adapt. An idea to resolve the problem of redundancy found with the rule $f(m_1, f(m, r, td, m_1), td, m_2) \rightarrow f(m, r, td, m_2)$ is to decompose $K^{\Downarrow}$ facts into two cases like it is done for Diffie-Hellman exponentiation.

Another kind of rewriting rule that we have not treated is the $verify$ rule that have a ground term for right-hand side term. For instance, instead of using $checksign(sign(m, k), pk(k)) \rightarrow m$ and pattern matching for extracting messages and verifying signature, we can use $getmess(sign(m, k)) \rightarrow m$, $verify(m, sign(m, k), pk(k) \rightarrow true$ and equality axiom. It works for trace property but is not totally treated for observational equivalence. Nevertheless, we can notice that the composition of the *checksign* rule and the *equality* rule: $\dfrac{\dfrac{K^{\Downarrow}(sign(m,k)) \quad K^{\Uparrow}(pk(k))}{K^{\Downarrow}(m)} \qquad K^{\Uparrow}(m)}{}$ may be equivalent to the $verify$ rule: $\dfrac{K^{\Downarrow}(sign(m,k)) \quad K^{\Uparrow}(m) \quad K^{\Uparrow}(pk(k))}{}$.

References

[1] https://team.inria.fr/pesto/.

[2] www.inria.fr.

[3] www.larousse.fr.

[4] www.loria.fr.

[5] David Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[6] Qingfeng Chen, Chengqi Zhang, and Shichao Zhang. *Secure transaction protocol analysis: models and applications*. Springer-Verlag, 2008.

[7] Stéphanie Delaune and Francis Klay. Vérification automatique appliquéea un protocole de commerce électronique. *Actes des 6emes Journées Doctorales Informatique et Réseau (JDIR'04)*.

[8] Danny Dolev and Andrew Yao. On the security of public key protocols. *Transactions on information theory*, 1983.

[9] Jannik Dreier. *Formal Verification of Voting and Auction Protocols : From Privacy to Fairness and Verifiability*. Theses, Université de Grenoble, November 2013.

[10] Jannik Dreier and Ali Kassem Pascal Lafourcade. Formal analysis of e-cash protocols. In *Proceedings of the 12th International Conference on Security and Cryptography, SECRYPT*, 2015.

[11] Santiago Escobar, Ralf Sasse, and José Meseguer. Folding variant narrowing and optimal variant termination. In *International Workshop on Rewriting Logic and its Applications*. Springer, 2010.

[12] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1992.

[13] Steve Kremer. Formal verification of cryptographic protocols. Invited tutorial, 7th School on Modelling and Verifying Parallel Processes (MOVEP'06), Bordeaux, France, 2006.

[14] Tatsuaki Okamoto. An electronic voting scheme. In *Advanced IT Tools*. Springer, 1996.

[15] Ralf Sasse. Protocol security properties. In *Lecture notes from ETH Zurich*, 2016.

[16] Benedikt Schmidt. *Formal analysis of key exchange protocols and physical protocols*. PhD thesis, 2012.

[17] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[18] William Stallings. *Cryptography and network security: principles and practices*. Pearson Education India, 2006.