Journal of Computer Security 0 (2021) 1
IOS Press

Automatic generation of sources lemmas in TAMARIN: towards automatic proofs of security protocols

Véronique Cortier [a], Stéphanie Delaune [b], Jannik Dreier [a,*], and Elise Klein [a]
[a] *Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France*
E-mails: veronique.cortier@loria.fr, jannik.dreier@loria.fr, elise.klein@inria.fr
[b] *Univ Rennes, CNRS, IRISA, France*
E-mail: stephanie.delaune@irisa.fr

Abstract. TAMARIN is a popular tool dedicated to the formal analysis of security protocols. One major strength of the tool is that it offers an interactive mode, allowing to go beyond what push-button tools can typically handle. TAMARIN is for example able to verify complex protocols such as TLS, 5G, or RFID protocols. However, one of its drawback is its lack of automation. For many simple protocols, the user often needs to help TAMARIN by writing specific lemmas, called "sources lemmas", which requires some knowledge of the internal behaviour of the tool.

In this paper, we propose a technique to *automatically* generate sources lemmas in TAMARIN. Following the intuition of manually written sources lemmas, our lemmas try to keep track of the origin of a term by looking into emitted messages or facts. We prove formally that our lemmas indeed hold, for arbitrary protocols that make use of cryptographic primitives that can be modelled with a subterm convergent equational theory (modulo associativity and commutativity). We have implemented our approach within TAMARIN. Our experiments show that, in most examples of the literature, we are now able to generate suitable sources lemmas automatically, in replacement of the hand-written lemmas. As a direct application, many simple protocols can now be analysed fully automatically, while they previously required user interaction.

Keywords: Formal Verification, Tamarin Prover

1. Introduction

Security protocols are notoriously subtle to design and analyse. Many different tools have been developed in order to detect flaws and prove security properties such as authentication, secrecy, or privacy. However, even a simple property like secrecy is undecidable in general [1]. Hence several tools focus on the analysis of a decidable fragment, e.g. by bounding the number of sessions (e.g. AVISPA [2], DeepSec [3]). But when considering wider classes of protocols, more general cryptographic primitives, and an unlimited number of sessions, one necessarily goes beyond the decidable fragment, possibly losing termination or even automation.

One popular tool in that direction is ProVerif [4], a push-button tool that has been able to analyse hundred of protocols including e.g. TLS 1.3 [5], the ARINC823 avionic protocol [6], or the Neuchâtel voting protocol [7]. However, ProVerif may fail to prove some protocols because of some internal approximations. In that case, the user must either simplify the model or just give up.

[*] Corresponding author. E-mail: jannik.dreier@loria.fr.

0926-227X/$35.00 © 2021 – IOS Press. All rights reserved.

Another approach has been developed in the tool TAMARIN [8]. One key feature of TAMARIN is that it provides an interactive mode: if the tool fails to automatically prove a property by itself, the user may help the tool, for example by writing intermediate lemmas, or by manually guiding the proof search. Thanks to this approach, TAMARIN supports many features that are typically out of reach of many tools (Diffie-Hellman, stateful protocols), and has been able to prove complex protocols such as 5G AKA [9] with exclusive or, group key agreement protocols [10], or Noise framework [11] with Diffie-Hellman keys.

However, the fact that TAMARIN is not fully automatic makes it more difficult to use, at least in the learning phase. In particular, TAMARIN fails to automatically prove some "simple" protocols of the literature such as the well-known Needham-Schroder protocol or the Denning-Sacco protocol. This is a barrier when teaching the tool for example at the university or in summer schools.

Automation in TAMARIN fails in particular if it encounters "partial deconstructions". To speed up the analysis, TAMARIN computes in advance, for each protocol and intruder fact, all possible origins (called *sources*) of these facts, which are then repeatedly used in later steps of the analysis. However, this pre-computation can stop in an incomplete stage if TAMARIN lacks sufficient information about the origins of some fact(s). In practice, as soon as TAMARIN encounters such a "partial deconstruction", it is unlikely that it will be able to prove any interesting property automatically. To solve the issue, the user needs to manually write a "sources lemma" to help TAMARIN. Unfortunately, this manual step has to be done for many protocols, even simple ones.

Our contribution. In this paper, we automate the generation of sources lemmas. The main idea is to provide a systematic analysis of the origins of a term in a protocol. Intuitively, either a term has been forged by the attacker, or it comes from an earlier step in the protocol. To avoid the exploration of too many cases, we base our analysis on "deepest protected" subterms (when such a subterm exists). We prove that the sources lemmas that we generate are indeed true. Our result holds for any protocol provided that the cryptographic primitives can be expressed as a convergent subterm theory (modulo associativity and commutativity) with the finite variant property. This is the case of most standard cryptographic primitives such as symmetric and asymmetric encryptions, as well as signatures.

A preliminary version of these results have been presented in [12]. However, we noticed that sometimes, reasoning on the origin of a term by following the message flow is not enough. Indeed, a term may be first stored in a protocol fact and then released later on. This happens for example for protocols with private channels or counters, as those are often modeled through dedicated facts. Hence in this paper, we have also developed a way to generate sources lemmas that reason on protocol facts when following messages is not enough.

Interestingly, the correctness of TAMARIN does not rely on the fact that we are able to prove that our sources lemmas hold. TAMARIN will verify them anyway (as done with sources lemmas written by the user). This means that our technique can also be used even in cases where our theoretical justification does not apply. Our theoretical justification simply explains why TAMARIN has a good chance to work. We have implemented our technique in TAMARIN, as a new option `--auto-sources`. With this option, when partial deconstructions are detected, a sources lemma is generated automatically and added to the original model, so that the user can see it and possibly amend it, if needed. We have validated our approach with two kind of experiments.

- First, we consider simple protocols of the literature, used as benchmarks for most tools. We modelled a handful of them and ran TAMARIN. Our approach is able to solve all partial deconstructions.

Actually, we found out that for these simple examples, this was the only reason they were not entirely automatic, hence thanks to our `--auto-sources` option, TAMARIN can now analyse all these examples automatically.

- We also wanted to evaluate how our technique behaves on more complex protocols and on protocols that have not been specified by ourselves. Hence we considered all the models provided within TAMARIN's distribution, and that contained "partial deconstructions" and "sources lemmas". For a majority of them, our technique successfully close all partial deconstructions and for about a half of these, TAMARIN is now even able to analyse the whole protocol automatically.

Unsurprisingly, complex protocols still require the existing manually written intermediate lemmas or tailored heuristics. However, our technique considerably improves the degree of automation of TAMARIN, yielding a better trade-off between what can be done automatically, and what needs to be done manually.

Related work. The TAMARIN prover was first presented in 2012 [13]. Since then, work on the tool mostly focused on widening its scope and adding new features. In particular, support for various equational theories has been added (bilinear pairing [14], user-defined convergent equational theories [15], and exclusive-or [16, 17]), as well as the possibility to perform equivalence proofs [18]. There also have been various extensions, e.g., for different input languages (e.g., [19, 20]). Concerning improved automation, there is prior work aiming at automatically learning proof heuristics [21]. To the best of our knowledge, this work, initially presented at ESORICS 2020 [12], is the first to focus on the automatic generation of sources lemmas.

In the context of symbolic models, several other tools have been developed for analysing the security of protocols and most of them are fully automated like AVISPA [2], Maude-NPA [22, 23], DeepSec [3], or SPEC [24], at the cost of bounding the number of sessions or limiting the expressivity of the tool w.r.t. covered primitives or properties. The main competitor of TAMARIN prover is ProVerif [4], that is fully automated but cannot handle primitives like exclusive-or.

2. Overview

We illustrate our technique on a simple challenge-response protocol.

$$I \rightarrow R : \{\mathsf{req}, I, n\}_{\mathrm{pk}(R)}$$
$$R \rightarrow I : \{\mathsf{rep}, n\}_{\mathrm{pk}(I)}$$

The initiator sends a nonce n encrypted with the public key of the responder, and then waits for the corresponding answer, i.e. the nonce n encrypted with his own public key. The symbols req and rep are constants used to avoid confusion between the two types of messages: they indicate whether the ciphertext corresponds to a request or a reply. The full TAMARIN input file modeling this protocol is given in Appendix. In particular, the responder role is as follows:

```
rule Rule_R:
 [ In(aenc{'req', I, x}pk(ltkR)), !Ltk(R, ltkR), !Pk(I, pkI) ]
 --[]->  [ Out(aenc{'rep', x}pkI) ]
```

Intuitively, at the reception of a message of the form `aenc{'req', I, x}pk(ltkR)`, the agent `R` (with private key `ltkR`) sends the message `aenc{'rep', x}pkI` on the network to the agent `I` (with public key `pkI`). Note that there are other rules modeling the Initiator role, as well as the key generation. The latter rule creates the `!LtK` and `!Pk` facts used here to retrieve the agents' public and private keys.

This protocol rule models the behavior of the responder role. It can be triggered arbitrary many times, possibly with different values for `x`. When loading this model in TAMARIN, it turns out that the proof attempt of e.g. a simple secrecy property of nonce *n* does not terminate due to partial deconstructions. In TAMARIN's interactive interface, they are identified by dashed green arrows as shown in Figure 1. The green arrow symbolizes a *deconstruction chain*. Deconstruction chains are used in TAMARIN's intruder reasoning to extract values from messages output by the protocol. In this example, TAMARIN tries to extract a fresh value from the message output by the rule `Rule_R` (at the top). TAMARIN has computed that if it can decrypt the output of the rule (rule `d_0_adec`) and then extract the second term (rule `d_0_snd`), it obtains the value x.7 (a renaming of the variable `x` given in the initial rule definition). However, here TAMARIN is unable to continue its deconstruction, as x.7 can potentially be any value: directly the desired fresh value, or a pair of values, or an encryption, or something completely different. As this deconstruction is incomplete, it is called a *partial deconstruction*.

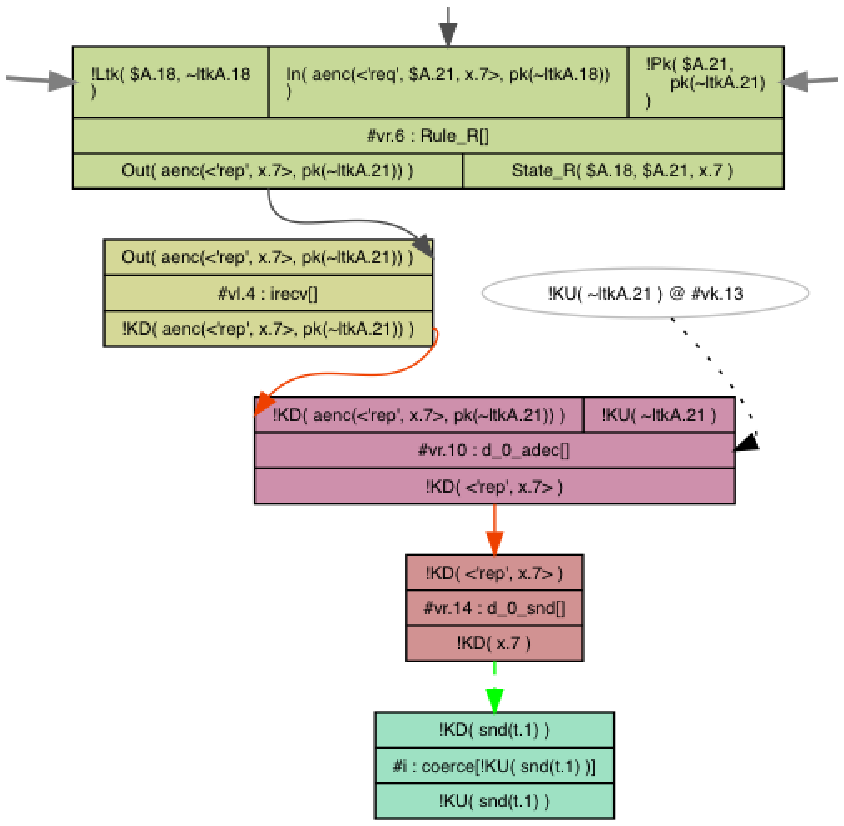
!Ltk($A.18, ~ltkA.18)
In(aenc(<'req', $A.21, x.7>, pk(~ltkA.18)))
!Pk($A.21, pk(~ltkA.21))
#vr.6 : Rule_R[]
Out(aenc(<'rep', x.7>, pk(~ltkA.21)))
State_R($A.18, $A.21, x.7)
Out(aenc(<'rep', x.7>, pk(~ltkA.21)))
#vl.4 : irecv[]
!KD(aenc(<'rep', x.7>, pk(~ltkA.21)))
!KU(~ltkA.21) @ #vk.13
!KD(aenc(<'rep', x.7>, pk(~ltkA.21)))
!KU(~ltkA.21)
#vr.10 : d_0_adec[]
!KD(<'rep', x.7>)
!KD(<'rep', x.7>)
#vr.14 : d_0_snd[]
!KD(x.7)
!KD(snd(t.1))
#i : coerce[!KU(snd(t.1))]
!KU(snd(t.1))

Fig. 1. Example of a partial deconstruction

In the above example, TAMARIN does not know anything about the contents of the variable x.7, hence, to ensure soundness, it is obliged to consider this case as a potential source for any value, which leads to an explosion of the number of cases, and often to non termination issues. This is the case here: the rule

`Rule_R` producing the x.7 requires an input, which could itself be the result of (a different instantiation of) the same source, and so on.

To get rid of partial deconstructions, TAMARIN uses *sources lemmas*. They are a special type of lemmas which are applied at the precomputation phase. More precisely, after computing the initial *raw sources* without any lemmas, TAMARIN computes the *refined sources* using the sources lemmas to hopefully discard partial deconstructions. To ensure that the refined sources are correct, one further has to prove the sources lemmas correct, using only the raw sources. This can be done either automatically by TAMARIN or manually in the interactive mode.

The idea behind a sources lemma is to provide more information regarding the origin of the message mentioned in the partial deconstruction, i.e., the one corresponding to the variable identified by the dashed green arrow. Going back to our example and assuming that:

(1) `R(aenc{'req', I, x}pk(ltkR), x)` is added as a label to the responder rule; and
(2) `I(aenc{'req', I, n}pkR)` is added as a label to the initiator rule,

a sources lemma could be as follows:

```
lemma typing [sources]:
  "All x m #i. R(m,x)@#i ==> ( (Ex #j. I(m)@#j & #j < #i)
                              |(Ex #j. KU(x)@#j & #j < #i ))"
```

This lemma says that whenever the responder receives the value `x` inside a message `m` (at time point `#i`), either this message (actually a ciphertext) has been forged by the attacker who therefore knew `x` before, denoted `KU(x)`, or it has been produced (for the first time) by another protocol rule, here the one denoted `I(m)`. Indeed, a quick inspection of the protocol shows that here this is the only option to produce an output having the right format.

When generating the refined sources from the raw sources, TAMARIN applies the sources lemmas. In this case, the sources lemma above will allow it to learn that x is either a nonce (generated by the initiator role) or a message already known by the attacker. This solves the partial deconstruction as the previous sources will be refined into two refined sources. The first one is the case where the intruder learns the nonce generated by the initiator, by passing the initiator's message to the responder, and then extracting the nonce like the variable x.7 above. However, TAMARIN now knows that x.7 is not any value, but the initiator's nonce. The second case will be discarded by TAMARIN since, if the intruder already knew x before, it is useless to extract it again.

3. TAMARIN syntax and semantics

We explain here the syntax and semantics of TAMARIN, as presented in [13, 16], as necessary background for the remainder of the paper.

3.1. Term algebra

Cryptographic messages are represented by a (sorted) term algebra. In TAMARIN, terms are all of sort *msg* and there are two incomparable subsorts *fr* and *pub* used to represent respectively fresh names (e.g. nonces or keys) and public names (e.g. agent names). We assume an infinite set $\mathcal{N}$ of names of each sort and an infinite set $\mathcal{V}$ of variables of each sort as well. A variable x of sort s is denoted $x : s$. The sort *msg* is often omitted, that is, the variable x typically denotes a variable of sort *msg*. Each cryptographic primitive is represented by a function symbol $\mathsf{f} : s_1 \times \cdots \times s_n \to s$ that takes n arguments of sort resp. $s_1, \ldots, s_n$ and returns a term of sort s. We assume given a *signature* Σ, i.e. a set of function symbols with their arities. Then the set of terms is built from the application of symbols of Σ to names and variables and is denoted $T_\Sigma(\mathcal{N}, \mathcal{V})$. The set of variables occurring in a term t is denoted $vars(t)$. A term is *ground* if it contains no variable. A substitution θ is *grounding for t* if $t\theta$ is ground.

Example 1. *The standard primitives are often expressed by the signature*

$$\Sigma_{\mathsf{stand}} = \{enc(\_,\_), dec(\_,\_), encA(\_,\_), decA(\_,\_), pk(\_), \langle\_,\_\rangle, fst(\_), snd(\_)\}$$

where all functions are of sort msg $\times \cdots \times$ *msg* $\to$ *msg.*

They model respectively symmetric encryption and decryption, asymmetric encryption and decryption, and concatenation and (left and right) projections.

The properties of the primitives are reflected through an equational theory E. In TAMARIN, user defined equational theories are given as a convergent rewrite system. TAMARIN additionally supports built-in theories such exclusive or [16] and a set of equations for Diffie-Hellman (DH) exponentiation [13]. The equality modulo associativity and commutativity (AC) is denoted $=_{AC}$ and the normal form of a term t, modulo AC, is denoted $t{\downarrow}$ (we consider any representative of the normal form of t). Two terms t_1 and t_2 are *unifiable* (modulo AC) if there exists a substitution θ such that $t_1\theta =_{AC} t_2\theta$. *Positions* of a term t are defined as usual considering AC operators as binary symbols. A *subterm* of t is a term t' such that $t' = t|_p$ for some position p.

TAMARIN assumes equational theories that have the finite variant property [25], that is where all the instances of a given term follow a finite number of different patterns. Formally, a convergent equational theory E has the *finite variant property* if for any term t, there exists a finite number of substitutions $\sigma_1, \ldots, \sigma_k$ such that, for any substitution θ, there is $1 \leqslant i \leqslant k$, there exists a substitution θ' such that $(t\theta){\downarrow} =_{AC} t\sigma_i\theta'$. A particular class of rewriting systems is the class of *subterm* rewriting system. A rewriting system is said subterm if it is defined by a set of equations of the form $l \to r$ such that r is a subterm of l or a (public) constant. Many cryptographic primitives can be modeled by (convergent) subterm rewriting systems, such as signatures, symmetric and asymmetric encryption, pair, hash, etc. Our theoretical development only consider equational theories that can be defined by a subterm rewriting system, convergent modulo AC, that have the finite variant property. TAMARIN is not limited to subterm equational theories, and actually our approach can be applied in this general setting too, relying on Tamarin to establish the correctness of the generated lemmas.

Example 2. *Orienting from left to right the equations below yields a subterm convergent rewrite system that is usually used to model concatenation and asymmetric encryption. Here, there is no AC symbol.*

$$decA(encA(x, pk(y)), y) = x \quad fst(\langle x, y\rangle) = x \quad snd(\langle x, y\rangle) = y$$

In what follows, we will consider sets and multisets. Given a multiset S, $set(S)$ denotes the set of its elements. The symbol $\subseteq$ denotes the set inclusion. We will write $S \subseteq S'$ even if S and S' are multisets, which is then interpreted as $set(S) \subseteq set(S')$. In contrast, $\subseteq^\sharp$ denotes the multiset inclusion. Similarly, $\cup^\sharp$ denotes the multiset union and $\setminus^\sharp$ the multiset difference.

3.2. Transition system

In TAMARIN, a protocol execution is modeled as a transition system where a state contains a multiset of facts, representing the current knowledge of the attacker and the current steps of the protocol, for each agent and each session. Formally, we assume a set of *fact symbols* $\mathcal{F}$ partitioned into *linear* and *persistent* fact symbols. A *fact* is an expression $\mathsf{F}(t_1, \ldots, t_n)$ where $\mathsf{F} \in \mathcal{F}$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{N}, \mathcal{V})$. Given a multiset of facts F, $\mathit{lfacts}(F)$ denotes the multiset of its linear facts while $\mathit{pfacts}(F)$ denotes the multiset of its persistent facts.

Linear facts represent resources that are consumed. TAMARIN includes three pre-defined linear fact symbols: $\mathsf{Fr}(n)$ models the generation of a fresh name n, $\mathsf{Out}(m)$ represents a message m sent over the network by a participant, and $\mathsf{In}(m)$ denotes that the adversary has sent message m, that can then be received by an agent of the protocol. Persistent facts represent facts that remain forever and are not consumed by rules. TAMARIN includes the persistent fact symbol K that models the knowledge of the attacker, as well as $\mathsf{K}^\uparrow$ and $\mathsf{K}^\downarrow$ that allow to distinguish between the terms built by the attacker and those obtained from listening to the network or by decomposing learned messages. Then the protocol may use other user defined facts, that can be either linear or persistent. The latter are denoted using the ! operator in front of the fact.

The protocol execution is specified through labelled multiset rewriting rules $[l]{-\!\![}\, a \,]\!\!\rightarrow [r]$ where l, a, r are multisets of facts. The multiset l denotes the *premises* of the rule that need to be present in the state in order for the rule to be executed; a denotes the *actions* of the rule (later used to specify properties), while r contains the *conclusions*, added to the state. There are three kinds of rules.

*3.2.1. Fresh name generation (*FRESH*)*

This is the only rule that can produce facts of the form $\mathsf{Fr}(n)$. Moreover, to ensure freshness, a distinct name n is used for each application.

$$[\,]{-\!\![}\,]\!\!\rightarrow [\mathsf{Fr}(x : \mathit{fr})]$$

*3.2.2. Message deduction rules (*MD*)*

They are pre-defined in TAMARIN and represent the attacker's actions.

$$[\mathsf{Out}(x)]{-\!\![}\,]\!\!\rightarrow [\mathsf{K}^\downarrow(x)] \quad \text{and} \quad [\mathsf{K}^\uparrow(x)]{-\!\![}\, \mathsf{K}(x) \,]\!\!\rightarrow [\mathsf{In}(x)]$$

model the fact that the attacker can learn any message sent by the protocol and conversely, may send any message of her knowledge. Note that this is the only rule where the predicate K appears as an action of a rule. The rules

$$[\,]{-\!\![}\, \mathsf{K}^\uparrow(x) \,]\!\!\rightarrow [\mathsf{K}^\uparrow(x : \mathit{pub})] \quad \text{and} \quad [\mathsf{Fr}(x : \mathit{fr})]{-\!\![}\, \mathsf{K}^\uparrow(x) \,]\!\!\rightarrow [\mathsf{K}^\uparrow(x : \mathit{fr})]$$

express respectively that the attacker can learn any public name and can create fresh name on his own. Finally, the attacker can extend his knowledge by applying function symbols. The intuitive rule is:

$$[\mathsf{K}(x_1), \ldots, \mathsf{K}(x_n)]{-\!\![}\,]\!\!\rightarrow [\mathsf{K}(\mathsf{f}(x_1, \ldots, x_n))] \quad \text{for any} \quad \mathsf{f} \in \Sigma$$

Actually, this rule is split into two cases in TAMARIN, depending on whether the attacker is building a term, or decomposing it. Formally, for any substitution θ (in normal form), we consider the rule

$$[\mathsf{K}^\uparrow(x_1\theta), \ldots, \mathsf{K}^\uparrow(x_n\theta)]{-\!\![}\, \mathsf{K}^\uparrow(\mathsf{f}(x_1, \ldots, x_n)\theta) \,]\!\!\rightarrow [\mathsf{K}^\uparrow(\mathsf{f}(x_1, \ldots, x_n)\theta)]$$

when $\mathsf{f}(x_1,\dots,x_n)\theta$ is in normal form. When the term $\mathsf{f}(x_1,\dots,x_n)\theta$ reduces to a subterm of $x_{i_0}\theta$ for some i_0 (remember that we only consider subterm theories), then we consider

$$[\mathsf{K}^{\alpha_1}(x_1\theta),\dots,\mathsf{K}^{\alpha_n}(x_n\theta)] \!-\!\![\ \mathsf{K}^{\downarrow}(\mathsf{f}(x_1,\dots,x_n)\theta\downarrow)\ ]\!\!\rightarrow [\mathsf{K}^{\downarrow}(\mathsf{f}(x_1,\dots,x_n)\theta\downarrow)]$$

where $\alpha_i = \uparrow$ for all $i \neq i_0$ and $\alpha_{i_0} = \downarrow$. Intuitively, the deduction rule is annotated with $\mathsf{K}^{\uparrow}$ when the attacker applies a "constructor" term such as an encryption and a pair. It can also be annotated with $\mathsf{K}^{\uparrow}$ when the attacker applies a deconstructor (for example, a decryption), if the term cannot be further reduced (for example, the decryption fails). Conversely, the deduction rule is annotated with $\mathsf{K}^{\downarrow}$ when the attacker decomposes a term. Finally, it is possible to switch from $\mathsf{K}^{\downarrow}$ to $\mathsf{K}^{\uparrow}$ thanks to the "coerce" rule:

$$[\mathsf{K}^{\downarrow}(m)] \!-\!\![\ \mathsf{K}^{\uparrow}(m)\ ]\!\!\rightarrow [\mathsf{K}^{\uparrow}(m)]$$

for any message m in normal form that is not a pair.

3.2.3. Protocol rules

Then the protocol as well as additional attacker capabilities are specified through protocol rules, that are multiset rewriting rules that satisfy some conditions.

Definition 1. *A* protocol rule *is a multiset rewriting rule $[l] \!-\!\![\ a\ ]\!\!\rightarrow [r]$ such that*

(1) *it does not contain fresh names and* Fr *does not occur in r*
(2) K, $\mathsf{K}^{\uparrow}$, $\mathsf{K}^{\downarrow}$, *and* Out *do not occur in l*
(3) K, $\mathsf{K}^{\uparrow}$, $\mathsf{K}^{\downarrow}$, In *do not occur in r*
(4) $vars(r) \subseteq vars(l) \cup \{x \in \mathcal{V} \mid x : pub\}$.

The first condition guarantees in particular that fresh names are only produced thanks to the fresh name generation rule. The last three rules are easily met by any rule modeling a protocol step.

Example 3. *Going back to our running example, the rule given in Section 2 is a protocol rule where Ltk and Pk are user-defined persistent facts used to model generation of long-term keys. Actually, our model contains the following rule:*

$$[\mathit{Fr}(xsk)] \!-\!\![]\!\!\rightarrow [!Ltk(xid, xsk), !Pk(xid, pk(xsk)), \mathit{Out}(pk(xsk))]$$

where xsk is variable of sort fr, and xid is a variable of sort pub. This protocol rule represents the possibility to generate key pairs $(xsk, pk(xsk))$ for any identity xid. The public part of the key is revealed to the attacker.

3.3. Execution traces

A set of protocol rules P induces a transition relation $\rightarrow_P$ between states. We have $S \overset{set(a\theta)}{\rightsquigarrow_P} S'$ if there exists a rule $ru \in P \cup \mathsf{MD} \cup \{\mathsf{Fresh}\}$ of the form $[l] \!-\!\![\ a\ ]\!\!\rightarrow [r]$ and a grounding substitution θ such that

- $lfacts(l\theta) \subseteq^{\sharp} S$, the linear facts of $l\theta$ should be present in S, with enough occurrences,

- $pfacts(l\theta) \subseteq S$,
- and $S' = (S \setminus^{\#} lfacts(l\theta)) \cup^{\#} r\theta$. The linear facts of $l\theta$ are removed and all the conclusion facts are added to the state.

Moreover, if the applied rule is the FRESH rule then $r\theta = \{\mathsf{Fr}(n)\}$ and n must be a new name not used earlier. The execution of a protocol is simply modeled by a sequence of transitions. A *trace* of a protocol is the sequence of actions that appear in the execution. Formally, we have that:

$$traces(P) = \{[A_1, \ldots, A_n] \mid \emptyset \rightsquigarrow_P^{A_1} \cdots \rightsquigarrow_P^{A_n} S'\}.$$

Example 4. *Continuing Example 3, the protocol rule modeling key generation can be used twice (or even more) to generate two key pairs for two different identities leading to the following trace:*

$$\begin{aligned}
&\{\} \rightsquigarrow \{\mathit{Fr}(ska)\} \rightsquigarrow F_a \cup \{\mathit{Out}(pk(ska))\} \\
&\quad \rightsquigarrow \{\mathit{Fr}(skb)\} \rightsquigarrow F_a \cup F_b \cup \{\mathit{Out}(pk(ska)), \mathit{Out}(pk(skb))\} \\
&\quad \rightsquigarrow F_a \cup F_b \cup \{K^{\downarrow}(pk(ska)), \mathit{Out}(pk(skb))\}
\end{aligned}$$

where $F_a = \{!Ltk(A, ska), !Pk(A, pk(ska))\}$, $F_b = \{!Ltk(B, skb), !Pk(B, pk(skb))\}$. Here ska and skb are names of sort fr whereas A, B are public names of sort pub. This corresponds to the application of the FRESH *rule followed by the protocol rule to obtain key material for the first agent A and then for a second agent B. The last rule corresponds to an application of an* MD *rule adding the public key of A to the knowledge of the attacker.*

3.4. Properties

Security properties are expressed as properties on the traces of a protocol. TAMARIN offers a first order logic to specify properties. Formulas make use of variables of a novel sort *temp* to reason about when a fact occurs and to be able to express that some event occurs before another one. The full syntax and semantics of the logic is provided in [13]. We provide here only informally the semantics of atomic formulas:

- $F@i$, where i is of sort *temp*, refers to the fact F that occurs in the i^{th} element of the trace;
- $i \doteq j$ expresses that the timepoints i and j are equal;
- $i \lessdot j$ expresses that timepoint i occurs before j;
- $t_1 \approx t_2$ says that t_1 and t_2 are equal (modulo the equational theory).

The first order logic is built from atomic formulas and closed by the boolean connectors $\vee$, $\wedge$, and $\neg$, as well as the quantifications $\exists$ and $\forall$.

A set of protocol rules P *satisfies* a formula ϕ, denoted $P \models \phi$ if, for any trace $tr \in traces(P)$, then tr satisfies ϕ.

Example 5. *Continuing the running example, a typical lemma expressing nonce secrecy of the challenge is as follows:*

```
lemma nonce_secrecy:
  "not(Ex A B s #i #j. (SecretI(A, B, s)@#i  &  K(s)@#j))"
```

This requires us to annotate the rule of the Initiator role with the action fact `SecretI`*. Then intuitively this lemma expresses that there does not exit any trace such that* `SecretI(A,B,s)` *occurs at stage i (for some A, B, and s) and the attacker knows s at stage j. If we consider only the three protocol rules mentioned so far (initiator's rule, responder's rule, and key generation), then this security property is satisfied. However, as expected, the same lemma is not satisfied as soon as we model corruption, for example with the following rule.*

```
rule Reveal_ltk: [!Ltk(xid, xsk)] --[RevLtk(xid)]-> [Out(xsk)]
```

TAMARIN also allows to express diff-equivalence, a refined notion of equivalence. This can be used for example to state that a protocol preserves unlinkability, anonymity, or other privacy properties such as ballot privacy. For example, the fact that Alice remains anonymous is often expressed as the property that $P(Alice) \sim P(Bob)$. This intuitively says that an adversary should not see the difference when Alice is playing protocol P or Bob is playing protocol P. The formal definition of diff-equivalence can be found in [13]. We do not need to provide it here as our automatically generated lemmas are simple trace properties and do not use diff-equivalence. Note however that our approach applies to protocols with diff-equivalence as well since our generated lemmas also helps TAMARIN to terminate in the case of diff-equivalence properties.

4. Automatically generated sources lemmas

Whenever TAMARIN fails to complete a deconstruction, we aim at providing the tool with a sources lemma that resolves the partial deconstruction. We formalise here our approach and prove it to be correct.

4.1. Definitions

We introduce the notion of *protected* term, which is any term that is headed by a function symbol that is not a pair (because we know the adversary can always open such terms) nor an *AC* symbol (simply because our heuristic does not apply to case of failures due to an *AC* theory).

Definition 2. *A* protected term *t is a term whose head symbol is not $\langle \_, \_ \rangle$ nor an AC symbol. Given a term t and a variable x occurring in t, we say that t' is a* deepest protected subterm w.r.t. x *if t' is a protected term, subterm of t that contains x and such that one of the paths from the root of t' to x contains only pair symbols $\langle \_, \_ \rangle$ (except for head symbol at top level).*

Intuitively, if t' is a deepest protected subterm w.r.t. x, then the only way to obtain t' is either by extracting it directly from some output, or by building it, in which case x is already known to the attacker.

Example 6. *Let $t = enc(\langle x, enc(\langle b, x\rangle, k_2)\rangle, k_1)$. There are two deepest protected subterms w.r.t. x, namely t itself and $t' = enc(\langle b, x\rangle, k_2)$.*

We denote by $St_{\mathsf{pair}}(u)$ the set of subterms of u that can be obtained from u simply by projecting. Formally, $St_{\mathsf{pair}}(u)$ is formally defined as

$$St_{\mathsf{pair}}(u) = \begin{cases} \{u\} \cup St_{\mathsf{pair}}(u_1) \cup St_{\mathsf{pair}}(u_2) & \text{if } u = \langle u_1, u_2 \rangle \\ \{u\} & \text{otherwise} \end{cases}$$

Normalised traces. In order to keep track of the origin of a protected subterm, we need to assume that the shape of a term is not modified by the application of the equational theory. Fortunately, since we assume an equational theory with the finite variant property, it is possible to compute in advance the shapes of all the terms obtained after normalisation. Given a set of protocol rules P, TAMARIN computes the variants $\mathsf{Variant}(P)$ of P such that, for any rule $ru \in P$, for any substitution θ, there is $ru' \in \mathsf{Variant}(P)$ and a substitution θ' such that $ru\theta =_E ru'\theta'$ and (ru', θ') is *normalised*, that is, for any fact $F(u')$ occurring in ru', we have that $(u\theta')\downarrow =_{AC} u'\theta'$. Moreover, $ru' = (ru\sigma)\downarrow$ for some σ. TAMARIN considers only traces that are *normalised*, i.e. executions of the form $\emptyset \rightsquigarrow^{A_1}_{\mathsf{Variant}(P)} S_1 \cdots \rightsquigarrow^{A_n}_{\mathsf{Variant}(P)} S_n$ and such that:

- the execution involves only rules $ru \in \mathsf{Variant}(P)$ and substitutions θ such that (ru, θ) is normalised;
- pairs are always decomposed before being used, that is, if $\mathsf{K}^{\uparrow}(u)$ appears in A_i then $\mathsf{K}^{\uparrow}(t) \in S_{i-1}$ for any $t \in St_{\mathsf{pair}}(u)$[1].

We write $P \models_{\mathsf{norm}} \phi$ if for any normalised trace tr of P, tr satisfies ϕ. Then, given a formula ϕ that does not contain the fact $\mathsf{K}^{\uparrow}$ nor $\mathsf{K}^{\downarrow}$, we have $P \models \phi$ if, and only if, $P \models_{\mathsf{norm}} \phi$, which is what is actually checked by TAMARIN. This follows from the soundness of TAMARIN [13].

In some cases, computing the variants $\mathsf{Variant}(ru)$ of a protocol rule ru may introduce new variables on the right of the rule, and thus lead to rules that are not protocol rules (according to Definition 1).

Example 7. *The rule $[\mathit{In}(decA(x,y))]{-}[]{\rightarrow}[\mathit{Out}(x)]$ is a protocol rule. However, one of its variant is $[\mathit{In}(z)]{-}[]{\rightarrow}[\mathit{Out}(encA(z, pk(y)))]$ which is not a protocol rule according to Definition 1.*

However, such cases correspond to badly defined protocols and TAMARIN typically raises a warning in this case. Hence, in what follows, we consider *well-formed protocol rules P*, that is such that $\mathsf{Variant}(P)$ is still a set of protocol rules. In practice, protocol rules representing a protocol are indeed well-formed.

4.2. Algorithm

Given a set P of protocol rules, TAMARIN first computes its variants $\mathsf{Variant}(P)$. It then precomputes sources as already explained. Whenever TAMARIN fails to complete a deconstruction, it returns the partial deconstruction. For the moment, assume that we can extract a rule $ru = [l]{-}[\, a \,]{\rightarrow}[r]$ of $\mathsf{Variant}(P)$ and a variable x for which the deconstruction has failed. In practice there might be multiple composed rules, as explained in Section 7.1, but the approach is similar. It must be the case that x appears in some fact of l.

For each deepest protected subterm t occurring in a rule of P, we assume new fact symbols Left_t and Right_t that will be used to further annotate the rules of $\mathsf{Variant}(P)$. These facts will appear only in the sources lemmas we generate.

The sources lemma $\mathsf{SourceLemma1}(P, ru, x)$ associated to a partial deconstruction on variable x and rule ru for protocol P is defined by Algorithm 1. Intuitively, we first look for any occurrence of x in the premisses of ru, under a (deepest) protected term t_1 and we annotate the rule ru with $\mathsf{Left}_{t_1}(t_1, x)$. Then we look for all facts in the conclusions of a rule that may have produced t_1, that is that contain a term t_2 that can be unified with t_1 and we annotate those rules with $\mathsf{Right}_{t_1}(t_2)$. Finally, we generate the formula that says that if we have $\mathsf{Left}_{t_1}(y, x)$ at some step i, then either x is already known to the attacker, that is

[1]This comes from the fact that, whenever the attacker learns a pair $\mathsf{K}^{\downarrow}(\langle m_1, m_2\rangle)$, she cannot directly convert it in $\mathsf{K}^{\uparrow}(\langle m_1, m_2\rangle)$ since the coerce rule does not apply to terms headed with a pair. Hence it is necessary to decompose it first (with $\mathsf{K}^{\downarrow}$ rules) and then reconstruct it (with $\mathsf{K}^{\uparrow}$ rules).

$\mathsf{K}(x)$ holds at an earlier step, or y has been obtained from the protocol, that is $\mathsf{Right}_{t_1}(y)$ holds at some earlier step.

Algorithm 1 $\mathsf{SourceLemma1}(P, ru, x)$

```
Input: $P,\ ru = [l]\!-\![\ a\ ]\!\!\rightarrow\![r],\ x$
  $\Psi := \top$
  for all $t_1$ deepest protected term w.r.t. $x$ that is subterm of $v_i$ for some $F(v_1, \dots, v_n) \in l$ do
    % we annotate $ru$ with the fact that $x$ may provide from $t_1$
    $a := a \cup \{\mathsf{Left}_{t_1}(t_1, x)\}$
    % then we identify from which facts $t_1$ may provide.
    for all rule $ru' = [l']\!-\![\ a'\ ]\!\!\rightarrow\![r'] \in P$ do
      if $t_1$ unifiable with $t_2$ modulo $AC$ for some $t_2$ protected subterm in $F'(v') \in r'$ then
        % we annotate $ru'$ with the fact that $t_2$ may be used to produce $x$
        $a' := a' \cup \{\mathsf{Right}_{t_1}(t_2)\}$
      end if
    end for
    Let $\phi$ the formula defined as follows
        $$\forall y, x, i\ \mathsf{Left}_{t_1}(y, x)@i \implies \big((\exists k\ \mathsf{Right}_{t_1}(y)@k\ \wedge\ k \lessdot i) \vee (\exists k\ \mathsf{K}^{\uparrow}(x)@k\ \wedge\ k \lessdot i)\big)$$
    $\Psi := \Psi \wedge \phi$
  end for
  return $\Psi$
```

Example 8. *Going back to our running example developped in Section 2, our algorithm* $\mathsf{SourceLemma1}$ *will be applied on the rule* `Rule_R` *and the variable* `x`*.*

```
rule Rule_R:
 [ In(aenc{'req', I, x}pk(ltkR)), !Ltk(R, ltkR), !Pk(I, pkI) ]
 --[]->  [ Out(aenc{'rep', x}pkI) ]
```

The only deepest protected subterm w.r.t. `x` *is* `aenc{'req', I, x}pk(ltkR)` *and it plays the role of t_1. Therefore, we annotate this rule with $\mathsf{Left}_{aenc\{'req',I,x\}pk(ltkR)}(aenc\{'req', I, x\}pk(ltkR), x)$, and then we identify which fact may provide t_1. Actually the only candidate is the encrypted term $t_2 = aenc\{'req', I,\ n\}pkR$ coming from the* `Rule_I` *given below:*

```
rule Rule_I:
    [ Fr(~n), !Pk(R, pkR),!Ltk($I, ltkI)]
  --[SecretI($I,R,~n)]->
    [ Out(aenc{'req',I, ~n}pkR)]
```

Therefore, an annotation $\mathsf{Right}_{aenc\{'req',I,x\}pk(ltkR)}(aenc\{'req', I,\ n\}pkR)$ is added to this rule. The resulting formula ϕ is exactly the one given by the algorithm, i.e.:

$$\forall y, x, i\ \mathsf{Left}_{aenc\{'req',I,x\}pk(ltkR)}(y, x)@i \implies \big((\exists k\ \mathsf{Right}_{aenc\{'req',I,x\}pk(ltkR)}(y)@k\ \wedge\ k \lessdot i)$$
$$\vee\ (\exists k\ K^{\uparrow}(x)@k\ \wedge\ k \lessdot i)\big)$$

In the implementation, since it is not possible to use terms as indices of a fact, we use the position of the term t_1 inside the rule `Rule_R` *as part of the fact name. The formula above alone allows us to get rid of the 6 partial deconstructions produced on this simple example, and the secrecy lemma, as well as the sources lemma itself, are proved automatically by Tamarin.*

5. Soundness of our algorithm

We can show that under our assumptions the generated sources lemmas always hold, which explains why TAMARIN is usually able to prove them.

Theorem 1. *Given a set of well-formed protocol rules P, a rule $ru \in \mathsf{Variant}(P)$, a variable x occurring in ru, and $\Psi = \bigwedge_{i=1}^{n} \phi_i$ be the conjunction of formulas returned by $\mathsf{SourceLemma1}(\mathsf{Variant}(P), ru, x)$, then for any $i \in \{1, \ldots, n\}$, we have that ϕ_i is satisfied by $\mathsf{Variant}(P)$, that is $\mathsf{Variant}(P) \models_{\mathsf{norm}} \phi_i$.*

Proof. Let P be a set of protocol rules, $ru \in \mathsf{Variant}(P)$, and x a variable occurring in ru. Let $\Psi = \bigwedge_{i=1}^{n} \phi_i$ be the formula returned by $\mathsf{SourceLemma1}(\mathsf{Variant}(P), ru, x)$, and let $i_0 \in \{1, \ldots, n\}$. The rule ru is of the form $[l]\!-\!\![\ a\ ]\!\!\rightarrow[r]$ and ϕ_{i_0} is of the form:

$$\forall \tilde{y}, \tilde{x}, i\ \mathsf{Left}_{t_1}(\tilde{y}, \tilde{x})@i \implies \big(\exists k\ \mathsf{Right}_{t_1}(\tilde{y})@k\ \wedge\ k \lessdot i\big) \vee \big(\exists k\ \mathsf{K}^{\uparrow}(\tilde{x})@k\ \wedge\ k \lessdot i\big)$$

for some t_1 deepest protected term w.r.t. x. By definition of a deepest protected subterm, $t_1|_{p_0} = x$ for some position p_0 and there are only pairs along the path p_0 (except at position ϵ).

Let tr be a normalised trace of $\mathsf{Variant}(P)$. Let us show that tr satisfies ϕ_{i_0}.

$$tr = \emptyset \rightsquigarrow^{A_1} S_1 \cdots \rightsquigarrow^{A_{n-1}} S_{n-1} \rightsquigarrow^{A_n} S_n$$

Let i be such that $\mathsf{Left}_{t_1}(m, n) \in A_i$ for some terms m, n. Then the i^{th} applied rule must be a rule ru' in $\mathsf{Variant}(P)$ such that t_1 is a subterm of some t' with $t_1|_{p'_0} = x'$ for some p'_0 and there are only pairs along the path p'_0 (except at position ϵ):

$$ru' = [\{F'(t')\} \cup l']\!-\!\![\ \mathsf{Left}_{t_1}(t_1, x') \cup a'\ ]\!\!\rightarrow[r']$$

Moreover, there exists a substitution σ_i in normal form (the one used to instantiate ru') such that $m =_{AC} (t_1\sigma_i)\!\downarrow$ and $n =_{AC} x'\sigma_i\!\downarrow$. Since the trace is normalised, $m =_{AC} t_1\sigma_i$ and $n =_{AC} x'\sigma_i$. Let $u =_{AC} (t'\sigma_i)\!\downarrow$. Again, we have $u =_{AC} t'\sigma_i$. Since t_1 is a subterm of t' and t_1 is not headed by an AC symbol, we have that m is a subterm of u (modulo AC). Moreover $F'(u) \in S_{i-1}$ by definition of the application of a rule.

Let $j < i$ be the first occurence of j such that m (modulo AC) is a subterm of a fact in S_j and consider the j^{th} rule that has been applied.

- Either this rule is a rule ru'' in $\mathsf{Variant}(P)$ of the form

  $$ru'' = [l'']\!-\!\![\ a''\ ]\!\!\rightarrow[\{F''(w'')\} \cup r'']$$

  and there exists σ_j in normal form (the substitution used to instantiate ru'') such that m (modulo AC) is a subterm of $u' = (w''\sigma_j)\!\downarrow$. Since the trace is normalised, $(w''\sigma_j)\!\downarrow =_{AC} w''\sigma_j$. Let p' be the position at which m occurs in $w''\sigma_j$, i.e. such that $w''\sigma_j|_{p'} =_{AC} m$.

- $*$ Either p' is a path of w'' that does not end on a variable. Then $w''|_{p'} = w'$ with w' a protected subterm of w''.
  We have that $w'\sigma_j =_{AC} m =_{AC} t_1\sigma_i$ thus w' and t_1 are unifiable (modulo AC) thus we have annotated ru'', that is, $\mathsf{Right}_{t_1}(w') \in a''$, which concludes this case.
- $*$ Or p' is a path of w'' that ends on a variable or is not a path at all. Then there must exist a variable y in w'' such that m (modulo AC) is a subterm of $y\sigma_j$. Then y also appears in some premise fact $F'''(w''')$, thanks to the definition of a protocol rule and the fact that the variant rules are still protocol rules. Therefore m (modulo AC) is a subterm of a fact in S_{j-1} (since $(w'''\sigma_j)\downarrow =_{AC} w'''\sigma_j$), which contradicts the minimality of j.

• Or the rule is a MD rule. Since m is a protected term, the rule cannot be $[]{-[}\, \mathsf{K}^{\uparrow}(x) \,]{\to}[\mathsf{K}^{\uparrow}(x : pub)]$ nor $[\mathsf{Fr}(x : fr)]{-[}\, \mathsf{K}^{\uparrow}(x) \,]{\to}[\mathsf{K}^{\uparrow}(x : fr)]$ since these two rules only generate names. By minimality of j, it cannot be the rule $[\mathsf{Out}(x)]{-[]\to}[\mathsf{K}^{\downarrow}(x)]$, nor $[\mathsf{K}^{\uparrow}(x)]{-[}\, \mathsf{K}(x) \,]{\to}[\mathsf{In}(x)]$, nor the rule $[\mathsf{K}^{\downarrow}(x)]{-[}\, \mathsf{K}^{\uparrow}(x) \,]{\to}[\mathsf{K}^{\uparrow}(x)]$ either. So it must be a deduction rule, either in the $\mathsf{K}^{\uparrow}$ version or in the $\mathsf{K}^{\downarrow}$ version.

- $*$ Either it is the rule

$$[\mathsf{K}^{\uparrow}(x_1\theta), \ldots, \mathsf{K}^{\uparrow}(x_n\theta)]{-[}\, \mathsf{K}^{\uparrow}(\mathsf{f}(x_1, \ldots, x_n)\theta) \,]{\to}[\mathsf{K}^{\uparrow}(\mathsf{f}(x_1, \ldots, x_n)\theta)]$$

  with $\mathsf{f}(x_1, \ldots, x_n)\theta$ in normal form. We have $\mathsf{K}^{\uparrow}(x_1\theta), \ldots, \mathsf{K}^{\uparrow}(x_k\theta) \in S_{j-1}$. Then, by minimality of j, and since m is not headed with an AC symbol, we must have $m =_{AC} t_1\sigma_i =_{AC} \mathsf{f}(x_1\theta, \ldots, x_k\theta)$, otherwise we would have that m is subterm of some $x_i\theta$ hence subterm of S_{j-1} or m is a constant, which cannot be the case since m is a protected subterm. Remember that $x'\sigma_i$ is a subterm at position $p'_0 = i_0.p'$ (for some i_0) of

$$m =_{AC} t_1\sigma_i =_{AC} \mathsf{f}(x_1\theta, \ldots, x_k\theta)$$

  such that there are only pairs along p', that is, $x'\sigma_i \in St_{\mathsf{pair}}(x_{i_0}\theta)$. Since the trace is normalised (i.e. pairs are decomposed before being used), we get that $\mathsf{K}^{\uparrow}(x'\sigma_i) \in S_{j-1}$, that is $\mathsf{K}^{\uparrow}(n) \in S_{j-1}$. Now, by inspection of the rules, we notice that the only way to obtain $\mathsf{K}^{\uparrow}(t)$ in a state is through a rule annotated by $\mathsf{K}^{\uparrow}(t)$, hence we can conclude that $\mathsf{K}^{\uparrow}(n)$ appears in one of the actions of an earlier rule.
- $*$ Or the rule

$$[\mathsf{K}^{\alpha_1}(x_1\theta), \ldots, \mathsf{K}^{\alpha_n}(x_n\theta)]{-[}\, \mathsf{K}^{\downarrow}(\mathsf{f}(x_1, \ldots, x_n)\theta\downarrow) \,]{\to}[\mathsf{K}^{\downarrow}(\mathsf{f}(x_1, \ldots, x_n)\theta\downarrow)]$$

  has been applied, with $\mathsf{f}(x_1, \ldots, x_k)\theta$ that can be reduced at top level. Since the equational theory is a subterm theory, it must be the case that $\mathsf{f}(x_1, \ldots, x_k)\theta\downarrow$ is a subterm of one of the $x_i\theta$, hence a subterm of a fact of S_{j-1}, which contradicts the minimality of j.

This concludes the proof. □

Even if all the sources lemmas generated using our algorithm are correct, we will see in Section 7 that we make the choice to implement a slightly different version of our algorithm for practical reasons, in particular to avoid non-termination issues.

6. Improvement over the first algorithm

Unfortunately, the algorithm presented in Section 4 does not allow one to discard all partial deconstructions. In particular, it is useless to solve partial deconstructions that do not come from an encrypted term.

6.1. Motivation

We illustrate the weakness of this first algorithm through a simple example.

Example 9 (Private Channel)**.** *Consider the following* TAMARIN *rules:*

```
rule SendPwd: [ Fr(~pw) ] --[ Pwd(~pw) ]-> [ Ch(~pw) ]

rule Auth: [ Ch(x) ] --[ Forward(x) ]-> [ Ch(x) ]

rule Corrupt: [ Ch(x) ] --[ Corrupt(x) ]-> [ Out(x) ]
```

The first rule allows a user to send his password on a private channel (modelled using the fact `Ch`*), and the second one can be used to model the fact that when a first authority receives such a password, she may send it to another authority for performing additional checks. The third rule allows one to model corruption: at some point a password may be leaked and revealed to the attacker.*

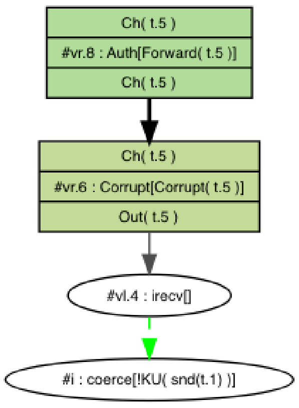
Ch(t.5)
#vr.8 : Auth[Forward(t.5)]
Ch(t.5)
Ch(t.5)
#vr.6 : Corrupt[Corrupt(t.5)]
Out(t.5)
#vl.4 : irecv[]
#i : coerce[!KU(snd(t.1))]

Fig. 2. Example of a partial deconstruction

On the above example, TAMARIN will generate 3 partial deconstructions, one of them is reproduced in Figure 2. TAMARIN does not know anything about the contents of the variable $t.5$, it is thus obliged to consider this case as a potential source for any value, and this will actually lead to non termination when trying to prove for instance a simple secrecy lemma as the one stated below:

```
lemma password_secrecy:
  " not(
     Ex #i x. Pwd(x)@i & (Ex #j. K(x) @ j) & not(Ex #r. Corrupt(x)@r)
       )"
```

It is easy to see that our algorithm SourceLemma1 will be of no help since there is no encrypted term involved in this protocol.

6.2. Algorithm

The sources lemma $\mathsf{SourceLemma2}(P, ru, x)$ associated to a partial deconstruction on a variable x and a rule ru for protocol P are defined by Algorithm 2. Intuitively, we now consider any unprotected occurrence of x in the premisses of ru that occurs in a fact $F(v_1, \dots, v_n)$ that is not an input, and we annotate the rule with $\mathsf{Left}_{F(v_1,\dots,v_n)}(v_1, \dots, v_n)$. Then, we look for all facts in the conclusions of a rule that may have produced $F(v_1, \dots, v_n)$, that is, facts $F(v'_1, \dots, v'_n)$ that can be unified with $F(v_1, \dots, v_n)$, and we annotate the rule with $\mathsf{Right}_{F(v_1,\dots,v_n)}(v'_1, \dots, v'_n)$. We then generate the lemma as expected, that says that if we have $\mathsf{Left}_{F(v_1,\dots,v_n)}(y_1, \dots, y_n)$ at some step i, then $\mathsf{Right}_{F(v_1,\dots,v_n)}(y_1, \dots, y_n)$ holds at an earlier step. The reason why we do not need to consider the case where x is already known by the attacker is that we reason here on protocol facts, which cannot be accessed or modified by the attacker.

Algorithm 2 $\mathsf{SourceLemma2}(P, ru, x)$

```
Input: $P$, $ru = [l]\!-\![\, a \,]\!\!\rightarrow\![r]$, $x$
  $\Psi := \top$
  for all $F(v_1, \dots, v_n) \in l \setminus \{In(\_)\}$ such that $x$ unprotected subterm of some $v_i$ do
    % we annotate $ru$ with the fact that $x$ may come from $v_i$
    $a := a \cup \{\mathsf{Left}_{F(v_1,\dots,v_n)}(v_1, \dots, v_n)\}$
    % then we identify from which facts $F(v_1, \dots, v_n)$ may provide.
    for all rule $ru' = [l']\!-\![\, a' \,]\!\!\rightarrow\![r'] \in P$ do
      if $F(v_1, \dots, v_n)$ unifiable with $F(v'_1, \dots, v'_n)$ modulo AC for some $F(v'_1, \dots, v'_n) \in r'\}$ then
        % we annotate $ru'$ with the fact that $F(v'_1, \dots, v'_n)$ may be used to produce $F(v_1, \dots, v_n)$
        $a' := a' \cup \{\mathsf{Right}_{F(v_1,\dots,v_n)}(v'_1, \dots, v'_n)\}$
      end if
    end for
    Let $\phi$ the formula defined as follows

      $\forall y_1, \dots, y_n, j\ \mathsf{Left}_{F(v_1,\dots,v_n)}(y_1, \dots, y_n)@j \implies (\exists k\{\mathsf{Right}_{F(v_1,\dots,v_n)}(y_1, \dots, y_n)@k \ \wedge\ k \lessdot j)$

    $\Psi := \Psi \wedge \phi$
  end for
  return $\Psi$
```

Example 10. *Going back to our running example developped in Section 6.1, we will apply our algorithm on rule* `Auth` *w.r.t. x. An annotation* $\mathsf{Left}_{Ch(x)}(x)$ *will be added on rule* `Auth`*, and the annotation* $\mathsf{Right}_{Ch(x)}$ *will be added twice: one on* `SendPwd` *with pw as a parameter, and the other one on* `Auth` *with x as a parameter. Indeed, these two rules are the only ones producing a fact of the form* $Ch(\cdot)$*.*

The resulting sources lemma is as follows (with $t_1 = Ch(x)$*):*

$$\forall y, i\ \mathsf{Left}_{t_1}(y)@i \implies (\exists k\ \mathsf{Right}_{t_1}(y)@k \ \wedge\ k \lessdot i)$$

This sources lemma is correct (as formally stated and proved in the theorem above). However, it does not allow TAMARIN *to generate the refined sources due to a loop on rule* `Auth`*. We will come back to this issue in Section 7.*

We can show that under the same assumptions than Theorem 1, the sources lemmas generated by Algorithm 2 always hold, which explains why TAMARIN is usually able to prove them. The proof follows the same lines as the one done for Theorem 1. Actually, its proof is easier since the fact $F(v_1, \dots, v_n)$ (where F is a fact symbol) cannot be produced by the attacker.

Theorem 2. *Given a set of (well-formed) protocol rules P, a rule ru* $\in \mathsf{Variant}(P)$*, a variable x occurring in ru, and* $\Psi = \bigwedge_{i=1}^{n} \phi_i$ *be the conjunction of formulas returned by* $\mathsf{SourceLemma2}(\mathsf{Variant}(P), ru, x)$*, then for any* $i \in \{1, \dots, n\}$*, we have that* ϕ_i *is satisfied by* $\mathsf{Variant}(P)$*, that is* $\mathsf{Variant}(P) \models_{\mathsf{norm}} \phi_i$*.*

Proof. Let P be a set of protocol rules, $ru \in \mathsf{Variant}(P)$ and a variable x occurring in ru, let $\Psi = \bigwedge_{i=1}^{n} \phi_i$ be the formula returned by $\mathsf{SourceLemma2}(\mathsf{Variant}(P), ru, x)$, and let $i_0 \in \{1, \dots, n\}$. The rule ru is of the form $[l]\!-\![\, a \,]\!\!\rightarrow\![r]$ and ϕ_{i_0} is of the form:

$$\forall y_1, \dots, y_n, i \; \mathsf{Left}_{F(v_1,\dots,v_n)}(y_1, \dots, y_n)@i \implies \big(\exists k \; \mathsf{Right}_{F(v_1,\dots,v_n)}(y_1, \dots, y_n)@k \;\wedge\; k \lessdot i\big)$$

for some $F(v_1, \dots, v_n)$ such that x is an unprotected subterm of some v_i.

Let tr be a normalised trace of $\mathsf{Variant}(P)$. Let us show that tr satisfies ϕ_{i_0}.

$$tr = \emptyset \rightsquigarrow^{A_1} S_1 \cdots \rightsquigarrow^{A_{n-1}} S_{n-1} \rightsquigarrow^{A_n} S_n$$

Let i be such that $\mathsf{Left}_{F(v_1,\dots,v_n)}(m_1, \dots, m_n) \in A_i$ for some terms $m_1, \dots, m_n$. Then the i^{th} applied rule must be an instance of a rule ru' in $\mathsf{Variant}(P)$ such that:

$$ru' = [\{F(v_1, \dots, v_n))\} \cup l']\!-\![\, \mathsf{Left}_{F(v_1,\dots,v_n)}(v_1, \dots, v_n) \cup a' \,]\!\!\rightarrow\![r']$$

Moreover, there exists a substitution σ_i in normal form (the one used to instantiate ru') such that $m_k =_{AC} (v_k\sigma_i)\!\downarrow$ for any $k \in \{1, \dots, n\}$. Since the trace is normalised, $m_k =_{AC} v_k\sigma_i$ for any $k \in \{1, \dots, n\}$. We have that $F(m_1, \dots, m_n) \in S_{i-1}$ by definition of the application of a rule.

Let $j < i$ be the first occurrence of j such that $F(m_1, \dots, m_n)$ is a fact in S_j and consider the j^{th} rule that has been applied. This rule is necessarily an instance of a rule $ru'' \in \mathsf{Variant}(P)$ of the form:

$$ru'' = [l'']\!-\![\, a'' \,]\!\!\rightarrow\![\{F(v'_1, \dots, v'_n)\} \cup r'']$$

and there exists σ_j in normal form (the substitution used to instantiate ru'') such that $m_k =_{AC} v'_k\sigma_j\!\downarrow$ for any $k \in \{1, \dots, n\}$. Since the trace is normalised, $(v'_k\sigma_j\!\downarrow \,=_{AC} v'_k\sigma_j$ for any $k \in \{1, \dots, n\}$. We have that $v_k\sigma_i =_{AC} m_k =_{AC} v'_k\sigma_j$ for any $k \in \{1, \dots, n\}$, thus $F(v_1, \dots, v_n)$ and $F(v'_1, \dots, v'_n)$ are unifiable (modulo AC) thus we have annotated ru'', that is $\mathsf{Right}_{F(v_1,\dots,v_n)}(v'_1, \dots, v'_n) \in a''$, which concludes this case. □

7. Implementation and experimental evaluation

We have implemented our approach in TAMARIN version 1.6.1 [26]. The automatic generation of sources lemmas is activated using the command line option `--auto-sources`. When TAMARIN is called, it will first load the theory and run the pre-computations normally (in particular it computes rule variants and sources). If TAMARIN is called using `--auto-sources`, and if the theory does not contain a sources lemma but has partial deconstructions, our new algorithm(s) are executed on the computed rule variants to generate a new sources lemma, which is then added to the theory, as well as the required rule annotations. In the interactive mode, the user can inspect the generated lemmas and annotations, and prove lemmas as usual. He can also download the modified theory if he wants to export the lemma, or modify it. In the automatic mode, TAMARIN directly tries to prove the generated sources lemma. When showing the results, TAMARIN displays the sources lemma among the other lemmas, and whether it managed to prove it.

7.1. Implementation

Dealing with composed rules. Actually, during the precomputations, TAMARIN might compute the composition of several rules. For example, when a rule ru_1 depends on a rule ru_2 in the sense that ru_1 can only be executed if ru_2 has been executed previously, TAMARIN will return the composition of both, not only ru_1. This yields bigger steps and it allows TAMARIN to prove lemmas more quickly.

Thus, the sources computed by TAMARIN are actually composed variants of initial protocol rules. Formally, given two rules $ru_1 = [l_1]\!-\![\, a_1 \,]\!\!\rightarrow\![r_1]$ and $ru_2 = [l_2]\!-\![\, a_2 \,]\!\!\rightarrow\![r_2]$, we define the *composition* of ru_1 and ru_2 w.r.t. θ, denoted $ru_1 \circ_\theta ru_2$ as the rule $[l]\!-\![\, a \,]\!\!\rightarrow\![r]$ defined as follows:

$$l = l_1\theta \cup^{\#} (l_2\theta \setminus^{\#} r_1\theta), \quad a = a_1\theta \cup a_2\theta, \text{ and } \quad r = (r_1\theta \setminus^{\#} l_2\theta) \cup^{\#} r_2\theta.$$

We denote $ru_1 \circ_\theta ru_2 \circ_\theta \cdots \circ_\theta ru_k$ the rule ru obtained by iterating $k-1$ compositions: $ru = ((ru_1 \circ_\theta ru_2) \circ_\theta \cdots) \circ_\theta ru_k$. Since the rules do not share any variable, θ is just the union of substitutions θ_i where the domain of θ_i is the set of variables of ru_i. It is easy to check that compositions of protocol rules yield protocol rules. Not all compositions are computed by TAMARIN, but we do not need to characterize which compositions are considered exactly. We simply show that any sources lemma generated from a composed rule is also sound.

Algorithm 3 describes how to generate a sources lemma from a composed rule. The idea is simply to identify, given a variable x, for which the partial deconstruction is incomplete, at which positions x appears in the some rule ru_i used for composition. We then generate the sources lemmas based on this rule. Note that Algorithm 3 is well defined only if $\mathsf{SourceLemma1}(P, ru_i, v_i|_p)$ (resp. $\mathsf{SourceLemma2}(P, ru_i, v_i|_p)$) is called only in case $v_i|_p$ is a variable. However, this follows from the fact that $v_i\theta|_p = x$ is a variable (with the notations of Algorithm 3).

Note that there are two subtleties here: First, in the first part of the algorithm concerning the protected subterms (SourceLemma1), we examine the premises of all rules inside the composed rule for occurrences of the variable x. For the second part, concerning the facts (SourceLemma2), we *only* examine the premises of the composed rule, meaning that if the variable occurs in the premise of an initial rule which was already solved during the composition, this occurrence is ignored. This turns out to be sufficient to resolve the partial deconstructions, and simplifies the generated lemmas. Second, we only use the second algorithm if the first one did not produce a lemma for a given variable x. Again, this is sufficient and generates simpler lemmas which are easier to prove for TAMARIN.

Theorem 3. *Given a set of well-formed protocol rules P, a composed rule $ru = ru_1 \circ_\theta ru_2 \circ_\theta \cdots \circ_\theta ru_k$ with $ru_i \in \mathsf{Variant}(P)$, a variable x occurring in ru, and $\Psi = \bigwedge_{i=1}^{n} \phi_i$ be the conjunction of formulas returned by $\mathsf{SourceLemmaComp}(\mathsf{Variant}(P), ru, x)$, then for any $i \in \{1, \ldots, n\}$, we have that ϕ_i is satisfied by $\mathsf{Variant}(P)$, that is $\mathsf{Variant}(P) \models_{\mathsf{norm}} \phi_i$.*

```
Algorithm 3 SourceLemmaComp($P, ru, x$)
Input: $P, ru = ru_1 \circ_\theta ru_2 \circ_\theta \cdots \circ_\theta ru_k,\ x$
  let $l, a, r$ such that $ru = [l]\!-\!\![\ a\ ]\!\!\rightarrow\![r]$, and $l_i, a_i, r_i$ such that $ru_i\theta = [l_i]\!-\!\![\ a_i\ ]\!\!\rightarrow\![r_i]$
  $\Psi = \top$
  for all position $p$ and $1 \leqslant j \leqslant k$ such that there exists $F(v) \in l_j$ such that $v|_p = x$ do
    for all $i$ such that $F(v) = F(v_i\theta)$ with $F(v_i)$ in the premisses of $ru_i$ do
      if $p$ is a position of $v_i$ then
        $\Psi_{i,p} = \mathsf{SourceLemma1}(P, ru_i, v_i|_p)$
        if $\Psi_{i,p} \neq \top$ then
          $\Psi = \Psi \wedge \Psi_{i,p}$
        end if
      end if
    end for
  end for
  if $\Psi \neq \top$ then
    return $\Psi$
  else
    for all position $p$ such that there exists $F(v) \in l$ such that $v|_p = x$ do
      for all $i$ such that $F(v) = F(v_i\theta)$ with $F(v_i)$ in the premisses of $ru_i$ do
        if $p$ is a position of $v_i$ then
          $\Psi_{i,p} = \mathsf{SourceLemma2}(P, ru_i, v_i|_p)$
          if $\Psi_{i,p} \neq \top$ then
            $\Psi = \Psi \wedge \Psi_{i,p}$
          end if
        end if
      end for
    end for
    return $\Psi$
  end if
```

Proof. The correctness of Algorithm 3 is a direct consequence of Theorems 1 and 2. Indeed, let $\Psi = \bigwedge_{i=1}^{n} \phi_i$ be the formula returned by $\mathsf{SourceLemmaComp}(\mathsf{Variant}(P), ru, x)$, and $i \in \{1, \dots, n\}$. Then ϕ_i is actually an element of the conjunction of the formula returned by $\mathsf{SourceLemma1}(\mathsf{Variant}(P), ru_i, v_i|_p)$ or $\mathsf{SourceLemma2}(\mathsf{Variant}(P), ru_i, v_i|_p)$ for some $ru_i \in \mathsf{Variant}(P)$ and some variable $v_i|_p$ of ru_i. Applying Theorem 1 (or Theorem 2, respectively), we have that $\mathsf{Variant}(P) \models_{\mathsf{norm}} \phi_i$, hence the conclusion. □

Heuristic and Optimizations. Our first experiments using Algorithm 3 showed that, for some examples, the generated lemmas, while true, caused TAMARIN to loop in the precomputations. This happened when the algorithm considered the case where a fact in the premises of a rule might have been produced by a fact in the conclusion of the same rule, as in Example 9. Hence, we have implemented an additional check that ignores this case, should it arise. For instance, in Example 9, the rule `Auth` is not annotated with $\mathsf{Right}_{Ch(x)}$, although the conclusion fact unifies with its premise fact. Note that in this example the generated lemma is still true, as the premise and conclusion are actually equal, and TAMARIN successfully proves the lemma.

In general, this additional condition means that the generated lemmas could potentially be false, however we did not observe this in practice. On the contrary, the examples that looped can now be proven correct, as described for Example 9. Also note that this does not contradict our theorems, as our lemmas are not minimal - we consider potentially too many cases, so removing some (unnecessary) ones can still result in a correct lemma.

Finally, we implemented a small optimization: in the case where our algorithm is unable to find any matching conclusions (i.e., no Right annotation is placed), we simplify the generated formula by replacing the Right by $\perp$. For example, instead of

$$\forall y, i \ \mathsf{Left}_{t_1}(y)@i \implies (\exists k \ \mathsf{Right}_{t_1}(y)@k \ \wedge \ k \lessdot i)$$

we obtain

$$\forall y, i \ \mathsf{Left}_{t_1}(y)@i \implies \perp.$$

This is obviously correct if there are no $\mathsf{Right}_{t_1}(y)$ annotations.

7.2. Evaluation

To evaluate the effectiveness of our approach, we selected several classical examples from the SPORE library of cryptographic protocols [27] and checked for standard properties such as secrecy of the exchanged key and mutual (injective and non-injective) authentication. Because of partial deconstructions, many of them were not entirely automatically verifiable in TAMARIN previously (except for extremely simple examples such as CCITT with only one message). The results are presented in Table 1, the TAMARIN models are available in the directory `examples/features/auto-sources/spore` of the TAMARIN repository [26]. Our approach succeeded in all cases.

To see whether our approach works on more complicated examples, we selected all files from the TAMARIN github repository [26] that contained lemmas annotated with `sources`, and that were not marked as "experimental", "work in progress", or "manual". It turned out that in some cases these examples did not actually contain any partial deconstructions, and that these "sources" lemmas were actually used to prove other protocol invariants. As our approach is only meant to handle partial deconstructions, we removed these examples from the set. Table 2 summarizes our results on the remaining examples, the files can be found in the directory `examples/features/auto-sources/tamarin-repo` of the TAMARIN repository [26].

It turns out that our algorithm still succeeds in generating successful sources lemmas in the majority of cases, in the sense that the sources lemma resolve all the partial deconstructions and can be proved correct by TAMARIN. Our examples include protocols with equivalence properties and SAPIC-generated[2] theories. However, as the examples are more complex, even with a correct sources lemma, TAMARIN does not always succeed in proving all other lemmas fully automatically.

Note also that although most examples are handled by Algorithm 1, Algorithm 2 applies for some more complex examples (OpenID Connect, Alethea, 5G AKA, PKCS11 AEAD/SIV). Again, the algorithm is successful in resolving the partial deconstructions in most cases, even though many of the files do not become fully automatic. However, for example in the case of Alethea, all following lemmas can be proven using the same heuristics as used in the initial files. Note also that two Alethea models initially

[2]SAPIC translates from applied pi models to TAMARIN theories.

Protocol Name	**Partial Dec.**	**Resolved**	**Automatic**	**Time**
Andrew Secure RPC	14	✓	✓	45.5s
Modified Andrew Secure RPC	21	✓	✓	135.3s
BAN Concrete Andrew Secure RPC	0	-	✓	10.3s
Lowe modified BAN Andrew Secure RPC	0	-	✓	31.8s
CCITT 1	0	-	✓	0.9s
CCITT 1c	0	-	✓	1.2s
CCITT 3	0	-	✓	181.4s
CCITT 3 BAN	0	-	✓	4.0s
Denning Sacco Secret Key	5	✓	✓	0.8s
Denning Sacco Secret Key - Lowe	6	✓	✓	2.8s
Needham Schroeder Secret Key	14	✓	✓	3.4s
Amended Needham Schroeder Secret Key	21	✓	✓	7.1s
Otway Rees	10	✓	✓	8.5s
SpliceAS	10	✓	✓	5.6s
SpliceAS 2	10	✓	✓	6.7s
SpliceAS 3	10	✓	✓	7.1s
Wide Mouthed Frog	5	✓	✓	0.6s
Wide Mouthed Frog Lowe	14	✓	✓	3.3s
WooLam Pi f	5	✓	✓	0.7s
Yahalom	15	✓	✓	3.4s
Yahalom - BAN	5	✓	✓	0.9s
Yahalom - Lowe	21	✓	✓	2.4s

Table 1

SPORE examples. "Partial Dec." indicates the number of partial deconstructions, "Resolved" indicates whether our auto-generated lemmas resolve them, and can be proven correct by TAMARIN. "Automatic" means that our auto-generated lemmas are then sufficient to directly prove or disprove the desired security properties.

did not contain a sources lemma although partial deconstructions are present, and by using our auto-generated lemmas we were able to improve the verification times slightly. In the first version of our work [12], our approach failed on these examples as we only used Algorithm 1.

We also encountered a few examples (most 5G AKA models and TPM Envelope) where the generated lemmas are sufficient to resolve the partial deconstructions, but TAMARIN does not succeed in (automatically) proving those lemmas correct. This is probably due to the complexity of the models – the initial files contained stronger invariants as sources lemmas and/or used special heuristics to prove them. In the case of PKCS11 AEAD/SIV, the generated lemma causes a loop in the precomputations similarly to the case of a rule where its conclusion matches its premise, but this time involving multiple intermediate rules. This causes our safeguard of avoiding unifications within the rule itself to fail in avoiding the problem. We tried to manually break the loop by removing the problematic annotation, but then the lemma becomes incorrect. Again, this model seems to require a stronger lemma.

We also analysed the examples where our algorithm failed to generate a correct sources lemma. The reasons turned out to be either a too complex equational theory (e.g., FOO and Okamoto, using blind signatures, or NSLPK3XOR and Chaum using XOR), or a particular modeling used in SAPIC that causes rules to have unbound variables (PKCS11-templates). Both cases are out of scope for our approach, as the files violate our initial assumptions on the equational theories or the well-formedness of the rules. It is thus expected that our generated lemmas do not solve the partial deconstructions, and even that they

Name	**Partial Dec.**	**Resolved**	**Automatic**	**Time (new)**	**Time (previous)**
Feldhofer (Equivalence)	5	✓	✓	3.8s	3.7s
NSLPK3	12	✓	✓	1.8s	1.8s
NSLPK3 untagged	12	✓	✗[1]	-	-
NSPK3	12	✓	✓	2.5s	2.3s
JCS12 Typing Example	7	✓	✗[2]	0.3s	0.2s
Minimal Typing Example	6	✓	✓	0.1s	0.1s
Simple RFID Protocol	24	✓	✗[2]	0.6s	0.5s
StatVerif Security Device	12	✓	✓	0.2s	0.3s
Envelope Protocol	9	✓	✗[2]	26.0s	26.8s
TPM Exclusive Secrets	9	✓	✗[2]	1.8s	1.9s
NSL untagged (SAPIC)	18	✓	✓	4.1s	21.2s
StatVerif Left-Right (SAPIC)	18	✓	✓	19.4s	14.2s
OpenID Connect	18	✓	✓	18.0	26.7
Alethea Voting Phase Privacy (Equiv.)	30	✓	✗[3]	1423.5	1454.6
Alethea Voting Ph. Receipt-Freeness (Equiv.)	30	✓	✗[3]	1727.0	1849.0
Alethea Voting Phase Malicious Server[4]	30	✓	✗[3]	1961.1	2036.2
Alethea Voting Phase Voter Abstention[4]	15	✓	✗[3]	92.5	113.4
5G AKA Privacy (Equivalence)	390	✓	✗[2,5]	-	-
5G AKA Binding Channel	225	○	-	-	-
5G AKA Binding Channel Fix	225	○	-	-	-
5G AKA Non Binding Channel	225	○	-	-	-
5G AKA Non Binding Channel Fix	225	○	-	-	-
TPM Envelope (Equivalence)	9	○	-	-	-
PKCS11 AEAD	144	✗[6]	-	-	-
PKCS11 SIV	162	✗[6]	-	-	-
PKCS11-templates (SAPIC)	68	✗	-	-	-
NSLPK3XOR	24	✗	-	-	-
Chaum Offline Anonymity	128	✗	-	-	-
FOO Eligibility	70	✗	-	-	-
Okamoto Eligibility	66	✗	-	-	-

Table 2

Examples from TAMARIN repository. "*New*" and "*previous*" verification times indicate the total verification time of all lemmas, either with the auto-generated lemma, or the manual sources lemma (if provided).

Resolved:
✓ The generated lemma removes all partial deconstructions and is automatically proven correct by TAMARIN.
○ The generated lemma removes all partial deconstructions, however TAMARIN does not terminate while trying to prove its correctness automatically.
✗ The generated lemma fails to remove all partial deconstructions.

Automatic:
✓ All other lemmas are automatically proven or disproven by TAMARIN, without further annotations or special proof heuristics.
✗ TAMARIN fails to prove (some of) the other lemmas without additional annotations or special proof heuristics.

[1] The sources lemma needs to be annotated with `reuse` for the following lemmas to be proven automatically.
[2] The file contains further intermediate lemmas annotated with reuse.
[3] The file requires the use of a special proof heuristic.
[4] The original file did not contain a sources lemma, although partial deconstructions are present.
[5] TAMARIN does not terminate while trying to prove the other lemmas using the generated lemma.
[6] TAMARIN does not terminate while trying to compute the refined sources using the generated lemma.

may be wrong. In the NSLPK3XOR example, the generated sources lemma helps a bit by discarding some partial deconstructions (the number of partial deconstructions went down from 24 to 12). However, in all other examples, this is not the case, and the number of partial deconstruction even increases. This can happen if the lemma causes the existing partial deconstructions to occur repeatedly instead of removing them.

When our approach succeeds, the verification times are close to timings measured using the manual sources lemmas. All timings have been measured on a standard laptop (Core i7, 16GB RAM, Ubuntu 20.04).

8. Conclusion

We have provided a technique that allows to automatically generate sources lemmas in TAMARIN, which otherwise had to be written by the user. In return, most simple protocols can now be analyzed automatically with TAMARIN.

As future work, it should be possible to further improve the level of automation. First, in several cases where our sources lemmas solve the partial deconstructions but are not yet sufficient to prove the security properties specified by the user, we are actually close to full automation. What is missing is to indicate to TAMARIN that it should reuse one of the properties (e.g. secrecy of some long-term key) to prove another property (e.g. authentication). One interesting direction is to investigate how to automate these "re-use" annotations, without increasing the complexity of the tool.

Our result holds for subterm convergent theories (modulo AC) that have the finite variant property. However, our algorithm does not generate lemmas for terms headed with an AC symbol (for example exclusive or) as the resulting lemmas would be false in most cases. Hence, manual sources lemmas are still necessary. A future research direction is to explore how to extend our result to tackle this case, which may require to write more complex sources lemmas, e.g. to account for all possible decompositions induced by the exclusive or operator.

Thanks to our sources lemma, the automation of TAMARIN has improved, in particular on simple protocols. It would be interesting to compare extensively the tools ProVerif and TAMARIN, in order to identify on which cases they are both automatic, and on which kind of protocols, one of the two tools is more likely to conclude automatically. This should also provide directions to improve the automation of both tools.

Acknowledgments

This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR and grant agreement No 645865-SPOOC), as well as from the French National Research Agency (ANR) under the projects TECAP and SEVERITAS.

References

[1] N. Durgin, P. Lincoln, J. Mitchell and A. Scedrov, Undecidability of bounded security protocols, in: *Workshop on Formal Methods and Security Protocols*, Trento, Italia, 1999.

[2] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò and L. Vigneron, The AVISPA Tool for the automated validation of internet security protocols and applications, in: *17th International Conference on Computer Aided Verification, CAV'2005*, K. Etessami and S. Rajamani, eds, Lecture Notes in Computer Science, Vol. 3576, Springer, Edinburgh, Scotland, 2005, pp. 281–285.

[3] V. Cheval, S. Kremer and I. Rakotonirina, DEEPSEC: Deciding Equivalence Properties in Security Protocols - Theory and Practice, in: *Proceedings of the 39th IEEE Symposium on Security and Privacy (S&P'18)*, IEEE Computer Society Press, 2018, pp. 525–542.

[4] B. Blanchet, An Efficient Cryptographic Protocol Verifier Based on Prolog Rules, in: *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, IEEE Computer Society, Cape Breton, Nova Scotia, Canada, 2001, pp. 82–96.

[5] K. Bhargavan, B. Blanchet and N. Kobeissi, Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate, in: *IEEE Symposium on Security and Privacy (S&P'17)*, San Jose, CA, 2017, pp. 483–503.

[6] B. Blanchet, Symbolic and Computational Mechanized Verification of the ARINC823 Avionic Protocols, in: *30th IEEE Computer Security Foundations Symposium (CSF'17)*, Santa Barbara, CA, USA, 2017, pp. 68–82.

[7] V. Cortier, D. Galindo and M. Turuani, A formal analysis of the Neuchâtel e-voting protocol, in: *3rd IEEE European Symposium on Security and Privacy (EuroSP'18)*, London, UK, 2018, pp. 430–442.

[8] S. Meier, B. Schmidt, C. Cremers and D. Basin, The TAMARIN Prover for the Symbolic Analysis of Security Protocols, in: *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, LNCS, Vol. 8044, Springer, 2013, pp. 696–701.

[9] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse and V. Stettler, A Formal Analysis of 5G Authentication, in: *25th ACM Conference on Computer and Communications Security (CCS'18)*, 2018.

[10] B. Schmidt, R. Sasse, C. Cremers and D. Basin, Automated Verification of Group Key Agreement Protocols, in: *IEEE Symposium on Security and Privacy (S&P'14)*, 2014.

[11] G. Girol, L. Hirschi, R. Sasse, D. Jackson, C. Cremers and D. Basin, A Spectral Analysis of Noise: A Comprehensive, Automated, Formal Analysis of Diffie-Hellman Protocols, in: *Usenix Security*, 2020.

[12] V. Cortier, S. Delaune and J. Dreier, Automatic generation of sources lemmas in Tamarin: towards automatic proofs of security protocols, in: *ESORICS 2020 - 25th European Symposium on Research in Computer Security*, 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part II, Vol. 12309, Guilford, United Kingdom, 2020, pp. 3–22.

[13] B. Schmidt, S. Meier, C.J.F. Cremers and D.A. Basin, Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties, in: *CSF 2012*, 2012, pp. 78–94.

[14] B. Schmidt, R. Sasse, C. Cremers and D. Basin, Automated Verification of Group Key Agreement Protocols, in: *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 179–194.

[15] J. Dreier, C. Duménil, S. Kremer and R. Sasse, Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols, in: *POST 2017 - 6th International Conference on Principles of Security and Trust*, Proceedings of the 6th International Conference on Principles of Security and Trust, Vol. 10204, Springer, Uppsala, Sweden, 2017, pp. 117–140.

[16] J. Dreier, L. Hirschi, S. Radomirovic and R. Sasse, Automated Unbounded Verification of Stateful Cryptographic Protocols with Exclusive OR, in: *CSF 2018*, 2018, pp. 359–373.

[17] J. Dreier, L. Hirschi, S. Radomirović and R. Sasse, Verification of Stateful Cryptographic Protocols with Exclusive OR, *Journal of Computer Security* **28**(1) (2020), 1–34.

[18] D. Basin, J. Dreier and R. Sasse, Automated Symbolic Proofs of Observational Equivalence, in: *22nd ACM SIGSAC Conference on Computer and Communications Security (ACM CCS 2015)*, ACM, Denver, United States, 2015, pp. 1144–1155.

[19] S. Kremer and R. Künnemann, Automated analysis of security protocols with global state, *J. Comput. Secur.* **24**(5) (2016), 583–616. doi:10.3233/JCS-160556.

[20] D.A. Basin, M. Keller, S. Radomirovic and R. Sasse, Alice and Bob Meet Equational Theories, in: *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, N. Martí-Oliet, P.C. Ölveczky and C.L. Talcott, eds, Lecture Notes in Computer Science, Vol. 9200, Springer, 2015, pp. 160–180. doi:10.1007/978-3-319-23165-5_7.

[21] Y. Xiong, C. Su, W. Huang, F. Miao, W. Wang and H. Ouyang, SmartVerif: Push the Limit of Automation Capability of Verifying Security Protocols by Dynamic Strategies, in: *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, 2020, pp. 253–270. ISBN 978-1-939133-17-5. https://www.usenix.org/conference/usenixsecurity20/presentation/xiong.

[22] S. Escobar, C. Meadows and J. Meseguer, A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties, *Theoretical Computer Science* **367**(1–2) (2006), 162–202.

[23] S. Escobar, C. Meadows and J. Meseguer, A rewriting-based inference system for the NRL Protocol Analyzer: grammar generation, in: *ACM workshop on Formal methods in security engineering (FMSE'05)*, 2005, pp. 1–12.

[24] A. Tiu and J.E. Dawson, Automating Open Bisimulation Checking for the Spi Calculus, in: *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, 2010, pp. 307–321.
[25] H. Comon-Lundh and S. Delaune, The finite variant property: How to get rid of some algebraic properties, in: *Proc. 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, LNCS, Vol. 3467, Springer, Nara, Japan, 2005, pp. 294–307.
[26] Main source code repository of the Tamarin prover for security protocol verification, Accessed on 12/03/2021.
[27] Security Protocols Open Repository, Accessed on 04/24/2020.

Appendix A. TAMARIN source of our running examples

These files are also available on our fork of the TAMARIN github repository [26].

```
theory runningAlgo1
begin

/* We formalize the following challenge-response protocol
    1. I -> R: {'req',I, n}pk(R)
    2. I <- R: {'rep',n}pk(I)                           */

builtins: asymmetric-encryption

// Public key infrastructure
rule Register_pk:
  [ Fr(~ltkA) ]
  -->
  [ !Ltk($A, ~ltkA), !Pk($A, pk(~ltkA)), Out(pk(~ltkA)) ]

rule Reveal_ltk:
  [ !Ltk(A, ltkA) ] --[ RevLtk(A) ]-> [ Out(ltkA) ]

rule Rule_I:
  let m1 = aenc{'req', $I, ~n}pkR in
    [ Fr(~n), !Pk(R, pkR),!Ltk($I, ltkI)]
  --[SecretI($I,R,~n)]->
    [ Out(m1), State_I($I, R, ~n)]

rule Rule_R:
  let m1 = aenc{'req', I, x}pk(ltkR)
      m2 = aenc{'rep', x}pkI in
    [ !Ltk(R, ltkR), In(m1), !Pk(I, pkI)]
  -->
    [ Out(m2), State_R(R, I, x)]

lemma nonce_secrecy:
  "not(Ex A B s #i. SecretI(A, B, s) @ i  & (Ex #j. K(s) @ j)
       & not (Ex #r. RevLtk(A) @ r)
       & not (Ex #r. RevLtk(B) @ r)
       )"



theory runningAlgo2
begin

/*
We formalize the following password-based protocol
    1. U -> A1: pw
    2. A1 -> A2: pw
The two first exchanges are done on a private channel.
This corresponds to the fact the password is sent on a
private channel to an authority A1, and then forwarded
to another authority A2.
We also add a rule to model corruption.
*/

rule SendPwd: [ Fr(~pw) ] --[ Pwd(~pw) ]-> [ Ch(~pw) ]

rule Auth: [ Ch(x) ] --[ Forward(x) ]-> [ Ch(x) ]

rule Corrupt: [ Ch(x) ] --[ Corrupt(x) ]-> [ Out(x) ]

lemma password_secrecy:
  "not(
   Ex #i x. Pwd(x)@i & (Ex #j. K(x) @ j) & not(Ex #r. Corrupt(x)@r)
   )"

end
```