Automated analysis of equivalence properties for security protocols using else branches

Ivan Gazeau, Steve Kremer

Automated analysis of equivalence properties for security protocols using else branches *

Ivan Gazeau and Steve Kremer

LORIA, INRIA Nancy - Grand-Est

Abstract. In this paper we present an extension of the AKISS protocol verification tool which allows to verify equivalence properties for protocols with else branches, i.e., disequality tests. While many protocols are represented as linear sequences or inputs, outputs and equality tests, the reality is often more complex. When verifying equivalence properties one needs to model precisely the error messages sent out when equality tests fail. While ignoring these branches may often be safe when studying trace properties this is not the case for equivalence properties, as for instance witnessed by an attack on the European electronic passport. One appealing feature of our approach is that our extension re-uses the saturation procedure which is at the heart of the verification procedure of AKISS as a black box, without need to modify it. As a result we obtain the first tool that is able verify equivalence properties for protocols that may use xor and else branches. We demonstrate the tool's effectiveness on several case studies, including the AKA protocol deployed in mobile telephony.

1 Introduction

Security protocols are communication protocols that rely on cryptographic primitives, e.g. encryption, or digital signatures to ensure security properties, e.g., confidentiality or authentication. Well-known examples of security protocols include TLS [23], Kerberos [28] and IKE [27]. These protocols are extremely difficult to design as they must ensure the expected security property, even if the network is under control of an attacker: each message sent on the network can be intercepted by the attacker, each received message potentially originates from the attacker, and the attacker may manipulate all received data by applying functions on it. Moreover, as several sessions of the protocol may be executed concurrently, one must consider all possible interleavings, and the attacker may even participate in some of these sessions as a legitimate participant. As a result, a security proof by hand is extremely tricky as it would require to explore all of the possible cases.

A successful approach to discover weaknesses in such protocols, or show their absence is to use dedicated formal verification tools. A variety of tools for analysing protocols exist: ProVerif [14], Scyther [21], Maude-NPA [24], Tamarin [31], AVISPA [7], ...

* The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreements No 645865-SPOOC), as well as from the French National Research Agency (ANR) under the project Sequoia.

These tools were generally initially developed for verifying trace properties of rather simple protocols. In the last years there has been a large body of works for extending these tools to handle more general properties and more complex protocols.

Most tools were designed for analysing trace properties: a protocol cannot reach a bad state, e.g., a state where the attacker knows a secret value. Many important security properties are however stated in terms of *indistinguishability*: can an attacker distinguish two protocols? For instance *real-or-random* secrecy states the indistinguishability of two protocols, one outputting at the end of the run the "real" secret, used within the protocol, while the other protocol outputs a freshly generated random secret. Similarly, *unlinkability* can be modeled by the adversary's inability to distinguish two sessions run by the same party form two sessions run by two different parties. More generally, strong flavours of secrecy [12], anonymity and unlinkability [5], as well as vote privacy [22], are expressed as indistinguishability. This notion is naturally modelled in formal models through behavioural equivalences in cryptographic process calculi, such as the spi [3] and applied pi calculus [1]. During the last years, several specific tools for checking such equivalences have been developed [32,17,15], or existing tools have been extended to handle these properties [13,10,30].

Similarly, many tools were designed to verify protocols that have a simple, linear execution flow: many protocol specification languages allow several roles in parallel, each consisting of a sequence of input, or output actions with the possibility to check equality between parts of messages. More complex protocols do however require branching, and allow to react differently according to whether an equality test holds or not (rather than just halting if a test fails). As demonstrated by an attack on the European electronic passport [20], taking into account the exact error message in case a test fails may be crucial: the fact that in some versions of the passport output different error messages allowed an attacker to trace a given passport. In this paper we will extend the AKISS tool with the ability to verify protocols that have else branches.

An overview of the AKISS *tool.* The AKISS tool [15] is a verification tool for checking indistinguishability properties. More precisely, it verifies trace equivalence in a replication free (i.e., considering a bounded number of sessions) and positive (no else branches) fragment of the applied pi calculus. The tool allows a wide range of cryptographic primitives that are specified by the means of a user defined equational theory. The tool is correct for any equational theory that can be oriented into a convergent rewrite system which has the finite variant property and was shown to guarantee termination on any subterm convergent equational theory. This class of theories include classical cryptographic primitives such as encryption, signatures and hashes, but also non-interactive zero knowledge proofs. Moreover, even though termination is not guaranteed protocols relying on blind signatures or trapdoor commitments have been successfully analysed. In addition, a recent extension of AKISS provides support for protocols that use the exclusive or (xor) operation [8].

In a nutshell, AKISS proceeds as follows. Protocols are translated into first-order Horn clauses. Next, the set of Horn clauses is saturated using a dedicated Horn clause resolution procedure. This saturated set of clauses provides a finite representation of all reachable states of the protocols, of the intruder knowledge and equality tests that hold on the protocol outputs. These equality tests are used by the adversary to distin-

guish protocols, i.e., its aim is to find a test which holds on one protocol, but not the other. Next, AKISS uses this saturated set of Horn clauses to decide trace equivalence when the processes specifying the protocol are determinate (the precise definition of determinacy will be given in section 2). On general processes, AKISS may over- and under-approximate trace equivalence, as discussed in [15].

Our contributions. Our main contribution is to extend the AKISS tool to allow more complex protocols which allow non trivial else branches. An interesting point of our approach is that we do not need to modify the saturation procedure of AKISS: we only need to saturate positive processes (in which disequality tests are ignored). The algorithm is based on the following simple observation: whenever a trace is not executable because of the failure of a disequality test $t_1 \neq t_2$, the saturation of the process in which $t_1 \neq t_2$ is replaced by $t_1 = t_2$ computes all the traces that fail to execute on the original process (due to this particular disequality test). This test can then be confronted to the other process that we expect to be trace equivalent.

From a theoretical point, given that the saturation of AKISS was shown to terminate on any subterm convergent rewrite system our algorithm provides, *en passant*, a new decidability result for the class of subterm convergent equational theories for protocols with else branches, generalising the results of [11,19] that do not allow else branches and the result of [16], which only applies to a particular equational theory. Moreover, the result is modular, in the sense that if we generalize the saturation procedure to other equational theories support for else branches comes for "for free". From a more practical point, we have implemented our new procedure in the AKISS tool and demonstrate its effectiveness on several case studies. Hence, we provide the first tool that is able to handle protocols that require both xor and else braches: in addition to previously analysed protocols, such as the private authentication protocol and the BAC protocol implemented in the European passport, this allows us to analyse protocols using xor, such as the AKA protocol [4] used in 3G and 4G mobile telephony, as well as xor-based RFID protocols with key update (which requires an else branch for the modelling). A previous analysis of the AKA protocol with ProVerif replaced the use of xor with encryption [6]. Replacing xor by encryption may however miss attacks as it was shown by Ryan and Schneider in [29].

Related work. We consider two kinds of tools: those restricted to a bounded number of sessions (as in our work) and those that allow for an unbounded number of sessions. The first kind of tools includes the SPEC and APTE tools. SPEC [32] allows to verify a symbolic bisimulation: it only supports a fixed equational theory (encryption, signature, hash and mac) and has no support for else branches. The APTE tool also supports a fixed equational theory (similar to SPEC), but allows else branches. Both tools are not restricted to determinate processes. Tools that allow protocol verification for an unbounded number of sessions include ProVerif, Maude NPA and Tamarin. Given that the underlying problem is undecidable when the number of sessions is not bounded, termination is not guaranteed. Each of these tools allows for else branches user-defined equational theories, but ProVerif and Tamarin do not include support for xor. While Maude NPA does support xor in principle, termination fails even on simple examples [30]. We may also note that the support for else branches in Maude NPA is very recent [34].

Finally, each of these three tools checks a more fine-grained relation that trace or observational equivalence, called *diff-equivalence*: this equivalence requires both processes to follow the same execution flow and is too fine-grained for some applications.

Full proofs, omitted because of lack of space, are available in [25].

2 A formal model for security protocols

In this section we introduce our formal language for modelling security protocols. Messages are modelled as *terms* and equipped with an *equational theory*, that models the algebraic properties of cryptographic primitives. The protocols themselves will be modelled in a process calculus similar to the applied pi calculus [1]: protocol participants are modelled as processes and their interaction through message passing.

2.1 Term algebra

Terms are built over the following atomic messages : the set of *names* $\mathcal{N}$, that is partitioned into *private names* $\mathcal{N}_{\mathsf{prv}}$ and *public names* $\mathcal{N}_{\mathsf{pub}}$; the set of *message variables* $\mathcal{X}$, denoted x, y, ...; the set of parameters $\mathcal{W} = \{\mathsf{w}_1, \mathsf{w}_2, \ldots\}$. Private names are used to model fresh, secret values, such as nonces or cryptographic keys. Public names represent publically known values such as identifier and are available to the attacker. Parameters allow the adversary to refer to messages that were previously output.

We consider a *signature* Σ, i.e., a finite set of function symbols together with their arity. Function symbols of arity 0 are called *constants*. Given a signature Σ and a set of atoms $\mathcal{A}$ we denote by $\mathcal{T}(\Sigma, \mathcal{A})$ the set of *terms*, defined as the smallest set that contains $\mathcal{A}$ and is closed under application of function symbols. We denote by $vars(t)$ the set of *variables* occurring in a term t. A *substitution* is a function from variables to terms, lifted to terms homomorphically. The application of a substitution σ to a term u is written $u\sigma$, and we denote $dom(\sigma)$ its *domain*, i.e. $dom(\sigma) = \{x \mid \sigma(x) \neq x\}$. We denote the identity substitution whose domain is the empty set by $\emptyset$.

We equip the signature Σ with an *equational theory* E: an equational theory is defined by a set of equations $M = N$ with $M, N \in \mathcal{T}(\Sigma, \mathcal{X})$. The equational theory E induces an equivalence relation $=_E$ on terms: $=_E$ is the smallest equivalence on terms, which contains all equations $M = N$ in E, is closed under application of function symbols and substitutions of variables by terms.

Example 1. As an example we model the exclusive-or operator. Let $\Sigma_{\mathsf{xor}} = \{\oplus, \mathsf{0}\}$, and the equational theory $\mathsf{E}_{\mathsf{xor}}$ defined by the following equations:

$$x \oplus x = \mathsf{0} \qquad x \oplus (y \oplus z) = (x \oplus y) \oplus z \qquad x \oplus \mathsf{0} = x \qquad x \oplus y = y \oplus x$$

Additional primitives, e.g. pairs, symmetric and asymmetric encryptions, signatures, hashes, *etc*, can be modelled by extending the signature and equational theory.

Example 2. Let $\Sigma^{+}_{\mathsf{xor}} = \Sigma_{\mathsf{xor}} \uplus \{\langle\cdot,\cdot\rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{h}\}$, and $\mathsf{E}^{+}_{\mathsf{xor}}$ be defined by extending $\mathsf{E}_{\mathsf{xor}}$ with the following equations : $\mathsf{proj}_1(\langle x, y\rangle) = x$, and $\mathsf{proj}_2(\langle x, y\rangle) = y$.

The symbol $\langle \cdot, \cdot \rangle$ models pairs and proj_1 and proj_2 projections of the first and second element. The unary symbol h models a cryptographic hash function. Let $AUTN = \langle SQN_N \oplus AK, MAC \rangle$, then we have $\mathsf{proj}_1(AUTN) \oplus AK =_E SQN_N$.

As we build on the AKISS tool [15,8] we suppose in the following that the signature and equational theory are an extension the theory of exclusive or, i.e., Σ is such that $\Sigma_{\mathsf{xor}} \subseteq \Sigma$ and $\mathsf{E} = \mathsf{E}_{\mathsf{xor}} \cup \{M = N \mid M, N \in \mathcal{T}(\Sigma \setminus \Sigma_{\mathsf{xor}}, \mathcal{X})\}$. and that E can be oriented into a convergent rewrite system which has the finite variant property. This allows to model a wide range of cryptographic primitives, including symmetric and asymmetric encryption, digital signatures, hash functions and also zero knowledge proofs or blind signatures. We refer the reader to [15] for the precise technical definitions, which are not crucial for this paper.

2.2 Process calculus

Syntax Let $\mathcal{C}h$ be a set of public channel names. A *protocol* is a set of processes and a *process* is generated by the following grammar:

$$\begin{array}{rrr} P, P', P_1, P_2 ::= & \mathbf{0} & \text{null process} \\ & \mathbf{in}(c, x).P & \text{input} \\ & \mathbf{out}(c, t).P & \text{output} \\ & [s = t].P & \text{test}^{=} \\ & [s \neq t].P & \text{test}^{\neq} \end{array}$$

where $x \in \mathcal{X}, s, t \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$, and $c \in \mathcal{C}h$.

A receive action $\mathbf{in}(c, x)$ acts as a binding construct for the variable x and *free* and *bound variables* of processes are defined as usual. We also assume that each variable is bound at most once. A process is *ground* if it does not contain any free variables. For sake of conciseness, we sometimes omit the null process at the end of a process.

Following [15], we only consider a minimalistic core calculus. Given that we only consider a bounded number of sessions (i.e., a process calculus without replication) and that we aim at verifying trace equivalence, parallel composition, denoted $P \parallel Q$ can be added as syntactic sugar to denote the set of all interleavings at a cost of an exponential blow-up (see [15]). Similarly, we can encode conditionals: a process

$$\mathbf{if}\ t_1 = t_2\ \mathbf{then}\ P\ \mathbf{else}\ Q$$

can be encoded by the set $\{[t_1 = t_2].P, [t_1 \neq t_2].Q\}$. As usual we omit **else** Q when $Q = 0$ and sometimes write **if** $t_1 \neq t_2$ **then** P **else** Q for **if** $t_1 = t_2$ **then** Q **else** P. This will ease the specification of protocols and improve readability. A protocol typically consists of the set of all possible interleavings.

Example 3. As an example consider a simplified version of the AKA protocol, which is vulnerable to replay attacks, depicted in Figure 1. The network (NS) and mobile station (MS) share a secret key k_{IMSI}. NS generates a nonce r which it sends to MS together with $\mathrm{f}(k_{IMSI}, r)$ where f models a message authentication code (MAC). MS verifies

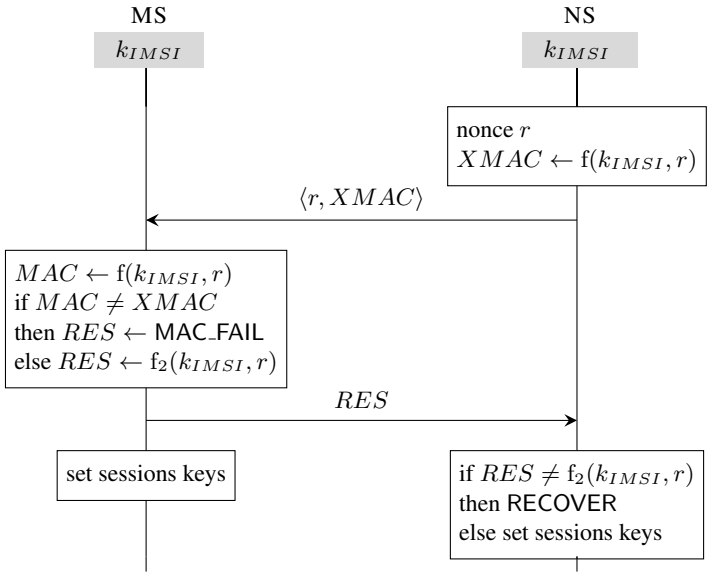

Fig. 1: A simplified version of the AKA protocol

the MAC: if successful it sends another MAC based on function f_2 and generates sessions keys from r and k_{IMSI}; otherwise, it sends an error message. NS checks whether the received message is the expected MAC or the error message. In case the MAC is received it generates the sessions keys; otherwise it starts a recovery protocol.

Using the additional operators introduced above, we can model the protocol as $MS \parallel NS$ where

$$NS \hat{=} \textbf{out}(c, \langle r, \mathrm{f}(k_{IMSI}, r)\rangle).\textbf{in}(c, x).\textbf{if } x \neq \mathrm{f}_2(k_{IMSI}, r) \textbf{ then out}(c, \mathsf{RECOVER})$$
$$MS \hat{=} \textbf{in}(c, y).\ \textbf{if } y \neq \mathrm{f}(k_{IMSI}, r) \textbf{ then out}(c, \mathsf{MAC\_FAIL}) \textbf{ else out}(c, \mathrm{f}_2(k_{IMSI}, r))$$

If we skip the actions $[x \neq \mathrm{f}_2(k_{IMSI}, r)].\textbf{out}(c, \mathsf{RECOVER})$, the protocol corresponds to a set of 12 processes which include, for instance, the 4 following processes.

$$\textbf{out}(c, \langle r, \mathrm{f}(k_{IMSI}, r)\rangle).\textbf{in}(c, y).[y \neq \mathrm{f}(k_{IMSI}, r)].\textbf{out}(c, \mathsf{MAC\_FAIL}).\textbf{in}(c, x)$$
$$\textbf{out}(c, \langle r, \mathrm{f}(k_{IMSI}, r)\rangle).\textbf{in}(c, y).[y = \mathrm{f}(k_{IMSI}, r)].\textbf{out}(c, \mathrm{f}_2(k_{IMSI}, r)).\textbf{in}(c, x)$$
$$\textbf{out}(c, \langle r, \mathrm{f}(k_{IMSI}, r)\rangle).\textbf{in}(c, y).\textbf{in}(c, x).[y \neq \mathrm{f}(k_{IMSI}, r)].\textbf{out}(c, \mathsf{MAC\_FAIL})$$
$$\textbf{out}(c, \langle r, \mathrm{f}(k_{IMSI}, r)\rangle).\textbf{in}(c, y).\textbf{in}(c, x).[y = \mathrm{f}(k_{IMSI}, r)].\textbf{out}(c, \mathrm{f}_2(k_{IMSI}, r))$$

Note that since a test is an invisible action, there is no need to consider traces where a test does not strictly precede its following action. The correctness of this optimization is proven in [15].

Semantics In order to define the operational semantics of our process calculus we define the notion of *message deduction*. Intuitively, message deduction models which

new messages an intruder can construct from previously learnt messages. The messages output during a protocol execution are presented by a *frame*:

$$\varphi = \{\mathsf{w}_1 \mapsto t_1, \ldots, \mathsf{w}_\ell \mapsto t_\ell\}$$

A frame is a substitution $dom(\varphi) = \{\mathsf{w}_1, \ldots, \mathsf{w}_\ell\}$. An intruder may refer to the ith term through the parameter w_i.

Definition 1. *Let φ be a frame, $t \in \mathcal{T}(\Sigma, \mathcal{N})$ and $R \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup dom(\varphi))$. We say that t is* deducible from φ using R*, written $\varphi \vdash_R t$, when $R\varphi =_E t$.*

Intuitively, an attacker can deduce new messages by applying function symbols in Σ to public names (in $\mathcal{N}_{\mathsf{pub}}$) and terms he already knows (those in φ). The term R is called a *recipe*.

A *configuration* is a pair (P, φ) where P is a ground process, and φ is a frame. The operational semantics is defined as a labelled transition relation on configurations $\xrightarrow{\ell}$ where ℓ is either an input, an output, or an unobservable action **test** defined as follows:

$$\textsc{Recv}\quad (\mathbf{in}(c,x).P, \varphi) \xrightarrow{\mathbf{in}(c,R)} (P\{x \mapsto t\}, \varphi) \quad \text{if } \varphi \vdash_R t$$

$$\textsc{Send}\quad (\mathbf{out}(c,t).P, \varphi) \xrightarrow{\mathbf{out}(c)} (P, \varphi \cup \{\mathsf{w}_{|\varphi|+1} \mapsto t\})$$

$$\textsc{Test}^{=}\quad ([s = t].P, \varphi) \xrightarrow{\mathbf{test}} (P, \varphi) \qquad \text{if } s =_E t$$

$$\textsc{Test}^{\neq}\quad ([s \neq t].P, \varphi) \xrightarrow{\mathbf{test}} (P, \varphi) \qquad \text{if } s \neq_E t$$

Intuitively, the labels have the following meaning:

- $\mathbf{in}(c, R)$ represents the input of a message sent by the attacker on channel c and the message is deduced using recipe R;
- $\mathbf{out}(c)$ represents the output of a message on channel c (adding the message to the frame);
- **test** represents the evaluation of a conditional (in the equational theory).

When $\ell \neq$ **test** we define $\stackrel{\ell}{\Rightarrow}$ to be $\xrightarrow{\mathbf{test}}{}^{*} \xrightarrow{\ell} \xrightarrow{\mathbf{test}}{}^{*}$ and we lift $\xrightarrow{\ell}$ and $\stackrel{\ell}{\Rightarrow}$ to sequences of actions. Given a protocol $\mathcal{P}$, we write $(\mathcal{P}, \varphi) \xrightarrow{\ell_1, \ldots, \ell_n} (P', \varphi')$ if there exists $P \in \mathcal{P}$ such that $(P, \varphi) \xrightarrow{\ell_1, \ldots, \ell_n} (P', \varphi')$, and similarly for $\stackrel{\ell}{\Rightarrow}$.

2.3 Trace equivalence

The fact that an attacker cannot distinguish two protocols will be modelled through *trace equivalence*. We first define the notion of a test which an attacker may apply on a frame to try to distinguish two processes.

Definition 2. *Let φ be a frame and R_1, R_2 be two terms in $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup dom(\varphi))$. The* test $R_1 \stackrel{?}{=} R_2$ holds *on frame φ, written $(R_1 = R_2)\varphi$, if $R_1\varphi =_E R_2\varphi$.*

Trace equivalence of processes P and Q states that any test that holds on process P (after some execution) also holds on process Q after the same execution.

Definition 3 ([15]). *A protocol $\mathcal{P}$ is trace included in a protocol $\mathcal{Q}$, denoted $\mathcal{P} \sqsubseteq \mathcal{Q}$, if whenever $(\mathcal{P}, \emptyset) \xRightarrow{\ell_1,\ldots,\ell_n} (P, \varphi)$ and $(R_1 = R_2)\varphi$, then there exists a configuration (Q', φ') such that $(\mathcal{Q}, \emptyset) \xRightarrow{\ell_1,\ldots,\ell_n} (Q', \varphi')$ and $(R_1 = R_2)\varphi'$.*

We say that $\mathcal{P}$ and $\mathcal{Q}$ are equivalent*, written $\mathcal{P} \approx \mathcal{Q}$, if $\mathcal{P} \sqsubseteq \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq \mathcal{P}$.*

This notion of equivalence does not coincide with the usual notion of trace equivalence as defined e.g. in [18]. It is actually coarser and is therefore sound for finding attacks. However, it has been shown that the classical and above defined notions coincide for the class of determinate processes [15].

Definition 4 ([15]). *We say that a protocol $\mathcal{P}$ is* determinate *if whenever $(\mathcal{P}, \emptyset) \xRightarrow{\ell_1,\ldots,\ell_n} (P, \varphi)$, and $(\mathcal{P}, \emptyset) \xRightarrow{\ell_1,\ldots,\ell_n} (P', \varphi')$, then for any test $R_1 \stackrel{?}{=} R_2$, we have that:*

$$(R_1 = R_2)\varphi \text{ if, and only if } (R_1 = R_2)\varphi'.$$

Determinacy of a protocol can be achieved through sufficient syntactic conditions, e.g. enforcing action-determinism [9]: all executions of an action determinate process ensure that we cannot reach a process P where the same action may lead to two different processes, e.g. we forbid the set of processes generated by $(\mathbf{out}(c, a).P_1) \parallel ((\mathbf{out}(c, b).P_2)$ but allow $(\mathbf{out}(c_1, a).P_1) \parallel ((\mathbf{out}(c_2, b).P_2)$. Action-determinism is automatically checked by AKISS. Whenever processes are not determinate, the above equivalence can be used to disprove trace equivalence, i.e., find attacks. The capability of AKISS to under approximate trace equivalence consists in finding a one-to-one mapping between each process of $\mathcal{P}$ and $\mathcal{Q}$ such that the pair of processes, which are determinate by construction, are equivalent. Such an approach is still possible with our procedure. In this paper we develop a procedure which checks trace equivalence on determinate processes and may be used for finding attacks on general processes.

3 Modelling using Horn clauses

Our decision procedure is based on a fully abstract modelling of a process in first-order Horn clauses which has initially been developed in [15] and adapted to support the Xor operator in [8]. In this section we recall the main definitions and theorems of [8].

3.1 Predicates

We define the set of *symbolic runs*, denoted $u, v, w, \ldots$, as the set of finite sequences of symbolic labels:

$$u, v, w := \epsilon \mid \ell, w$$

with $\ell \in \{\mathbf{in}(c, t), \mathbf{out}(c), \mathbf{test} \mid t \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X}), c \in \mathcal{C}h\}$

The empty sequence is denoted by ϵ. Intuitively, a symbolic run stands for a set of possible runs of the protocol. We denote $u \sqsubseteq_E v$ when u is a prefix (modulo E) of v.

$(P_0, \varphi_0) \models \mathsf{r}_{\ell_1,\ldots,\ell_n}$ if $(P_0, \varphi_0) \xrightarrow{L_1} (P_1, \varphi_1) \ldots \xrightarrow{L_n} (P_n, \varphi_n)$ such that $\ell_i =_E L_i\varphi_{i-1}$ for all $1 \leq i \leq n$

$(P_0, \varphi_0) \models \mathsf{k}_{\ell_1,\ldots,\ell_n}(R, t)$ if when $(P_0, \varphi_0) \xrightarrow{L_1} (P_1, \varphi_1) \xrightarrow{L_2} \ldots \xrightarrow{L_n} (P_n, \varphi_n)$ such that $\ell_i =_E L_i\varphi_{i-1}$ for all $1 \leq i \leq n$, then $\varphi_n \vdash_R t$

$(P_0, \varphi_0) \models \mathsf{i}_{\ell_1,\ldots,\ell_n}(R, R')$ if there exists t such that $(P_0, \varphi_0) \models \mathsf{k}_{\ell_1,\ldots,\ell_n}(R, t)$ and $(P_0, \varphi_0) \models \mathsf{k}_{\ell_1,\ldots,\ell_n}(R', t)$

$(P_0, \varphi_0) \models \mathsf{ri}_{\ell_1,\ldots,\ell_n}(R, R')$ if $(P_0, \varphi_0) \models \mathsf{r}_{\ell_1,\ldots,\ell_n}$ and $(P_0, \varphi_0) \models \mathsf{i}_{\ell_1,\ldots,\ell_n}(R, R')$

Fig. 2: Semantics of atomic formulas

We assume a set $\mathcal{Y}$ of *recipe variables* disjoint from $\mathcal{X}$, and we use capital letters X, Y, Z to range over $\mathcal{Y}$. We assume that such variables may only be substituted by terms in $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup \mathcal{W} \cup \mathcal{Y})$.

We consider four kinds of predicates over which we construct the atomic formulas of our logic. Below, w denotes a symbolic run, R, R' are terms in $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup \mathcal{W} \cup \mathcal{Y})$, and t is a term in $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$. Informally, these predicates have the following meaning (see Figure 2 for the formal semantics).

- r_w holds when the run represented by w is executable;
- $\mathsf{k}_w(R, t)$ holds if whenever the run represented by w is executable, the message t can be constructed by the intruder using the recipe R;
- $\mathsf{i}_w(R, R')$ holds if whenever the run w is executable, R and R' are recipes for the same term; and
- $\mathsf{ri}_w(R, R')$ is a short form for the conjunction of the predicates r_w and $\mathsf{i}_w(R, R')$.

A (ground) atomic formula is interpreted over a pair consisting of a process P and a frame φ, and we write $(P, \varphi) \models f$ when the atomic formula f holds for (P, φ) or simply $P \models f$ when φ is the empty frame. We consider first-order formulas built over the above atomic formulas and the usual connectives (conjunction, disjunction, negation, implication, existential and universal quantification). The semantics is defined as expected, but the domain of quantified variables depends on their type: variables in $\mathcal{X}$ may be mapped to any term in $\mathcal{T}(\Sigma, \mathcal{N})$, while recipe variables in $\mathcal{Y}$ are mapped to recipes, i.e. terms in $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup \mathcal{W})$.

3.2 Statements and saturation

We now identify a subset of the formulas, which we call *statements*. Statements will take the form of Horn clauses, and we shall be mainly concerned with them.

Definition 5 ([15]). *A* statement *is a Horn clause of the form* $H \Leftarrow \mathsf{k}_{u_1}(X_1, t_1), \ldots, \mathsf{k}_{u_n}(X_n, t_n)$ *where:*

– $H \in \{\mathsf{r}_{u_0},\ \mathsf{k}_{u_0}(R,t),\ \mathsf{i}_{u_0}(R,R'),\ \mathsf{ri}_{u_0}(R,R')\}$;
– $u_0, u_1, \ldots, u_n$ *are symbolic runs such that* $u_i \sqsubseteq_E u_0$ *for any* $i \in \{1,\ldots,n\}$;
– $t, t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$;
– $R, R' \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup \mathcal{W} \cup \mathcal{Y})$; *and*
– $X_1, \ldots, X_n$ *are distinct variables from* $\mathcal{Y}$.

Lastly, $vars(t) \subseteq vars(t_1, \ldots, t_n)$ *when* $H = \mathsf{k}_{u_0}(R,t)$.

In the definition above, we implicitly assume that all variables are universally quantified, i.e. all statements are ground. By abuse of language we sometimes call σ a grounding substitution for a statement $H \Leftarrow B_1, \ldots, B_n$ when σ is grounding for each of the atomic formulas $H, B_1, \ldots, B_n$.

In [15], the authors present a saturation-based procedure sat that given a ground process P produces a fully abstract set of solved statements $\mathsf{sat}(P)$. The procedure starts by translating P and the equational theory into a finite set of statements. Then this set is saturated by applying Horn clause resolution rules. Finally, if the procedure terminates (which is guaranteed for subterm convergent equational theories), the set of solved statements K produced by the saturation procedure is a sound and complete abstraction of P: any statement that holds on the protocol P is a logical consequence of K. The notion of logical consequence is formalised through the (infinite) set $\mathcal{H}_{\mathsf{e}}(K)$.

Definition 6 ([8]). *Given a set* K *of statements,* $\mathcal{H}(K)$ *is the smallest set of ground facts that is closed under the rules of Figure 3. We define* $\mathcal{H}_{\mathsf{e}}(K)$ *to be the smallest set of ground facts containing* $\mathcal{H}(K)$ *and that is closed under the rules of Figure 4.*

$$\textsc{Conseq}\ \frac{f = \Big(H \Leftarrow B_1, \ldots, B_n\Big) \in K \qquad \sigma \text{ grounding for } f \qquad B_1\sigma \in \mathcal{H}(K),\ \ldots,\ B_n\sigma \in \mathcal{H}(K)}{H\sigma \in \mathcal{H}(K)} \qquad \textsc{Extend}\ \frac{\mathsf{k}_u(R,t) \in \mathcal{H}(K)}{\mathsf{k}_{uv}(R,t) \in \mathcal{H}(K)}$$

Fig. 3: Rules of $\mathcal{H}(K)$

$$\textsc{Refl}\ \frac{}{\mathsf{i}_w(R,R) \in \mathcal{H}_{\mathsf{e}}(K)} \qquad \textsc{Cong}\ \frac{\mathsf{i}_w(R_1,R_1'), \ldots, \mathsf{i}_w(R_n,R_n') \in \mathcal{H}_{\mathsf{e}}(K) \quad \mathsf{f} \in \Sigma}{\mathsf{i}_w(\mathsf{f}(R_1,\ldots R_n), \mathsf{f}(R_1',\ldots R_n')) \in \mathcal{H}_{\mathsf{e}}(K)}$$

$$\textsc{Ext}\ \frac{\mathsf{i}_u(R,R') \in \mathcal{H}_{\mathsf{e}}(K)}{\mathsf{i}_{uv}(R,R') \in \mathcal{H}_{\mathsf{e}}(K)} \qquad \textsc{Eq. Conseq.}\ \frac{\mathsf{k}_w(R,t) \in \mathcal{H}(K) \qquad \mathsf{i}_w(R,R') \in \mathcal{H}_{\mathsf{e}}(K)}{\mathsf{k}_w(R',t) \in \mathcal{H}_{\mathsf{e}}(K)}$$

Fig. 4: Rules of $\mathcal{H}_{\mathsf{e}}(K)$

Theorem 1 ([8]). *Let* $K = \mathsf{sat}(P)$ *for some ground process* P. *We have that:*

– $P \models f$ *for any* $f \in K \cup \mathcal{H}_{\mathsf{e}}(K)$;
– *If* $(P, \emptyset) \xrightarrow{L_1,\ldots,L_n} (Q, \varphi)$ *then*

1. $\mathsf{r}_{L_1\varphi,\ldots,L_n\varphi} \in_E \mathcal{H}_{\mathsf{e}}(K)$;
2. *if* $\varphi \vdash_R t$ *then* $\mathsf{k}_{L_1\varphi,\ldots,L_n\varphi}(R,t) \in_E \mathcal{H}_{\mathsf{e}}(K)$;
3. *if* $\varphi \vdash_R t$ *and* $\varphi \vdash_{R'} t$, *then* $\mathsf{i}_{L_1\varphi,\ldots,L_n\varphi}(R,R') \in_E \mathcal{H}_{\mathsf{e}}(K)$.

4 Algorithm

We first introduce a few notations and preliminary definitions. We start by introducing the *recipe function*: its goal is to associate a sequence of labels to a symbolic run and a positive process (in the labels we replace the input terms of the symbolic run by the recipes used to deduce them).

Definition 7. *Given a positive process P and a symbolic run $\ell_1,\ldots,\ell_k$ we define a function* $\mathsf{rec}_P(\ell_1\ldots\ell_k) = L_1\ldots L_k$ *where*

$$L_i = \begin{cases} \boldsymbol{in}(c,R) \text{ if } \ell_i = \boldsymbol{in}(c,t) \text{ and } \mathsf{k}_{\ell_1\ldots\ell_{i-1}}(R,t) \in \mathcal{H}(\mathsf{sat}(P)\cup K) \\ \ell_i \qquad \text{otherwise} \end{cases}$$

and $K = \{\mathsf{k}_\epsilon(X_i,x_i) \mid 1 \le i \le n, vars(\ell_1,\ldots,\ell_k) = \{x_1,\ldots x_n\}, X_1,\ldots X_n \in \mathcal{Y}$ *are pairwise distinct and fresh*$\}$.

Note that several functions may satisfy the specification of this definition. Here we consider any possible implementation of this specification, e.g., the one presented in [15]. The complicated case is when the symbolic label is an input: in that case we need to retrieve the corresponding recipe in $\mathsf{sat}(P)$. As the symbolic labels may not be closed we simply enhance $\mathsf{sat}(P)$ with a recipe X for each variable x (the set K).

To check equivalence between processes we rely on the notion of *reachable identity test* written $\mathsf{RId}_{L_1,\ldots,L_k}(R,R')$ where $L_1,\ldots,L_k$ are (not necessarily ground) labels and R,R' (not necessarily ground) recipes. For a test t we denote by $\mathsf{lbl}(t)$ its sequence of labels $L_1,\ldots,L_k$. For commodity reason, we also define a reachability test as: $\mathsf{R}_{L_1,\ldots,L_k} \hat{=} \mathsf{RId}_{L_1,\ldots,L_k}(0,0)$.

Given a ground process P and a test t the predicate $\mathsf{Ver}_P(t)$ checks whether t holds in P. We define $\mathsf{Ver}_P(\mathsf{RId}_{L_1,\ldots,L_k})(R,R')$ to hold when $(P,\emptyset) \stackrel{L_1\sigma,\ldots,L_k\sigma}{\Longrightarrow} (P',\varphi)$ and $(R\sigma = R'\sigma)\varphi$ where σ is a bijection from $vars(L_1,\ldots,L_k,R,R')$ to fresh names $\{c_1,\ldots,c_n\}$. Finally the predicate is lifted to protocols and we write $\mathsf{Ver}_{\mathcal{P}}(t)$ for $\exists P \in \mathcal{P}.\ \mathsf{Ver}_P(t)$.

We note that when $\mathsf{Ver}_P(t)$ holds and P is positive then $\mathsf{Ver}_P(t\sigma)$ holds for any σ, as equality is stable by substitution. However, a disequality may hold when instantiated by distinct fresh names, while a different instantiation may make the test fail.

Next we define the process $\mathbf{rm}^{\neq}(P)$ which simply removes all inequality tests.

Definition 8. *Let P be a process such that $P = P_1.[t_1 \neq t'_1].P_2.\ldots.[t_m \neq t'_m].P_{m+1}$ and $P_1.P_2.\ldots.P_{m+1}$ is positive. We define the process* $\mathbf{rm}^{\neq}(P) \hat{=} P_1.\ldots.P_{m+1}$.

Given a process P we define the set of reachable identity tests $\mathsf{Test}^{\mathsf{RId}}(P)$.

$$\begin{aligned}\mathsf{Test}^{\mathsf{RId}}(P) = \{&\mathsf{RId}_{L_1,\ldots,L_k}(R,R') \mid \\ &\mathsf{ri}_{\ell_1,\ldots,\ell_k}(R,R') \Leftarrow \mathsf{k}_{u_1}(X_1,x_1),\ldots,\mathsf{k}_{u_n}(X_n,x_n) \in \mathsf{sat}(\mathbf{rm}^{\neq}(P)), \\ &L_1,\ldots,L_k = (\mathsf{rec}_P(\ell_1,\ldots,\ell_k))\sigma \text{ where } \sigma = \{X_j \mapsto X_{\min\{i \mid x_i = x_j\}} \mid 1 \le j \le k\}, \\ &\mathsf{Ver}_P(\mathsf{RId}_{L_1,\ldots,L_k}(R,R'))\}\end{aligned}$$

We also define reachability tests $\mathsf{Test}^{\mathsf{R}}(P)$ for a process P:

$$\mathsf{Test}^{\mathsf{R}}(P) = \{t \mid t \in \mathsf{Test}^{\mathsf{RId}}(P), \quad \exists L_1, \ldots, L_k,\ t = \mathsf{R}_{L_1,\ldots,L_k}\}$$

We note that we can only apply the sat function to positive processes. If P was already a positive process $\mathsf{Ver}_P(t)$ would hold for each of the constructed tests because of the soundness of sat. However, in general, $\mathsf{Ver}_{\mathbf{rm}^{\neq}(P)}(t)$ may hold while $\mathsf{Ver}_P(t)$ does not hold, which is why we explicitly test the validity of t in P.

In [15] it is shown that given a positive ground process P and a positive determinate protocol $\mathcal{Q}$, we have that

$$P \sqsubseteq_t \mathcal{Q} \quad \text{iff} \quad \forall t \in \mathsf{Test}^{\mathsf{RId}}(P).\ \mathsf{Ver}_{\mathcal{Q}}(t)$$

which can be used to check trace inclusion between protocols (as $\mathcal{P} \sqsubseteq_t \mathcal{Q}$ iff $\forall P \in \mathcal{P}.P \sqsubseteq_t \mathcal{Q}$) and trace equivalence (as $\mathcal{P} \approx_t \mathcal{Q}$ iff $\mathcal{P} \sqsubseteq_t \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq_t \mathcal{P}$). This result does however not hold for processes with disequality tests.

Example 4. Let $P = \mathbf{in}(c,x).\mathbf{out}(c,a)$ and $Q = \mathbf{in}(c,x).[x \neq a].\mathbf{out}(c,a)$. We have that $P \not\approx_t Q$ but all tests that hold on P also hold on Q (and vice-versa). In particular $\mathsf{R}_{\mathbf{in}(c,X).\mathbf{out}(c)} \in \mathsf{Test}^{\mathsf{R}}(P)$ holds in Q, as $(Q,\emptyset) \xrightarrow{\mathbf{in}(c,c_1).\mathbf{out}(c)} (0,\varphi)$ for a fresh name c_1.

Whenever a test holds on $\mathbf{rm}^{\neq}(P)$ but not on P, it must be that a disequality test in P did not hold. We therefore compute the *complement* of a process, which is the set of positive processes which transforms a disequality into an equality and removes remaining disequalities.

Definition 9. *Let P be a process such that*

$$P = P_1.[t_1 \neq t'_1].P_2.\ldots.[t_m \neq t'_m].P_{m+1}$$

and $P_1.P_2.\ldots.P_{m+1}$ is positive. We define the complement of P, $\mathsf{comp}(P)$ to be the set

$$\{P_1.P_2.\ldots.P_{i-1}.[t_i = t'_i].P_i.\ldots.P_m.P_{m+1} | 1 \leq i \leq m\}$$

We easily see that we have the following property.

Lemma 1. *Let P be a process and t a test. We have that*

$$\mathsf{Ver}_{\mathbf{rm}^{\neq}(P)}(t) \textit{ iff either } \mathsf{Ver}_P(t) \textit{ or } \mathsf{Ver}_{\mathsf{comp}(P)}(t)$$

Lastly, before explaining our algorithm we need to introduce the *shrink* operator on processes which is used in conjunction with the Inst operators on sequences of labels. Given a process P and a sequence of labels lbl we define a process that only executes instances of lbl (up to test actions which are ignored). In the following we suppose that variables in $\mathcal{Y}, \mathcal{X}$ and names in $\mathcal{N}$ are totally ordered by an order $<_{\mathcal{Y}}$, $<_{\mathcal{X}}$ resp. $<_{\mathcal{N}}$.

Definition 10. *Let P be a process, lbl a sequence of labels, σ an increasing bijection from $vars(lbl) \cap \mathcal{Y}$ to a set of fresh and pairwise distinct term variable in $\mathcal{X}$, θ an increasing bijection from $vars(lbl) \cap \mathcal{Y}$ to a set of fresh and pairwise distinct names in $\mathcal{N}$, such that $(P,\emptyset) \xrightarrow{L_1\theta,\ldots,L_n\theta} (P',\varphi)$. Let lbl_0 be the subsequence of lbl obtained by removing all **test** labels. We define $\mathsf{shrink}_{lbl}(P)$ as $\mathsf{shr}^{\emptyset}_{lbl_0}(P)$ where*

- $\mathsf{shr}^v_{lbl}([s \sim t].P) = [s \sim t].\mathsf{shr}^v_{lbl}(P) \quad$ *for* $\sim \in \{=, \neq\}$
- $\mathsf{shr}^v_{\boldsymbol{out}(c)\cdot lbl}(\boldsymbol{out}(c,t).P) = \boldsymbol{out}(c,t).\mathsf{shr}^v_{lbl}(P)$,
- $\mathsf{shr}^v_{\boldsymbol{in}(c,R)\cdot lbl}(\boldsymbol{in}(c,x).P)$
  $= \boldsymbol{in}(c_s, X_1\sigma).\ldots.\boldsymbol{in}(c_s, X_n\sigma).\boldsymbol{in}(c,x).[x = R\varphi\theta^{-1}\sigma].\mathsf{shr}^{vars(R)\cup v}_{lbl\sigma}(P)$
  where $vars(R) \cap \mathcal{Y} \setminus v = \{X_1, \ldots, X_n\}$ *and* $X_i < X_{i+1}$,
- 0 *otherwise*

and c_s *is a dedicated channel not appearing in* P.

Note that the function shrink depends on the chosen bijection but this only changes the process up to alpha renaming. Note that we cannot force an execution to contain an instance of a particular recipe R. The inserted test $[x = R\varphi\sigma]$ only ensures that the input of x is produced by some recipe R' such that $(R\sigma = R')\varphi$. We therefore additionally add inputs $\mathbf{in}(c_s, x_i)$ which will allow us to retrieve instance $R\theta$ of R that yields the same protocol message as R.

Definition 11. *Let* lbl, lbl' *be sequences of labels. If*

$$\boldsymbol{in}(lbl) = \boldsymbol{in}(c_s, R^1_1)\ldots\boldsymbol{in}(c_s, R^{n_1}_1).in(c_1, R_1).\ldots.\boldsymbol{in}(c_s, R^k_1)\ldots\boldsymbol{in}(c_s, R^{n_k}_k).in(c_k, R_k);$$

$$\boldsymbol{in}(lbl') = \boldsymbol{in}(c_1, R'_1).\ldots.\boldsymbol{in}(c_k, R'_k) \text{ and } vars(R'_i) \setminus \bigcup_{j<i} vars(R'_j) = \{X^1_i, \ldots, X^{n_i}_i\}$$

with $X^j_i < X^{j+1}_i$ *then we define* $\mathsf{Inst}(lbl, lbl') = \{X^j_i \mapsto R^j_i \mid 1 \le i \le k, 1 \le j \le n_i\}$. *Otherwise* $\mathsf{Inst}(lbl, lbl') = \bot$.

Example 5. Let $P = \mathbf{in}(c,x).\mathbf{out}(c,0)$ be a process and $lbl = \mathbf{in}(c, \langle Y, 0\rangle), \mathbf{out}(c)$ a sequence of labels. Consider two bijections $\sigma = \{Y \mapsto y\}$ and $\theta = \{Y \mapsto a\}$. The process $\mathsf{shrink}_{lbl}(P) = \mathbf{in}(c_s, y).\mathbf{in}(c,x).[x = \langle y, 0\rangle].\mathbf{out}(c,0)$ allows to identify the recipes Y such that $(P,\varphi) \xrightarrow{lbl\sigma} (P',\varphi')$. Indeed, assume $\mathsf{Test}^{\mathsf{R}}(\mathsf{shrink}_{lbl}(P))$ contains $r = \mathsf{R}_{\mathbf{in}(c_s,h(0)).\mathbf{in}(c,\langle h(0),0\rangle.\mathbf{out}(c)}$, then $r\tau = \mathsf{R}_{\mathbf{in}(c,\langle h(0),0\rangle.\mathbf{out}(c)}$ where $\tau = \mathsf{Inst}(lbl, \mathsf{lbl}(r))$ is such that $\mathsf{Ver}_{r\tau}(P)$ and $\mathsf{lbl}(r\tau)$ is an instance of l.

The algorithm `Equiv` for verifying trace equivalence on determinate processes is detailed in Algorithm 1.

Theorem 2. *Let* $\mathcal{P}$ *and* $\mathcal{Q}$ *be two determinate protocols. Then we have that*

$$\mathcal{P} \approx_t \mathcal{Q} \quad \textit{iff} \quad \texttt{Equiv}(P \approx_t \mathcal{Q})$$

The algorithm proceeds as follows. For each $P \in \mathcal{P}$ we check whether all traces of P are included in $\mathcal{Q}$. For this we compute the set Rid_P of reachable identity tests that hold in P and that need to be checked on $\mathcal{Q}$. We next pick a test rid from the set (and remove it from the set, denoted $rid := \mathrm{pop}(Rid_P)$) and check whether this test holds for some process Q in $\mathcal{Q}$. If this is not the case we violate trace equivalence. Otherwise we need to perform additional checks: indeed even if the test rid holds, an instance of rid might not hold on Q but still hold in P. Consider the following simple example:

$$P = \mathbf{in}(c,x).\mathbf{out}(c,a) \qquad Q_1 = \mathbf{in}(c,x).[x \neq a].\mathbf{out}(c,a)$$
$$\mathcal{Q} = \{Q_1, Q_2\} \qquad Q_2 = \mathbf{in}(c,x).[x = a].\mathbf{out}(c,a)$$

Algorithm 1: Decision procedure for $\mathcal{P} \approx_t \mathcal{Q}$

Function `Check` $(P, \mathcal{Q})$
- **Input:** Process P, protocol $\mathcal{Q}$
- **Output:** Boolean

$Rid_P := \mathsf{Test}^{\mathsf{RId}}(P)$;
while $Rid_P \neq \emptyset$ **do**
- $rid := \mathrm{pop}(Rid_P)$;
- $S_Q := \{Q \in \mathcal{Q} \mid \mathsf{Ver}_Q(rid)\}$;
- **if** $S_Q = \emptyset$ **then return** false;
- $Q := \mathrm{pop}(S_Q)$;
- **foreach** $\overline{Q} \in \mathsf{comp}(\mathsf{shrink}_{\mathsf{lbl}(rid)}(Q))$ **do**
  - $R_{\overline{Q}} := \{r \in \mathsf{Test}^{\mathsf{R}}(\overline{Q}) \mid |\mathsf{lbl}(r)| = |\overline{Q}|\}$;
  - **foreach** $r \in R_{\overline{Q}}$ **do**
    - $\sigma := \mathsf{Inst}(\mathsf{lbl}(r), \mathsf{lbl}(rid))$;
    - **if** $\mathsf{Ver}_P(\mathsf{R}_{\mathsf{lbl}(rid)}\sigma)$ **then** $Rid_P := Rid_P \cup \{rid\sigma\}$;

return true

Function `Equiv` $(\mathcal{P}, \mathcal{Q})$
- **Input:** Protocols $\mathcal{P}, \mathcal{Q}$
- **Output:** Boolean

return $\bigwedge_{P \in \mathcal{P}}$ `Check` $(P, \mathcal{Q}) \wedge \bigwedge_{Q \in \mathcal{Q}}$ `Check` $(Q, \mathcal{P})$

Note that $P \approx_t \mathcal{Q}$. Let $rid = \mathsf{RId}_{\mathbf{in}(c,X).\mathbf{out}(c)}(a, a)$. This test holds in Q_1. However, the more instantiated test $\mathsf{RId}_{\mathbf{in}(c,a).\mathbf{out}(c)}(a, a)$ would not hold. (Note that the test may actually only fail because reachability is violated.) We therefore need to identify the instances of rid Q_1 that do not hold on Q_1. The process $\mathsf{shrink}_Q(\mathsf{lbl}(rid))$ defines the process that only verifies instances of rid. Computing its complement defines the processes that verify the instances of rid that are not verified by Q: in our example we would identify the test $r = \mathsf{R}_{\mathbf{in}(c,a).\mathbf{out}(c)}$, as computing the complement transforms $[x \neq a]$ into $[x = a]$. Finally, we check whether r is verified by P. If this is the case, we add the more instantiated test $\mathsf{RId}_{\mathbf{in}(c,a).\mathbf{out}(c)}(a, a)$ to the set Rid_P of tests to be checked. We note that the fact that Q_1 does not verify r, but P does is not yet a violation of trace equivalence: another trace in $\mathcal{Q}$ may well verify the instantiated test. In our example, indeed the process Q_2 verifies $\mathsf{RId}_{\mathbf{in}(c,a).\mathbf{out}(c)}(a, a)$.

Theorem 2 above ensures partial correctness, i.e., soundness and completeness. We now state that total correctness only depends on the termination of sat.

Theorem 3. *If procedure* sat *terminates then procedure* `Equiv` *terminates.*

As it was shown in [15], termination of sat is ensured for a wide class of subterm convergent equational theories. While sat may not terminate in general on other theories such as xor, or blind signatures, the tool does terminate in practice on a wide range of examples [15,8] (and Theorem 2 ensures the correctness of the result).

5 Implementation and case studies

5.1 The AKISS tool

In addition to parallel composition $P \parallel Q$ and conditionals AKISS also supports non-deterministic choice $P +\!+ Q$, sequences $P :: Q$ and phases $P \gg Q$, which are convenient for defining complex scenarios under which we analyse protocols. A sequence $P :: Q$ contains all sequences of a trace of P followed by a trace of Q while the set of traces for a phase $P \gg Q$ contains all traces made of the beginning of a trace of P followed by a full trace of Q.

We model unlinkability for two sessions in each of the protocols below as follows:

$$P_A^1 \gg P_A^2 \approx P_A^1 \gg P_B^2$$

The attacker first interacts with a first session of protocol P executed by A, denoted P_A^1. Then, in a new phase he interacts with a second session of the protocol, which is either executed by A (process P_A^2) or by B (process P_A^2). The protocol P satisfies unlinkability if the two scenarios cannot be distinguished. Note that the use of the phase operator is preferable to the sequential composition, as an attacker may not be able to finish the first session completely before starting the second session.

The implementation of the tool and the files corresponding to our case studies are freely available at `https://github.com/akiss/`.

5.2 The AKA protocol

Unlike the simplified version of AKA described in Figure 1, the actual AKA protocol [25] provides a mechanism against replay attacks. In addition to the mac value, both the network and the mobile station store a counter SQN used as a timestamps: each time the network station starts a session with a same mobile station, it sends in addition to the random value and the mac an obfuscated message $SQN \oplus k_{IMSI}$ containing the incremented value of the counter. The mobile stores the maximum value which has been received. If the received value is not strictly greater than this maximum, the mobile sends a synchronization error message. Otherwise it updates the stored value.

Unlinkability is modelled as explained above: in a first session a mobile station A interacts with the network station and in a second session either mobile A or mobile B interact with the network station. The AKISS tool does not allow for comparison of integer. Instead we just check that the sent value was not the same as a previous one. Therefore during the first phase, since there was no SQN value sent to the mobile there is no need to perform a check while in the second session of the mobile A (in the first scenario) we check that the new SQN value is distinct from the first time.

Using the AKISS tool we find the (previously known) attack consisting of observing the first phase and sending the network station's message of the first phase in the second one: if the second phase is with the same mobile station then it sends a synchronization error message while if its another mobile station it sends a mac error message. Running our tool on a 30 core Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10GHz, the attack is found in 3 minutes.

5.3 Unlinkability on some other protocols

We also analysed the Basic Access Control (BAC) protocol [26], the Private Authentication Protocol (PAP) [2] and two RFID protocols [33]: LAK and SLK.

All these protocols use else branches to send error messages except for the LAK RFID protocol. However, even though this protocol does not contain branches, the scenarios required for expressing unlinkability does requires the use of an else branch for the key update. Indeed, when a session succeeds, the key is updated for the next session, while the previous key is reused in case of failure. This results into an *if then else* structure. Finally, for the SLK and LAK protocols where both the tag and the reader update their data, the scenarios to consider for two sessions are the following.

$$\begin{array}{rcl} Psame & = & ((Tag_{Aa} \parallel Reader_a) \gg (Tag_A \parallel Reader)) \\ & ++ & ((Tag_A \parallel (Reader_a \gg 0)) :: (Tag_{Au} \parallel Reader)) \\ & ++ & (((Tag_{Aa} \gg 0) \parallel Reader) :: (Tag_A \parallel Reader_u)) \\ & ++ & ((Tag_A \parallel Reader) :: (Ttag_{Au} \parallel Reader_u)) \end{array}$$

The roles with index u model the role with a preliminary update **if test then** R **else** R' where R' is R with updated values. As the update test is not defined before all inputs have been received, we introduce R_a for each role R to be the process where the last input and the update test are missing (and moreover the construct $R \gg 0$ allows the adversary to stop that instance of the role even earlier): in this case, the update will not happen anyway. Therefore, our scenario has four cases depending on whether the tag, the reader, both or none have reached the update test or not. The scenario where the instances of the tags corresponds to different tag is similar.

$$\begin{array}{rcl} Pdiff & = & ((Tag_A \parallel Reader_a) \gg (Tag_B \parallel Reader)) \\ & ++ & (((Tag_A \gg 0) \parallel Reader) :: (Tag_B \parallel Reader_u)) \end{array}$$

Note that as we consider a different tag in the second session we do not need to worry whether Tag_A was updated or not.

The AKISS tool establishes the equivalence for PAP in 4s. It finds known attacks on BAC in 1m30, on SLK in 6s and in 7h for LAK. The much longer time for LAK is due to the particular use of xor which leads to complex unifiers.

6 Conclusion

In this paper we present an extension of AKISS which allows automated verification of protocols with else branches. An appealing aspect of our approach is that we do not modify the saturation procedure underlying the AKISS tool. As a result we obtain a new decidability result for the class of subterm convergent equational theories and an effective automated analysis tool. We have been able to analyse several protocols including the AKA protocol, and RFID protocols which require both support for xor and else branches.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM, 2001.
2. M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.
3. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.
4. 3GPP. Technical specification group services and system aspects; 3G security; security architecture (release 9). Technical report, Technical Report TS 33.102 V9.3.0, 3rd Generation Partnership Project, 2010., 2010.
5. M. Arapinis, T. Chothia, E. Ritter, and M. D. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Comp. Soc., 2010.
6. M. Arapinis, L. I. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *19th Conference on Computer and Communications Security (CCS'12)*, pages 205–216. ACM, 2012.
7. A. Armando et al. The AVISPA tool for the automated validation of internet security protocols and applications. In *17th International Conference on Computer Aided Verification (CAV'05)*, LNCS, pages 281–285. Springer, 2005.
8. D. Baelde, S. Delaune, I. Gazeau, and S. Kremer. Symbolic verification of privacy-type properties for security protocols with XOR. In *30th Computer Security Foundations Symposium (CSF'17)*. IEEE Comp. Soc., 2017. To appear.
9. D. Baelde, S. Delaune, and L. Hirschi. Partial order reduction for security protocols. In *26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *LIPICS*, pages 497–510. Leibniz-Zentrum für Informatik, 2015.
10. D. A. Basin, J. Dreier, and R. Sasse. Automated symbolic proofs of observational equivalence. In *22nd Conference on Computer and Communications Security (CCS'15)*, pages 1144–1155. ACM, 2015.
11. M. Baudet. Deciding security of protocols against off-line guessing attacks. In *12th Conference on Computer and Communications Security (CCS'05)*, pages 16–25. ACM, 2005.
12. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *Symposium on Security and Privacy (S&P'04)*, pages 86–100. IEEE Comp. Soc., 2004.
13. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 75(1):3–51, 2008.
14. B. Blanchet, B. Smyth, and V. Cheval. *Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2016.
15. R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 23(4), 2016.
16. V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: symbolic equivalence of constraint systems. In *International Joint Conference on Automated Reasoning (IJCAR'10)*, volume 6173 of *LNAI*, pages 412–426. Springer, 2010.
17. V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *18th Conference on Computer and Communications Security (CCS'11)*, pages 321–330. ACM, 2011.
18. V. Cheval, V. Cortier, and S. Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492:1–39, 2013.

19. Y. Chevalier and M. Rusinowitch. Decidability of equivalence of symbolic derivations. *Journal of Automated Reasoning*, 2010. To appear.
20. T. Chothia and V. Smirnov. A traceability attack against e-passports. In *14th International Conference on Financial Cryptography and Data Security (FC'10)*, volume 6052 of *LNCS*, pages 20–34. Springer, 2010.
21. C. J. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of *LNCS*, pages 414–418. Springer, 2008.
22. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.
23. T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.1. RFC 4346, Internet Engineering Task Force, 2008.
24. S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*, volume 5705 of *LNCS*, pages 1–50. Springer, 2009.
25. I. Gazeau and S. Kremer. Automated analysis of equivalence properties for security protocols using else branches (extended version). Research report, Inria Nancy - Grand Est, 2017. `https://hal.inria.fr/hal-01547017`.
26. Machine readable travel documents. Doc 9303, International Civil Aviation Organization (ICAO), 2008.
27. C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet key exchange protocol version 2 (ikev2). RFC 7296, Internet Engineering Task Force, 2014.
28. C. Neuman, S. Hartman, and K. Raeburn. The kerberos network authentication service (v5). RFC 4120, Internet Engineering Task Force, 2005.
29. P. Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol. A cautionary tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.
30. S. Santiago, S. Escobar, C. Meadows, and J. Meseguer. A formal definition of protocol indistinguishability and its verification using Maude-NPA. In *10th International Workshop on Security and Trust Management (STM'14)*, volume 8743, pages 162–177. Springer, 2014.
31. B. Schmidt, S. Meier, C. Cremers, and D. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.
32. A. Tiu and J. Dawson. Automating open bisimulation checking for the spi-calculus. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 307–321. IEEE Comp. Soc., 2010.
33. T. van Deursen and S. Radomirovic. Attacks on RFID protocols. *IACR Cryptology ePrint Archive*, 2008:310, 2008.
34. F. Yang, S. Escobar, C. A. Meadows, J. Meseguer, and S. Santiago. Strand spaces with choice via a process algebra semantics. In *18th International Symposium on Principles and Practice of Declarative Programming (PPDP'16)*, pages 76–89. ACM, 2016.