Formal verification of protocols based on short authenticated strings

Stéphanie Delaune, Steve Kremer, Ludovic Robin

Formal verification of protocols based on short authenticated strings

Stéphanie Delaune*, Steve Kremer†, and Ludovic Robin†
*CNRS & IRISA, Rennes, France
†LORIA, INRIA Nancy - Grand-Est, France

***Abstract*—Modern security protocols may involve humans in order to compare or copy short strings between different devices. Multi-factor authentication protocols, such as Google 2-factor or 3D-secure are typical examples of such protocols. However, such short strings may be subject to brute force attacks. In this paper we propose a symbolic model which includes attacker capabilities for both guessing short strings, and producing collisions when short strings result from an application of weak hash functions. We propose a new decision procedure for analysing (a bounded number of sessions of) protocols that rely on short strings. The procedure has been integrated in the AKISS tool and tested on protocols from the ISO/IEC 9798-6:2010 standard.**

I. Introduction

Modern security protocols often include the human as a participant on his own, i.e., separately from the device he may own and use. Such security protocols are sometimes called *security ceremonies* [18]. In this paper we are interested in protocols where a human has to copy, or compare short strings, or verification codes on two devices. This kind of situations occurs rather frequently. In multi-factor authentication protocols, such as 3D-secure [1] or Google 2-factor [21], a short verification code is sent by SMS to the user's phone who has to copy it to another device (to prove ownership of his phone). Another example is bluetooth tethering [9], [10] where the human user has to compare codes displayed by two devices to be connected. There has also been a recent standard [23] defining several mechanisms for "manual" data transfer: these protocols use a human to ensure that two devices can exchange some data in an authentic way without sharing any cryptographic keys, nor relying on a PKI. One important aspect is that the user introduces a *low bandwidth* channel: he is only able to copy (or compare) short strings. The ISO/IEC 9798-6:2010 standard [23] suggests that such short strings are 16-20 bits long. As pointed out in [25], [26], traditional methods are not suitable to reason about the security of protocols over non standard channels, and the purpose of this paper is to pursue the effort towards formalizing protocols that rely on low bandwith channels.

As an illustrative example, we may consider the simplified commitment scheme [26] which is also described below. This protocol relies on the existence of two *empirical* channels. According to [26], an empirical channel is a low bandwidth channel that is

- *asynchronous*: the attacker may delay the delivery,
- *authentic*: the attacker may not inject messages, and
- *public*: the channel does not guarantee confidentiality.

Such a channel is typically implemented by an out-of-band channel to a human who copies the data. Message transmission on an empirical channel is denoted by $\rightarrow_e$:

$$\begin{array}{llll} 1. & A & \rightarrow & B: \ info_A, \mathsf{hash}(info_A, n_A, r_A) \\ 2. & B & \rightarrow_e & A: \ ack \\ 3. & A & \rightarrow & B: \ n_A \\ 4. & A & \rightarrow_e & B: \ r_A \end{array}$$

The goal of this protocol is to ensure that B will receive the information $info_A$ as sent to him by A. Here r_A is a short random value, whereas n_A is a long random nonce. We will now explain the need of the acknowledgment (sent in message 2) and the nonce n_A (included in the hash and sent in message 3). If we remove the nonce n_A we obtain a weaker protocol. Indeed, once the attacker has obtained the modified message 1, i.e. $info_A, \mathsf{hash}(info_A, r_A)$, he can compute the value of the short random r_A by brute force. Once r_A is known, the attacker can send an alternative information $info_I$ of his choice to B by modifying the first message to $info_I, \mathsf{hash}(info_I, r_A)$. If we remove message 2 a similar attack is possible as the attacker can just intercept messages 1 and 3 and perform a similar brute force attack.

The above example illustrates that short strings, that are subject to brute force, or guessing attacks are extremely tricky. One of the most successful approaches for analysing security protocols is the symbolic approach for automated security protocol analysis, based on the seminal ideas of Dolev and Yao [17]. Indeed, there exist today mature tools for analysing security protocols [7], [28], [4], [19], which have been used to successfully analyse real world protocols such as, e.g., authentication standards [5]. With the exception of a few tools (which we discuss in our related work section below), existing analysis tools do however not take into account adversary capabilities for reasoning about weak secrets. Therefore, we devise in this paper a symbolic protocol verification technique that allows to take such *guesses* into account.

Our contributions. In this paper we extend the AKISS verification tool [11] and the underlying theory to take into account protocols that manipulate short secrets that are subject

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 and No 714955-POPSTAR), as well as from the French National Research Agency (ANR) under the project Sequoia.

to brute force attacks. AKISS allows automated analysis of security protocols when restricted to a bounded number of sessions. Cryptographic primitives may be defined through arbitrary convergent equational theories that have the finite variant property. This class of theories includes standard cryptographic primitives such as symmetric and asymmetric encryption, digital signatures, hash functions, ..., as well as less commonly supported primitives such as blind signatures and zero knowledge proofs. As we will see the framework is actually flexible enough to model *weak* hash functions, i.e., hash functions whose output is short enough to be copied by a human. Termination is guaranteed for theories that are subterm convergent, but the tool may terminate in practice on protocols relying on other theories [11]. Protocols in AKISS are written in a process calculus that is similar to the applied pi calculus [2]. Our first contribution is an extension of the semantics to allow an attacker to learn a weak secret during the execution: the fact that a secret may be guessed is expressed by a test that succeeds if the attacker is given the secret to be guessed, but fails if the secret is replaced by a fresh name. We note that our definitions are stated for arbitrary equational theories. Our modeling is reminiscent of the treatment of guessing attacks in password based protocols [14] (see the related work below). Next, building on the theory underlying AKISS, we design a novel verification procedure to verify authentication properties expressed as reachability properties for our new semantics. Unlike the original theory underlying AKISS we also allow disequality tests (else branches) when verifying reachability properties.

The procedure has been implemented in the AKISS tool and used to analyze several protocols from the ISO/IEC 9798-6:2010 standard [23]. In addition to weak secrets, we also considered weak hash functions, i.e., hash functions whose output is short and that are subject to collisions. We encode the attacker's capability to find collisions through an equational theory in a way that is similar to [12]. Our case studies also required us to consider different kind of channels, depending on whether out-of-band channels are supposed to be synchronous or asynchronous, public or confidential and authentic or not. We propose an encoding for each of these different kinds of channels.

Related work. The works closest to ours are the ones by Roscoe et al. [26] and Delaune and Jacquemard [15]. In [26], they propose the use of the process algebra CSP and the model-checker FDR to analyze protocols relying on weak secrets. Their tool is however restricted to messages of finite length and particular equational theories, i.e., they analyze a finite state system and do not allow for user specified equational theories. In [15], they propose a decision procedure for solving a similar problem than the one studied here. However, they rely on a simpler setting, e.g. no disequality tests, only a fixed set of primitives preventing them to model weak hash functions. More importantly, they do not provide any tool support, and their procedure is actually far from being implementable. In [12], Chothia et al. use the ProVerif tool [8] to analyze commitment protocols. Their work inspired our encoding of weak hash functions, for which collisions may be found, through an equational theory. However, they do not offer support for short random secrets that may be guessed and learned by the adversary during a protocol execution.

Our definition of whether a secret is guessable is similar to the definition of dictionary attacks in password-based protocols [14], which can be encoded in terms of static equivalence and verified using tools able to check equivalence properties, such as [8], [11]. A similar definition was also proposed by Lowe [24] in the CSP process algebra. However, when verifying dictionary attacks we are only interested in knowing whether the password may be learned or not. In the protocols studied in this paper, the short secrets are often revealed anyway (see message 4 in the example above). Therefore, deciding whether a weak name is guessable is not a relevant question in the protocols studied here, and existing approaches cannot be used. Here, we need to dynamically add knowledge of the secret to the attacker knowledge. This is technically more challenging as we may need to consider different secrets that may be needed to be guessed in a same execution for a successful attack. Multiple short secrets naturally arise as soon as several sessions are considered. In particular, using symbolic techniques (that do not explicitly represent the infinite number of all possible executions), it is not sufficient to check separately whether secrets are guessable, as one may need to guess one secret to be able to guess a second one.

There has also been interesting work on modeling human behavior in protocols [6]. We do not consider this aspect, which is orthogonal, in our work.

All files related to the tool implementation and case studies are available at [3] and full proofs are provided in the long version of this paper [16].

Outline. In Section II, we recall some basic definitions for modelling messages. Then, in Section III, we explain how the attacker may extract information from the messages exchanged in the course of a protocol execution. We introduce our formalism for modelling protocols in Section IV. We restrict ourselves to a minimalistic core calculus which is sufficient to develop our main result. Later on (see Section VII), we will introduce a richer calculus and explain how this richer model is translated into the former. In Section V, we provide an abstract representation of processes expressed in our core calculus as Horn clauses. Next, we present our saturation procedure based on Horn clause resolution in Section VI. We present our algorithm, as well as its integration in AKISS in Section VII. Lastly, in Section VIII, we report the results we obtained when analysing two protocols from the ISO/IEC 9798-6:2010 standard.

II. Preliminaries

A. Term algebra

As usual in symbolic models we represent messages as terms. We consider three infinite and disjoint sets of *names*: $\mathcal{N}_{\mathsf{prv}}$ is the set of *private* names, which are used to represent

nonces or keys generated by honest participants, $\mathcal{N}_{\mathsf{pub}}$ is the set of *public* names, which are used to represent identifiers available both to the attacker and to honest participants, as well as attacker nonces, and $\mathcal{N}_{\mathsf{guess}}$ is the set of *guessable* names, which are used to represent data with low entropy such as passwords or PIN numbers. Let $\mathcal{N} = \mathcal{N}_{\mathsf{prv}} \uplus \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}}$. We consider two infinite and disjoint sets of variables, denoted $\mathcal{X}$ and $\mathcal{W}$. Variables in $\mathcal{X} = \{x, y, \ldots\}$ typically refer to unknown parts of messages expected by participants while variables in $\mathcal{W} = \{\mathsf{w}_1, \mathsf{w}_2, \ldots\}$ are used to store messages known by the attacker.

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

We associate an *equational theory* E to the signature Σ. The equational theory is defined by a set of equations of the form $u = v$ where $u, v \in \mathcal{T}(\Sigma, \mathcal{X})$, and induces an equivalence relation over terms: $=_{\mathsf{E}}$ is the smallest congruence relation on terms, which contains all equations $u = v$ in E, and that is closed under substitution of terms for variables.

Example 1: Let $\Sigma = \{\langle\cdot,\cdot\rangle/2, \mathsf{proj}_1/1, \mathsf{proj}_2/1, \mathsf{hash}/1\}$, and consider the equational theory $\mathsf{E}_{\mathsf{hash}}$ defined by adding the following equations:

$$\mathsf{proj}_1(\langle x, y\rangle) = x, \text{ and } \mathsf{proj}_2(\langle x, y\rangle) = y.$$

The function symbol $\langle\cdot,\cdot\rangle$ models pairs, whereas projection functions are denoted proj_1 and proj_2. We use the unary symbol hash to model a standard cryptographic hash function.

Let $m_A = \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle$ with $info_A \in \mathcal{N}_{\mathsf{pub}}$, and $r_A \in \mathcal{N}_{\mathsf{guess}}$. We have that:

$$\mathsf{hash}(\langle \mathsf{proj}_1(m_A), r_A\rangle) =_{\mathsf{E}_{\mathsf{hash}}} \mathsf{proj}_2(m_A).$$

We may also consider equations to model e.g. symmetric and asymmetric encryption. We will see in Section VIII how these equations can be used to model short hashes.

B. Finite variant property

A *rewrite system* $\mathcal{R}$ is a set of rewrite rules of the form $\ell \to r$ where $\ell, r \in \mathcal{T}(\Sigma, \mathcal{X})$, and $\mathrm{vars}(r) \subseteq \mathrm{vars}(\ell)$. A term t can be rewritten to u, denoted $t \to_{\mathcal{R}} u$, if there exists a position p in t, a rule $\ell \to r$ in $\mathcal{R}$ and a substitution σ such that:

- $t|_p = \ell\sigma$, i.e. the term at position p in t is equal to $\ell\sigma$; and
- $u = t[r\sigma]_p$, i.e. u is the term obtained by replacing $t|_p$ with $r\sigma$ in t.

The relation $\to_{\mathcal{R}}^*$ denotes the transitive and reflexive closure of $\to_{\mathcal{R}}$. A rewrite system $\mathcal{R}$ is *convergent* if it is:

- *confluent*: for any t, t_1, t_2 such that $t \to_{\mathcal{R}}^* t_1$ and $t \to_{\mathcal{R}}^* t_2$ there exists u such that $t_1 \to_{\mathcal{R}}^* u$ and $t_2 \to_{\mathcal{R}}^* u$; and
- *terminating*: it does not admit any infinite sequence $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$

We denote by $t{\downarrow_{\mathcal{R}}}$ (or simply $t{\downarrow}$) the normal form of a term t. In the following we only consider equational theories E that can be represented by a rewrite system $\mathcal{R}$ which is convergent, i.e., we have that $u =_{\mathsf{E}} v \Leftrightarrow u{\downarrow_{\mathcal{R}}} = v{\downarrow_{\mathcal{R}}}$.

Example 2: Continuing Example 1, we consider the rewrite system $\mathcal{R}_{\mathsf{hash}}$ given below:

$$\mathsf{proj}_1(\langle x, y\rangle) \to x \qquad \mathsf{proj}_2(\langle x, y\rangle) \to y$$

Actually, we have that $\mathcal{R}_{\mathsf{hash}}$ is convergent and it represents the equational theory $\mathsf{E}_{\mathsf{hash}}$ defined in Example 1.

Given a convergent rewrite system $\mathcal{R}$, we now define the notion of *complete set of variants*, which was first introduced in [13].

Definition 1: Consider a rewrite system $\mathcal{R}$ that is convergent, and a set of terms T. A set of substitutions $\mathrm{variants}_{\mathcal{R}}(T)$ is called a *complete set of variants* for the set of terms T, if for any substitution ω there exist $\sigma \in \mathrm{variants}_{\mathcal{R}}(T)$, and a substitution τ such that $x\omega{\downarrow} = x\sigma{\downarrow}\tau$ for any $x \in \mathrm{vars}(T)$, and $(t\omega){\downarrow} = (t\sigma){\downarrow}\tau$ for any $t \in T$.

The set of variants of t represents a pre-computation such that the normal form of any instance of t is equal to an instance of $t\sigma{\downarrow}$ for some σ in the set of variants, without the need to apply further rewrite steps. A rewrite system has the *finite variant property* if for any sequence of terms a finite, complete set of variants exists and is effectively computable. For the sake of readability, we will often write $\mathrm{variants}_{\mathcal{R}}(t_1, \ldots, t_n)$ instead of $\mathrm{variants}_{\mathcal{R}}(\{t_1, \ldots, t_n\})$.

Example 3: Continuing our running example, let $T_i = \{\mathsf{proj}_i(y)\}$ for $i \in \{1, 2\}$, let σ_1 be the identity substitution and $\sigma_2 = \{y \mapsto \langle y_1, y_2\rangle\}$. The set $\{\sigma_1, \sigma_2\}$ is a complete set of variants for T_1 (resp T_2). Indeed, for any substitution ω, either we have that $\mathsf{proj}_i(y\omega{\downarrow})$ is in normal form, and therefore σ_1 (together with $\tau = \omega{\downarrow}$) will satisfy the requirements. Otherwise, we have that $y\omega{\downarrow}$ is actually a pair $\langle m_1, m_2\rangle$, and we conclude choosing the substitution σ_2 (together with $\tau = \{y_1 \mapsto m_1, y_2 \mapsto m_2\}$).

This finite variant property is satisfied by many equational theories interesting for modelling cryptographic protocols, *e.g.* symmetric and asymmetric encryption, signatures, blind signatures, zero-knowledge proofs. Moreover, this property implies the existence of a complete set of unifiers, and gives us a way to compute it effectively [20].

Definition 2: Consider a convergent rewrite system $\mathcal{R}$. Let $\Gamma = \{u_1 = v_1, \ldots, u_k = v_k\}$ be a set of equations. A set of substitutions Σ is called a *complete set of $\mathcal{R}$-unifiers* for Γ if:

1) each $\sigma \in \Sigma$ is such that $u_i\sigma\downarrow = v_i\sigma\downarrow$ for $i \in \{1, \ldots, k\}$;
2) for each θ such that $u_i\theta\downarrow = v_i\theta\downarrow$ for any $i \in \{1, \ldots, k\}$, there exists $\sigma \in \Sigma$ and a substitution τ such that $x\theta\downarrow = x\sigma\tau\downarrow$ for any $x \in \mathrm{vars}(\Gamma)$.

We denote $\mathrm{csu}_{\mathcal{R}}(\Gamma)$ such a set.

When $\mathcal{R} = \emptyset$, it is well-known that for any

$$\Gamma = \{u_1 = v_1, \ldots, u_k = v_k\}$$

which admits a solution (i.e. there exists σ such that for any $i \in \{1, \ldots, n\}$, we have that $u_i\sigma = v_i\sigma$), such a complete set can be chosen with cardinality one. This element is actually unique up to some renaming and we denote it $\mathrm{mgu}(\Gamma)$ or $\mathrm{mgu}(u, v)$ when $\Gamma = \{u = v\}$.

In the following, we consider equational theories E that can be represented by a convergent rewrite system $\mathcal{R}$ that has the finite variant property.

III. Capabilities of the attacker

At a particular point in time, after some interaction with a protocol, an attacker may know a sequence of messages $t_1, \ldots, t_\ell \in \mathcal{T}(\Sigma, \mathcal{N})$. Such a sequence is organised into a *frame*

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

that is a substitution with *domain* $\mathrm{dom}(\varphi) = \{\mathsf{w}_1, \ldots, \mathsf{w}_\ell\}$, and *size* $|\varphi| = \ell$.

Definition 3: Let φ be a frame, and $t \in \mathcal{T}(\Sigma, \mathcal{N})$.

We say that t is *strongly deducible* from φ (using R), written $\varphi \vdash^{-}_{R} t$, when $R\varphi\downarrow = t\downarrow$ for some $R \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathrm{dom}(\varphi))$.

Similarly, we say that t is *weakly deducible* from φ (using R), written $\varphi \vdash^{+}_{R} t$, when $R\varphi\downarrow = t\downarrow$ for some $R \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathrm{dom}(\varphi))$.

Intuitively, an attacker is able to deduce new messages by applying function symbols in Σ to public names (those in $\mathcal{N}_{\mathsf{pub}}$) and terms he already knows (those in φ). The term R is called a *recipe*. Assuming that the attacker has access to names in $\mathcal{N}_{\mathsf{guess}}$, this gives him more power, and allows him to (weakly) deduce more terms.

Definition 4: Let φ be a frame, and $g \in \mathcal{N}_{\mathsf{guess}}$. We say that g is *guessable* from φ when there exist $R_1, R_2 \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathrm{dom}(\varphi))$ such that:

- $R_1\varphi =_{\mathsf{E}} R_2\varphi$, and
- $R'_1\varphi \neq_{\mathsf{E}} R'_2\varphi$ where $R'_i = R_i\{g \mapsto g'\}$ for some fresh name g' ($i \in \{1, 2\}$).

The idea underlying this definition is the following. The frame φ represents the information known by the attacker. To see if the attacker is able to infer the value of a guessable data, we check whether the attacker can distinguish a situation in which he has access to all the correct guesses from one where we provide him with a wrong value (here g' instead of g) for the weak data that he targeted. To distinguish the situation the attacker must apply a test (here R_1 and R_2) that holds when faced with the right value of the guess, but fails otherwise.

Example 4: Let $\varphi = \{\mathsf{w}_1 \mapsto \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle\}$ with $r_A \in \mathcal{N}_{\mathsf{guess}}$. The name r_A is guessable from φ. Indeed, considering the recipes $R_1 = \mathsf{proj}_2(\mathsf{w}_1)$ and $R_2 = \mathsf{hash}(\langle \mathsf{proj}_1(\mathsf{w}_1), r_A\rangle)$, we have that:

- $R_1\varphi =_{\mathsf{E}_{\mathsf{hash}}} \mathsf{hash}(\langle info_A, r_A\rangle) =_{\mathsf{E}_{\mathsf{hash}}} R_2\varphi$, whereas
- $(R_1\{r_A \mapsto r'_A\})\varphi \neq_{\mathsf{E}_{\mathsf{hash}}} R_2\{r_A \mapsto r'_A\}\varphi$.

However, we may note that r_A is not guessable from

$$\varphi' = \{\mathsf{w}_1 \mapsto \langle info_A, \mathsf{hash}(\langle info_A, \langle n_A, r_A\rangle\rangle)\rangle\}$$

assuming that $n_A \in \mathcal{N}_{\mathsf{prv}}$. Intuitively, without knowing n_A, there is no way for the attacker to check whether he has correctly guessed the value r_A.

This way of modelling guessable names allows one to take into account the fact that weak names are deducible by the attacker as soon as he has a way to verify that the guess he has performed is correct. This reflects the deduction capabilities of the attacker in a more accurate way than what happens in usual attacker models. However, this modelling comes with some limitations since it does not take into account the fact that a sufficiently large set of guessable names is as secure as a (strong) name. Indeed, in our setting, assuming that $r_1, \ldots, r_k$ are names from $\mathcal{N}_{\mathsf{guess}}$, it is possible to guess all these names from the message $\mathsf{hash}(\langle r_1, \langle r_2 \ldots, r_k\rangle\rangle)$, independently of k.

IV. Our cryptographic process calculus

We assume that cryptographic protocols are modelled using a simple process calculus which has some similarities with the applied pi calculus [2]. Following [11], we only consider a minimalistic core calculus, additionally extended with disequality tests and guess actions.

A. Syntax

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

P, Q	::=	$\mathbf{0}$	null process
		$\mathbf{in}(c, x).P$	input
		$\mathbf{out}(c, t).P$	output
		$[s = t].P$	positive test
		$[s \neq t].P$	negative test
		$\mathbf{guess}(g).P$	guess

where $x \in \mathcal{X}, s, t \in \mathcal{T}(\Sigma, \mathcal{N} \uplus \mathcal{X})$, $c \in \mathcal{C}h$, and $g \in \mathcal{N}_{\mathsf{guess}}$.

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

We may note that our calculus does not include operators for parallel composition, and we do not consider private channels. Given that we only consider a bounded number of sessions (*i.e.* a process calculus without replication) and that we aim at verifying a reachability based property, parallel composition, but also private channels, can be added as syntactic sugar to

denote the set of all interleavings. To ease the presentation, we also introduce a special action $\mathbf{guess}(g)$ in order to indicate at which step guessing on g has to be done. At the price of a (potentially exponential) blow-up on the number of processes needed to model a given protocol, we may indeed assume that this kind of actions is given explicitly.

Example 5: As a running example, we consider the following process where $c \in \mathcal{C}h$, $ack \in \mathcal{N}_{\mathsf{pub}}$, $r_A \in \mathcal{N}_{\mathsf{guess}}$, and $x, y \in \mathcal{X}$:

$$\begin{array}{rcl} P_{\mathsf{weak}} & = & \mathbf{out}(c, m_A).\mathbf{guess}(r_A).\mathbf{in}(c, y). \\ & & \mathbf{out}(c, r_A).[\mathsf{hash}(\langle \mathsf{proj}_1(y), r_A\rangle) = \mathsf{proj}_2(y)]. \\ & & [\mathsf{proj}_1(y) \neq info_A].\mathbf{0} \end{array}$$

where $m_A = \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle$.

Intuitively, this process models one possible interleaving (among many others) of the different actions of one session of the weak version of the protocol described in Section I. In order to keep this example simple, we discard the action $\mathbf{out}(c, ack)$ (output of a public name), and we have arbitrarily chosen to perform a guess action on r_A at the second step. Moreover, the purpose of the final test $[\mathsf{proj}_1(y) \neq info_A]$ is to see whether B can accept a message different from the expected one, i.e. $info_A$. Intuitively, an execution of this process until the end will correspond to an attack on the protocol. Of course, in order to perform a security analysis, given a protocol (as the one described in Section I), we have to explain how such interleavings are computed. This is briefly explained in Section VII-C where a richer calculus (with some convenient, additional operators) is introduced. Moreover, the way out-of-band channels are encoded is explained in Section VIII-B.However, the theoretical development explained in Sections V, VI, and VII, only rely on the syntax formally introduced in this section.

We may be interested to study a similar interleaving of actions for the strong version of the protocol (the one with $n_A \in \mathcal{N}_{\mathsf{prv}}$). This gives us:

$$\begin{array}{rcl} P_{\mathsf{strong}} & = & \mathbf{out}(c, m_A).\mathbf{guess}(r_A).\mathbf{in}(c, y). \\ & & \mathbf{out}(c, n_A).\mathbf{out}(c, r_A). \\ & & [\mathsf{hash}(\langle \mathsf{proj}_1(y), \langle n_A, r_A\rangle\rangle) = \mathsf{proj}_2(y)]. \\ & & [\mathsf{proj}_1(y) \neq info_A].\mathbf{0} \end{array}$$

B. Semantics

We can now define the semantics of our process calculus by means of a labelled transition relation on configurations. A *configuration* is a pair (P, φ) where P is a ground process, and φ is a frame used to record the messages that the participants have sent previously, and that are known by the attacker.

The relation $\xrightarrow{l}$ where l is either an input, an output, a test, or a guess is defined as follows:

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

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

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

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

$$\textsc{Guess} \quad (\mathbf{guess}(g).P, \varphi) \xrightarrow{\mathbf{guess}(g)} (P, \varphi \cup \{\mathsf{w}_{|\varphi|+1} \mapsto g\}) \quad \text{if } g \text{ is guessable from } \varphi$$

The label $\mathbf{in}(c, t)$ represents the input of a message t sent by the attacker over the channel c. The label $\mathbf{out}(c)$ indicates a message sent over channel c, and transition rule SEND records the message sent in the frame. The rule $\textsc{Test}^{=}$, respectively $\textsc{Test}^{\neq}$, checks equality, respectively difference, of s and t in the equational theory and is labelled by $\mathbf{test}^{=}$, respectively $\mathbf{test}^{\neq}$. The label $\mathbf{guess}(g)$ represents the fact that the value g has been guessed, and this is only possible when $g \in \mathcal{N}_{\mathsf{guess}}$ is indeed guessable. In such a case, g is added into the frame.

Example 6: Considering the process P_{weak} given in Example 5, we have that

$$\begin{array}{l} (P_{\mathsf{weak}}, \emptyset) \xrightarrow{\mathbf{out}(c)} \xrightarrow{\mathsf{guess}(r_A)} (P_2, \{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A\}) \\ \quad \xrightarrow{\mathbf{in}(c, \langle info_I, \mathsf{hash}(\langle info_I, \mathsf{w}_2\rangle)\rangle)} (P_3, \{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A\}) \\ \quad \xrightarrow{\mathbf{out}(c)} \xrightarrow{\mathbf{test}^{=}} \xrightarrow{\mathbf{test}^{\neq}} (\mathbf{0}, \{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A, \mathsf{w}_3 \mapsto r_A\}) \end{array}$$

where $info_I \in \mathcal{N}_{\mathsf{pub}}$. This execution corresponds to the attack described in the introduction. The fact that $\mathsf{guess}(r_A)$ can be fired is a direct consequence of the fact that r_A is guessable from φ (see Example 4). Note that this action would not be possible starting from $(P_{\mathsf{strong}}, \emptyset)$. This comes from the fact that r_A is not guessable from φ' (see Example 4).

In the rest of the paper, for our technical development, we also consider an alternative semantics : $\overset{l}{\twoheadrightarrow}$ is defined as $\xrightarrow{l}$, except that rules $\textsc{Test}^{\neq}$ and GUESS are replaced by the two following ones:

$$\begin{array}{rcl} (\mathbf{guess}(g).P, \varphi) & \xrightarrow{\mathbf{guess}(g)} & (P, \varphi \cup \{\mathsf{w}_{|\varphi|+1} \mapsto g\}) \\ ([s \neq t].P, \varphi) & \xrightarrow{\mathbf{test}^{\neq}} & (P, \varphi) \end{array}$$

i.e., this alternative semantics does neither check whether a name is indeed guessable, nor differences of terms.

The goal of this paper is to analyse confidentiality and authentication properties of protocols that rely on weak secrets. These properties can be directly encoded as reachability properties in a process. We therefore study the *Guess Reachability Problem* defined as follows.

Guess Reachability Problem

- *Input:* A ground process P
- *Output:* Do there exist $l_1, \ldots, l_n$, and a frame φ such that $(P, \emptyset) \xrightarrow{l_1, \ldots, l_n} (\emptyset, \varphi)$?

We emphasise that the execution is defined w.r.t. $\xrightarrow{l}$ and not $\overset{l}{\twoheadrightarrow}$. A difficulty is to ensure that g is guessable when a $\mathbf{guess}(g)$ action is fired and that disequalities hold when they occur. In case such a complete execution exists, we say that the process P is guess reachable.

In case, the ground process under study only contains one guess action at the end, this problem is similar to the problem of deciding whether a protocol is resistant against guessing attack. Here, the difficulty is to develop a procedure allowing us to consider several guesses. Note also that, contrary to the case of password-based protocols, guessing a data is not a goal in itself, and does not necessarily lead to an attack on a protocol. In many protocols, participants first commit to weak data and reveal it later in the execution.

V. Modelling using Horn clauses

As the original procedure described in [11] and implemented in the AKISS tool, our decision procedure is based on a fully abstract modelling of a process in first-order Horn clauses. We give the details of this modelling in this section.

A. Predicates

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

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

where l is of the form $\mathbf{in}(c,t)$, $\mathbf{out}(c)$, $\mathbf{test}^{=}$, $\mathbf{test}^{\neq}$, or $\mathbf{guess}(g)$ with $t \in \mathcal{T}(\Sigma, \mathcal{N} \uplus \mathcal{X})$, $c \in \mathcal{C}h$, and $g \in \mathcal{N}_{\mathsf{guess}}$.

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

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 $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathcal{W} \uplus \mathcal{Y})$.

We consider five kinds of predicates over which we construct the atomic formulas of our logic. Below, w denotes a symbolic run, R, R' are terms in $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathcal{W} \uplus \mathcal{Y})$, and t is a term in $\mathcal{T}(\Sigma, \mathcal{N} \uplus \mathcal{X})$. Informally, these predicates have the following meaning (see Figure 1 for the formal semantics).

- *reachability predicate*: r_w holds when the run w is executable in the alternative semantics $\overset{l}{\twoheadrightarrow}$, i.e., ignoring constraints on guesses and differences;
- *attacker knowledge predicate*: $\mathsf{k}^{-}_w(R,t)$ holds if whenever the run w is executable (w.r.t. $\twoheadrightarrow$), the message t can be constructed by the attacker using the recipe R that does not contain names in $\mathcal{N}_{\mathsf{guess}}$;
- *attacker extended knowledge predicate*: $\mathsf{k}^{+}_w(R,t)$ holds if whenever the run w is executable (w.r.t. $\twoheadrightarrow$), the message t can be constructed by the attacker using the recipe R that may contain names in $\mathcal{N}_{\mathsf{guess}}$;
- *identity predicate*: $\mathsf{i}_w(R,R')$ holds if whenever the run w is executable (w.r.t. $\twoheadrightarrow$), R and R' (that may contains names in $\mathcal{N}_{\mathsf{guess}}$) are recipes for the same term; and
- *reachable identity predicate*: $\mathsf{ri}_w(\overrightarrow{R}, \overrightarrow{R'})$ is a short form for the conjunction of $\mathsf{i}_w(R_1, R'_1), \ldots, \mathsf{i}_w(R_n, R'_n)$, and r_w, where $\overrightarrow{R} = (R_1, \ldots, R_n)$, and $\overrightarrow{R'} = (R'_1, \ldots, R'_n)$.

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.

Example 7: Let P_{weak} be the process defined in Example 5, and consider the following run:

$$w_0 = \mathbf{out}(c), \mathbf{guess}(r_A), \mathbf{in}(c, m_I), \mathbf{out}(c), \mathbf{test}^{=}, \mathbf{test}^{\neq}$$

with $m_I = \langle info_I, \mathsf{hash}(\langle info_I, r_A \rangle) \rangle$. We have $P_{\mathsf{weak}} \models \mathsf{r}_{w_0}$, and this is a direct consequence of the execution trace given in Example 6. We have that $P_{\mathsf{weak}} \models \mathsf{i}_{\mathbf{out}(c)}(R_1, R_2)$ where $R_1 = \mathsf{proj}_2(\mathsf{w}_1)$ and $R_2 = \mathsf{hash}(\langle \mathsf{proj}_1(\mathsf{w}_1), r_A \rangle)$. Indeed, the only frame reachable from P_{weak} through the run $\mathbf{out}(c)$ is φ as defined in Example 4. Let $t = \mathsf{hash}(\langle info_A, r_A \rangle)$, we have that $\varphi \vdash^{+}_{R_1} t$ (actually we have $\varphi \vdash^{-}_{R_1} t$) and $\varphi \vdash^{+}_{R_2} t$.

Let P_{strong} as given in Example 5, and w'_0 be as follows:

$$\mathbf{out}(c).\mathbf{guess}(r_A), \mathbf{in}(c, m'_I), \mathbf{out}(c), \mathbf{out}(c), \mathbf{test}^{=}, \mathbf{test}^{\neq}$$

with $m'_I = \langle info_I, \mathsf{hash}(\langle info_I, \langle n_I, r_A \rangle \rangle) \rangle$. We have also that $P_{\mathsf{strong}} \models \mathsf{r}_{w'_0}$ since such a trace is indeed executable w.r.t. the weak semantics $\twoheadrightarrow$.

B. Seed statements

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: A *statement* is a Horn clause of the form

$$H \Leftarrow \mathsf{k}^{\star_1}_{u_1}(X_1, t_1), \ldots, \mathsf{k}^{\star_n}_{u_n}(X_n, t_n)$$

where:

- $H \in \{\mathsf{r}_{u_0},\ \mathsf{k}^{-}_{u_0}(R,t),\ \mathsf{k}^{+}_{u_0}(R,t),\ \mathsf{i}_{u_0}(R,R'),\ \mathsf{ri}_{u_0}(\overrightarrow{R}, \overrightarrow{R'})\}$;
- $u_0, u_1, \ldots, u_n$ are symbolic runs such that $u_i \sqsubseteq u_0$ for any $i \in \{1, \ldots, n\}$;
- $t, t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{N} \uplus \mathcal{X})$;
- R, R' as well as elements of $\overrightarrow{R}$ (resp. $\overrightarrow{R'}$) are in $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathcal{W} \uplus \mathcal{Y})$;
- and $X_1, \ldots, X_n$ are distinct variables from $\mathcal{Y}$.

Lastly, we have that $\mathrm{vars}(t) \subseteq \mathrm{vars}(t_1, \ldots, t_n)$ when $H = \mathsf{k}^{\star}_{u_0}(R,t)$ with $\star \in \{-, +\}$.

In the definition above, we implicitly assume that all variables are universally quantified, i.e. all statements are ground.

$$
\begin{array}{ll}
(P_0,\varphi_0) \models \mathsf{r}_{l_1,\ldots,l_n} & \text{if } (P_0,\varphi_0) \xrightarrow{l_1}\!\!\!\!\rightarrow (P_1,\varphi_1) \xrightarrow{l_2}\!\!\!\!\rightarrow \ldots \xrightarrow{l_n}\!\!\!\!\rightarrow (P_n,\varphi_n) \\
(P_0,\varphi_0) \models \mathsf{k}^-_{l_1,\ldots,l_n}(R,t) & \text{if when } (P_0,\varphi_0) \xrightarrow{l_1}\!\!\!\!\rightarrow (P_1,\varphi_1) \xrightarrow{l_2}\!\!\!\!\rightarrow \ldots \xrightarrow{l_n}\!\!\!\!\rightarrow (P_n,\varphi_n) \text{ then } \varphi_n \vdash^-_R t \\
(P_0,\varphi_0) \models \mathsf{k}^+_{l_1,\ldots,l_n}(R,t) & \text{if when } (P_0,\varphi_0) \xrightarrow{l_1}\!\!\!\!\rightarrow (P_1,\varphi_1) \xrightarrow{l_2}\!\!\!\!\rightarrow \ldots \xrightarrow{l_n}\!\!\!\!\rightarrow (P_n,\varphi_n) \text{ then } \varphi_n \vdash^+_R t \\
(P_0,\varphi_0) \models \mathsf{i}_{l_1,\ldots,l_n}(R,R') & \text{if there exists } t \text{ such that } (P_0,\varphi_0) \models \mathsf{k}^+_{l_1,\ldots,l_n}(R,t) \text{ and } (P_0,\varphi_0) \models \mathsf{k}^+_{l_1,\ldots,l_n}(R',t) \\
(P_0,\varphi_0) \models \mathsf{ri}_{l_1,\ldots,l_n}(\overrightarrow{R},\overrightarrow{R'}) & \text{if } (P_0,\varphi_0) \models \mathsf{r}_{l_1,\ldots,l_n} \text{ and for all } 1 \le i \le k\ (P_0,\varphi_0) \models \mathsf{i}_{l_1,\ldots,l_n}(R_i,R'_i) \\
 & \qquad\qquad \text{where } \overrightarrow{R},\overrightarrow{R'} = (R_1,\ldots,R_k),(R'_1,\ldots,R'_k)
\end{array}
$$

Fig. 1. Semantics of atomic formulas

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 $H, B_1,\ldots,B_n$.

$$
\begin{array}{l}
\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow} \Leftarrow \{\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j,x_j\sigma\tau\downarrow)\}_{j\in R(m)} \\
\quad \text{for all } 0 \le m \le n \\
\quad \text{for all } \sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^{=}(m)}) \\
\quad \text{for all } \tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma,\ldots,\ell_m\sigma) \\
\\
\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, t_m\sigma\tau\downarrow) \Leftarrow \\
\qquad\qquad \{\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j,x_j\sigma\tau\downarrow)\}_{j\in R(m)} \\
\quad \text{for all } m \in S(n) \\
\quad \text{for all } \sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^{=}(m)}) \\
\quad \text{for all } \tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma,\ldots,\ell_m\sigma,t_m\sigma) \\
\\
\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, g_m) \Leftarrow \\
\qquad\qquad \{\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j,x_j\sigma\tau\downarrow)\}_{j\in R(m)} \\
\quad \text{for all } m \in G(n) \\
\quad \text{for all } \sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^{=}(m)}) \\
\quad \text{for all } \tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma,\ldots,\ell_m\sigma) \\
\\
\mathsf{k}^-_{\ell_1,\ldots,\ell_m}(\mathsf{f}(Y_1,\ldots,Y_k), \mathsf{f}(y_1,\ldots,y_k)\tau\downarrow) \Leftarrow \\
\qquad\qquad \{\mathsf{k}^-_{\ell_1,\ldots,\ell_m}(Y_j,y_j\tau\downarrow)\}_{j\in\{1,\ldots,k\}} \\
\quad \text{for all } 0 \le m \le n \\
\quad \text{for all function symbols } \mathsf{f} \text{ of arity } k \\
\quad \text{for all } \tau \in \mathrm{variants}_{\mathcal{R}}(\mathsf{f}(y_1,\ldots,y_k)). \\
\\
\mathsf{k}^+_{\ell_1,\ldots,\ell_m}(\mathsf{f}(Y_1,\ldots,Y_k), \mathsf{f}(y_1,\ldots,y_k)\tau\downarrow) \Leftarrow \\
\qquad\qquad \{\mathsf{k}^+_{\ell_1,\ldots,\ell_m}(Y_j,y_j\tau\downarrow)\}_{j\in\{1,\ldots,k\}} \\
\quad \text{for all } 0 \le m \le n \\
\quad \text{for all function symbols } \mathsf{f} \text{ of arity } k \\
\quad \text{for all } \tau \in \mathrm{variants}_{\mathcal{R}}(\mathsf{f}(y_1,\ldots,y_k)). \\
\\
\mathsf{k}^-_{\epsilon}(c,c) \Leftarrow \qquad \text{for all names } c \in \mathcal{N}^0_{\mathsf{pub}} \\
\mathsf{k}^+_{\epsilon}(g,g) \Leftarrow \qquad \text{for all names } g \in \mathcal{N}^0_{\mathsf{guess}}
\end{array}
$$

Fig. 2. Set $\mathsf{seed}(P,\mathcal{N}^0_{\mathsf{pub}},\mathcal{N}^0_{\mathsf{guess}})$ of seed statements

As mentioned above, our decision procedure is based on an abstract modelling of a process in first-order Horn clauses. In this section, given a ground process P we will give a set of statements $\mathsf{seed}(P)$ which will serve as a starting point for the modelling. We shall also establish that the set of statements $\mathsf{seed}(P)$ is a sound and (partially) complete abstraction of the ground process P. In order to formally define $\mathsf{seed}(P)$, we start by fixing some conventions.

Let $P = \mathsf{a}_1.\mathsf{a}_2.\ldots.\mathsf{a}_n$ be a ground process. We assume w.l.o.g. the following naming conventions:

1) if a_i is a receive action then $\mathsf{a}_i = \mathbf{in}(c_i,x_i)$.
2) if a_i is a send action then $\mathsf{a}_i = \mathbf{out}(c_i,t_i)$.
3) if a_i is a guess action then $\mathsf{a}_i = \mathbf{guess}(g_i)$.
4) if a_i is a positive test action then $\mathsf{a}_i = [s_i = t_i]$.
5) if a_i is a negative test action then $\mathsf{a}_i = [s_i \neq t_i]$.

Moreover, we assume that $x_i \neq x_j$ for any $i \neq j$. For each $1 \le i \le n$, we define the symbolic label ℓ_i as follows:

$$
\ell_i = \begin{cases}
\mathbf{in}(c_i,x_i) & \text{if } \mathsf{a}_i = \mathbf{in}(c_i,x_i) \\
\mathbf{out}(c_i) & \text{if } \mathsf{a}_i = \mathbf{out}(c_i,t_i) \\
\mathbf{guess}(g_i) & \text{if } \mathsf{a}_i = \mathbf{guess}(g_i) \\
\mathbf{test}^{=} & \text{if } \mathsf{a}_i = [s_i = t_i] \\
\mathbf{test}^{\neq} & \text{if } \mathsf{a}_i = [s_i \neq t_i]
\end{cases}
$$

For each $0 \le m \le n$, let $R(m)$ (resp. $S(m)$, $G(m)$, $T^{=}(m)$, and $T^{\neq}(m)$) be the set of indices of the receive (resp. send, guess, positive and negative test) actions amongst $\mathsf{a}_1,\ldots,\mathsf{a}_m$. Moreover, we denote by $|\Gamma|$ the number of elements in such a set Γ. Formally,

- $R(m) = \{\ i\ \mid\ 1 \le i \le m \text{ and } \mathsf{a}_i = \mathbf{in}(c_i,x_i)\ \}$;
- $S(m) = \{\ i\ \mid\ 1 \le i \le m \text{ and } \mathsf{a}_i = \mathbf{out}(c_i,t_i)\ \}$;
- $G(m) = \{\ i\ \mid\ 1 \le i \le m \text{ and } \mathsf{a}_i = \mathbf{guess}(g_i)\ \}$;
- $T^{=}(m) = \{\ i\ \mid\ 1 \le i \le m \text{ and } \mathsf{a}_i = [s_i = t_i]\ \}$;
- $T^{\neq}(m) = \{\ i\ \mid\ 1 \le i \le m \text{ and } \mathsf{a}_i = [s_i \neq t_i]\ \}$.

Given a set of public names $\mathcal{N}^0_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$, and a set of guessable names $\mathcal{N}^0_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$, the *set of seed statements* associated to P, $\mathcal{N}^0_{\mathsf{pub}}$, and $\mathcal{N}^0_{\mathsf{guess}}$ denoted $\mathsf{seed}(P,\mathcal{N}^0_{\mathsf{pub}},\mathcal{N}^0_{\mathsf{guess}})$, is defined to be the set of statements given in Figure 2. Intuitively, the first three kinds of seed statements allows one to express that as soon as suitable messages are deducible by the attacker to fill the receive actions occurring before step m, the process is indeed (weakly) executable until step m. Moreover, in case its m^{th} action is either a send action or a guess action, the corresponding term (or guess) will become (strongly) deducible by the attacker.

If $(\mathcal{N}^0_{\mathsf{pub}}, \mathcal{N}^0_{\mathsf{guess}}) = (\mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{guess}})$, then we write $\mathsf{seed}(P)$ as a shortcut for $\mathsf{seed}(P, \mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{guess}})$. This set is is said to be the set of seed statements associated to P.

Example 8: Continuing our running example, among others, we have the following statements in $\mathsf{seed}(P_{\mathsf{weak}})$:

$$
\begin{array}{ll}
f_1: & \mathsf{k}^-_{\mathbf{out}(c)}(\mathsf{w}_1, m_A) \Leftarrow \\
f_2: & \mathsf{k}^-_{\mathbf{out}(c).\mathbf{guess}(r_A)}(\mathsf{w}_2, r_A) \Leftarrow \\
f_3: & \mathsf{r}_w \Leftarrow \mathsf{k}^-_{\mathbf{out}(c).\mathbf{guess}(r_A)}(Y, \langle y_1, \mathsf{hash}(\langle y_1, r_A\rangle)\rangle) \\
f^+_4: & \mathsf{k}^+_{\mathbf{out}(c)}(\mathsf{hash}(X), \mathsf{hash}(x)) \Leftarrow \mathsf{k}^+_{\mathbf{out}(c)}(X, x) \\
f^-_5: & \mathsf{k}^-_{\mathbf{out}(c)}(\mathsf{proj}_2(Y), \mathsf{proj}_2(y)) \Leftarrow \mathsf{k}^-_{\mathbf{out}(c)}(Y, y) \\
f^-_6: & \mathsf{k}^-_{\mathbf{out}(c)}(\mathsf{proj}_2(Y), y_2) \Leftarrow \mathsf{k}^-_{\mathbf{out}(c)}(Y, \langle y_1, y_2\rangle)
\end{array}
$$

where $w = \mathbf{out}(c), \mathbf{guess}(r_A), \mathbf{in}(c, \langle y_1, \mathsf{hash}(\langle y_1, r_A\rangle)\rangle), \mathbf{out}(c), \mathbf{test}^=, \mathbf{test}^{\neq}$.

Intuitively, f_1 represents that m_A is (strongly) deducible through recipe w_1 once the first ouput has been performed, whereas f_2 represents that r_A is strongly deducible once the two first actions $\mathbf{out}(c)$ and $\mathbf{guess}(r_A)$ have been performed. The statement f_3 says that the process P_{weak} is (weakly) executable until its last action if a message of the form $\langle m, \mathsf{hash}(\langle m, r_A\rangle)\rangle$ (for some m) is deducible from the information available to the attacker after executing the two first actions, i.e. knowing the first output m_A and the value r_A.

Some statements, e.g., f^+_4, f^-_5 and f^-_6, represent the capabilities of the attacker. The two last ones (f^-_5 and f^-_6) are computed relying on $\mathrm{variants}_{\mathcal{R}}(\mathsf{proj}_2(y)) = \{\sigma_1, \sigma_2\}$ (see Example 3).

C. Soundness and completeness

We shortly show that the set of seed statements is a sound and (partially) complete modelling of a process. However, we need one more definition to state this fact.

Definition 6: Given a set K of statements, $\mathcal{H}(K)$ is the smallest set of ground facts such that:

$$\textsc{Conseq}\ \frac{\sigma \text{ grounding for } f = \Big(H \Leftarrow B_1, \ldots, B_n\Big) \in K \qquad B_1\sigma \in \mathcal{H}(K),\ \ldots,\ B_n\sigma \in \mathcal{H}(K)}{H\sigma \in \mathcal{H}(K)}$$

$$\textsc{Extend}\ \frac{\mathsf{k}^\star_u(R,t) \in \mathcal{H}(K) \text{ wih } \star \in \{-,+\}}{\mathsf{k}^\star_{uv}(R,t) \in \mathcal{H}(K)}$$

$$\textsc{Weakening}\ \frac{\mathsf{k}^-_u(R,t) \in \mathcal{H}(K)}{\mathsf{k}^+_u(R,t) \in \mathcal{H}(K)}$$

We show that as far as reachability predicates and attacker knowledge predicates are concerned, the set $\mathsf{seed}(P)$ is a complete abstraction of a process (w.r.t. $\twoheadrightarrow$).

Theorem 1: Let P be a ground process.

- Soundness: $P \models f$ for any $f \in \mathsf{seed}(P) \cup \mathcal{H}(\mathsf{seed}(P))$;
- Completeness: If $(P, \emptyset) \xrightarrow{l_1,\ldots,l_m} (Q, \varphi)$ then
  1) $\mathsf{r}_{l_1\downarrow,\ldots,l_m\downarrow} \in \mathcal{H}(\mathsf{seed}(P))$;
  2) if $\varphi \vdash^\star_R t$ with $\star \in \{-,+\}$ then $\mathsf{k}^\star_{l_1\downarrow,\ldots,l_m\downarrow}(R, t\downarrow) \in \mathcal{H}(\mathsf{seed}(P))$.

We will show how the completeness of $\mathsf{seed}(P)$ can be built upon to achieve full abstraction, i.e., including also identities of the process P and a procedure for checking guess reachability.

VI. Saturation

A. Saturation procedure

In this section we will describe the saturation procedure. It manipulates a set of statements called a *knowledge base*.

Definition 7: Given a statement $f = (H \Leftarrow B_1, \ldots, B_n)$,

- f is said to be *solved* if for all $1 \leq i \leq n$, we have that $B_i = \mathsf{k}^{\star_i}_{w_i}(X_i, x_i)$ for some $\star_i \in \{-,+\}$, $x_i \in \mathcal{X}$, and $X_i \in \mathcal{Y}$.
- f is said to be *well-formed* if whenever it is solved and $H = \mathsf{k}^\star_w(R,t)$ for some $\star \in \{-,+\}$, we have that $t \notin \mathcal{X}$.

A set of *well-formed* statements is called a *knowledge base*. If K is a knowledge base, $K_{\mathsf{solved}} = \{f \in K \mid f \text{ is solved }\}$.

The saturation procedure is a non-deterministic process whose purpose is to produce a knowledge base starting from an initial one. Roughly, the saturation procedure works as follows. Each time a new statement is *generated*, the knowledge base is *updated* with it. This process continues until reaching a fixed point. At the end, we obtain a knowledge base $K_{\mathsf{sat}} \in \mathsf{sat}(K_{\mathsf{init}})$. The set of reachable fixed points starting from the initial set K_{init} is denoted $\mathsf{sat}(K_{\mathsf{init}})$.

a) Generating new statements: Given a knowledge base K, new statements are generated by applying the rules in Figure 3. Each rule generates a new statement h. Roughly, the rule Resolution applies the standard rule of resolution from first-order logic between an unsolved statement f and a solved deduction statement g. We do not require $\star' = \star$ but only $\star' \leq \star$ which is formally defined as follows:

$$\star' \leq \star \text{ if, and only if, } \begin{cases} \text{either } \star' = \star; \\ \text{or } (\star', \star) = (-,+). \end{cases}$$

This condition reflects the intuition that any term that is strongly deducible, is also weakly deducible. The rule Equation allows us to derive new identities on recipes that may be imposed by the execution of the protocol. The rule Test allows us to derive a sequence of identities (at most one per guess action) that necessarily hold in an execution of the protocol. Once the statement h is generated, we update the knowledge base K with h as explained below.

b) Update: We will now define the update operator $\Cup$ which adds statements generated by the rules of Figure 3 to the knowledge base.

Definition 8: Let K be a knowledge base, and $f = H \Leftarrow \mathcal{B}$ a statement. The *update of K by f*, denoted $K \Cup f$, is $K \cup \{f\}$ when f is either not a solved deduction statement, or a well-formed and solved deduction statement. Otherwise, we have that $f = \mathsf{k}^\star_u(R,x) \Leftarrow \mathcal{B}$ with $\mathsf{k}^{\star'}_{u'}(X,x) \in \mathcal{B}$ for some $\star'$, u',

$$\textsc{Resolution}\ \frac{f = \Big(H \Leftarrow \mathsf{k}^{\star}_{uv}(X,t), B_1, \ldots, B_n\Big) \qquad g = \Big(\mathsf{k}^{\star'}_{w}(R,t') \Leftarrow B_{n+1}, \ldots, B_m\Big)}{f \in K, g \in K_{\mathsf{solved}} \qquad \sigma = \mathrm{mgu}(\mathsf{k}_u(X,t), \mathsf{k}_w(R,t')) \qquad t \notin \mathcal{X} \qquad \star, \star' \in \{-,+\} \qquad \star' \leq \star}$$
$$K = K \Cup h \text{ where } h = \Big(H \Leftarrow B_1, \cdots, B_m\Big)\sigma$$

$$\textsc{Equation}\ \frac{f = \Big(\mathsf{k}^{\star}_{u}(R,t) \Leftarrow B_1, \ldots, B_n\Big) \quad g = \Big(\mathsf{k}^{\star'}_{u'v'}(R',t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \quad f, g \in K_{\mathsf{solved}} \quad \sigma = \mathrm{mgu}(\langle u,t\rangle, \langle u',t'\rangle) \quad \star, \star' \in \{-,+\}}{K = K \Cup h \text{ where } h = \Big(\mathsf{i}_{u'v'}(R,R') \Leftarrow B_1, \ldots, B_m\Big)\sigma}$$

$$\textsc{Test}\ \frac{g = \mathsf{r}_w \Leftarrow \mathcal{B} \quad f_1 = \Big(\mathsf{i}_{w_1}(R_1, R'_1) \Leftarrow \mathcal{B}_1\Big) \quad \cdots \quad f_n = \Big(\mathsf{i}_{w_n}(R_n, R'_n) \Leftarrow \mathcal{B}_n\Big) \quad g, f_1, \ldots, f_n \in K_{\mathsf{solved}} \quad \sigma = \mathrm{mgu}(\{w_i x_i, = w\}_{1 \leq i \leq n}) \quad n = \#\mathsf{guess}(w), \text{ and } \#\mathsf{guess}(w_i) < i \text{ for } 1 \leq i \leq n}{K = K \Cup h \text{ where } h = \Big(\mathsf{ri}_w((R_1, \ldots, R_n), (R'_1, \ldots, R'_n)) \Leftarrow \mathcal{B}, \mathcal{B}_1, \ldots, \mathcal{B}_n\Big)\sigma}$$

Fig. 3. Saturation rules

and X (by definition of a statement). In such a case, we have that:

$$K \Cup f = K \cup \mathsf{i}_u(R, X) \Leftarrow \mathcal{B}$$

c) Initial knowledge base: We finally define on which knowledge base we initiate the saturation procedure.

Definition 9: Given a set of statements S, the *initial knowledge base* associated to S, denoted $K_{\mathsf{init}}(S)$, is defined to be the empty knowledge base updated by the set S, i.e.,

$$K_{\mathsf{init}}(S) = (((\emptyset \Cup f_1) \Cup \ldots) \Cup f_n)$$

where $f_1, \ldots, f_n$ is an enumeration of the statements in S.

Lemma 1: Given a ground process P, $\mathcal{N}^0_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$, and $\mathcal{N}^0_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$, the set $K_{\mathsf{init}}(\mathsf{seed}(P, \mathcal{N}^0_{\mathsf{pub}}, \mathcal{N}^0_{\mathsf{guess}}))$ is a knowledge base. Moreover, given a knowledge base K, any set of statements obtained through the saturation procedure is also a knowledge base.

Example 9: We consider the seed statements introduced in Example 8. An application of the resolution rule between f_6^- and the solved statement f_1 leads to the following solved statement

$$h = \mathsf{k}^-_{\mathbf{out}(c)}(\mathsf{proj}_2(\mathsf{w}_1), \mathsf{hash}(\langle info_A, r_A\rangle)) \Leftarrow$$

which is simply added to the knowledge base by the update operator. We may then decide to apply the equation rule between h and f_4^+ which leads to the statement:

$$\mathsf{i}_{\mathbf{out}(c)}(\mathsf{proj}_2(\mathsf{w}_1), \mathsf{hash}(X)) \Leftarrow \mathsf{k}^+_{\mathbf{out}(c)}(X, \langle info_A, r_A\rangle)$$

and then, after several resolution steps, we are able to derive the solved statement

$$\mathsf{i}_{\mathbf{out}(c)}(\mathsf{proj}_2(\mathsf{w}_1), \mathsf{hash}(\langle info_A, r_A\rangle) \Leftarrow$$

Similarly, we are able to derive $\mathsf{r}_w \Leftarrow \mathsf{k}^-_{\mathbf{out}(c),\mathbf{guess}(r_A)}(Y_1, y_1)$ by performing several resolution steps starting from f_3. Therefore, we may apply the TEST rule to derive f_{final}:

$$\mathsf{ri}_w(\mathsf{proj}_2(\mathsf{w}_1), \mathsf{hash}(\langle info_A, r_A\rangle) \Leftarrow \mathsf{k}^-_{\mathbf{out}(c),\mathbf{guess}(r_A)}(Y_1, y_1)$$

Intuitively, this statement means that the symbolic run w is indeed executable and leads to a test $\mathsf{proj}_2(\mathsf{w}_1) = \mathsf{hash}(\langle info_A, r_A\rangle)$ that holds in the resulting frame (provided the attacker knows a public value that will be used to instantiate y_1).

B. Soundness and completeness

We extend $\mathcal{H}(K)$ to establish that any $K_{\mathsf{sat}} \in \mathsf{sat}(K_{\mathsf{init}}(P))$ is a complete abstraction of P.

Definition 10: Let K be a set of statements. 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.

We have that the set of solved statements produced by the saturation procedure is a sound and complete abstraction of the ground process P (w.r.t. $\twoheadrightarrow$). More formally, we have the following result.

Theorem 2: Let $K \in \mathsf{sat}(K_{\mathsf{init}}(P))$ for some ground process P. We have that:

- Soundness: $P \models f$ for any $f \in K \cup \mathcal{H}_{\mathsf{e}}(K)$;
- Completeness: If $(P, \emptyset) \xrightarrow{l_1, \ldots, l_n} (Q, \varphi)$ then
  1) $\mathsf{r}_{l_1\downarrow, \ldots, l_n\downarrow} \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$;
  2) if $\varphi \vdash^+_R t$ then $\mathsf{k}^+_{l_1\downarrow, \ldots, l_n\downarrow}(R, t\downarrow) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$;
  3) if $\varphi \vdash^+_R t$ and $\varphi \vdash^+_{R'} t$, then $\mathsf{i}_{l_1\downarrow, \ldots, l_n\downarrow}(R, R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$.

VII. Our procedure

In this section, we first discuss the effectiveness and termination of the saturation procedure. Then, we describe our algorithm to solve the Guess Reachability Problem as stated at the end of Section IV. We finally discuss the integration of our algorithm in the verification tool AKISS.

A. Effectiveness of the saturation procedure

In this section, we will discuss some issues on how to effectively compute a saturated knowledge base and also discuss termination of the saturation procedure.

$$\textsc{Refl}\ \frac{}{\mathsf{i}_w(R,R)\in\mathcal{H}_\mathsf{e}(K)} \qquad \textsc{Tran}\ \frac{\mathsf{i}_w(R_1,R_2)\in\mathcal{H}_\mathsf{e}(K)\quad \mathsf{i}_w(R_2,R_3)\in\mathcal{H}_\mathsf{e}(K)}{\mathsf{i}_w(R_1,R_3)\in\mathcal{H}_\mathsf{e}(K)}$$

$$\textsc{Sym}\ \frac{\mathsf{i}_w(R_1,R_2)\in\mathcal{H}_\mathsf{e}(K)}{\mathsf{i}_w(R_2,R_1)\in\mathcal{H}_\mathsf{e}(K)} \qquad \textsc{Cong}\ \frac{\mathsf{i}_w(R_1,R_1'),\ldots,\mathsf{i}_w(R_n,R_n')\in\mathcal{H}_\mathsf{e}(K)\ \ \mathsf{f}\in\Sigma}{\mathsf{i}_w(\mathsf{f}(R_1,\ldots R_n),\mathsf{f}(R_1',\ldots R_n'))\in\mathcal{H}_\mathsf{e}(K)}$$

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

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

A first issue comes from the fact that $K_{\mathsf{init}}(P)$ is infinite. Indeed, the set $\mathsf{seed}(P)$ for a ground process P is infinite because $\mathcal{N}_{\mathsf{pub}}$ and $\mathcal{N}_{\mathsf{guess}}$ contain an infinite set of names. We follow [11] to overcome this difficulty. Intuitively, we do only need to consider names occurring in P for its saturation; clauses representing names not occurring in P do not influence the saturation of the remaining clauses. Formally, we have the following result.

Lemma 2: Let P be a ground process, $\mathcal{N}^P_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$ (resp. $\mathcal{N}^P_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$) be the finite set of public names (resp. guessable names) occurring in P, and $K^0 \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(P,\mathcal{N}^P_{\mathsf{pub}},\mathcal{N}^P_{\mathsf{guess}})))$. We have that $K \subseteq K^0 \cup \mathcal{S}$ for some $K \in \mathsf{sat}(K_{\mathsf{init}}(P))$ and a set $\mathcal{S}$ containing statements of the form:

- $\mathsf{k}^-_\epsilon(n,n) \Leftarrow$ for any $n \in \mathcal{N}_{\mathsf{pub}}$;
- $\mathsf{k}^+_\epsilon(g,g) \Leftarrow$ for any $g \in \mathcal{N}_{\mathsf{guess}}$;
- $\mathsf{i}_\epsilon(n,n) \Leftarrow$ for any $n \in \mathcal{N}_{\mathsf{pub}} \cup \mathcal{N}_{\mathsf{guess}}$;
- $\mathsf{ri}_u((R_1,\ldots,R_k),(R_1',\ldots,R_k')) \Leftarrow \mathcal{B}$ where $(R_{i_0},R_{i_0}') = (n,n)$ for some $1 \le i_0 \le k$, and $n \in \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}}$.

The saturation procedure may itself not terminate even if the initial knowledge base is finite. As shown in [11] in a simple setting, we conjecture that our saturation procedure terminates for the class of subterm convergent equational theories. More importantly, we have implemented our procedure and tested it on several examples considering various equational theories that are not subterm convergent, and our tool did terminate on all practical examples we have tested.

B. Description and correctness of the algorithm

Our procedure is described in Figure 5. Let P be a ground process and $K^0 \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(P,\mathcal{N}^P_{\mathsf{pub}},\mathcal{N}^P_{\mathsf{guess}})))$ where $\mathcal{N}^P_{\mathsf{pub}}$ (resp. $\mathcal{N}^P_{\mathsf{guess}}$) is the set containing all public names (resp. guessable names) occurring in P. In our procedure, P is represented by the set K^0_{solved} of solved statements of K^0.

The test GUESS-REACHABILITY(P) checks whether there exists a reachable identity $\mathsf{ri}_{\ell_1,\ldots,\ell_n}(\overrightarrow{R},\overrightarrow{R'}) \Leftarrow \mathcal{B}$ in K^0_{solved} that corresponds to a real execution after replacement of the remaining variables with fresh constants. For guess actions, we simply check whether the recipes $R_j\omega$ and $R_j'\omega$ are witnesses of the fact that the j^{th} guess action contains indeed a guessable name. If all these tests succeed then P is indeed guess reachable. Otherwise, in case no such a reachable identity exists, then process P is *not* guess reachable.

Theorem 3: Let P be a ground process, $\mathcal{N}^P_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$ (resp. $\mathcal{N}^P_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$) be the finite set of public names (resp. guessable names) occurring in P. Let $K^0 \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(P,\mathcal{N}^P_{\mathsf{pub}},\mathcal{N}^P_{\mathsf{guess}})))$.

We have that P is guess reachable if, and only if GUESS-REACHABILITY(P, K^0_{solved}) holds.

The soundness part of this theorem is an easy consequence of our algorithm. Completeness is more involved. It is based on Theorem 2 but some works remains to be done to ensure that disequality tests will be satisfied. More importantly, we have to ensure that enough reachable identity predicates will be generated and tested. This is needed to ensure that when a weak name is guessable, then our algorithm will find a test witnessing this fact. Intuitively, our procedure will only generate small tests and our completeness results shows that this is indeed sufficient when we are looking for an attack. Note that, we have to consider all the tests (one per guess) together, and this may require to perform instantiations that will not be needed when considering only one guess. Intuitively, this comes from the fact that the attacker may need to output messages that are quite complex to ensure that all the guessable data will become guessable when needed (on the same trace).

C. Integration in AKISS

We have implemented all of the before described theory in the AKISS tool. To ease the specification of the protocols, the AKISS tool supports additional operators in the process calculus for parallel composition ($P||Q$), sequential composition ($P :: Q$) [11], and we also allow the use of (statically defined) private channels. All these constructions are syntactic sugar and can be translated to sets of (linear) processes eventually at the cost of an exponential blow-up.

While the original version of AKISS only allowed the verification of equivalence properties we added support for reachability properties (specified by the means of *events*). Similarly we allow to define authentication properties as (non-injective) correspondance properties. These correspondance properties are specified by annotating the protocols with *begin* and *end* events and are internally translated into simple reachability properties. Note that such a translation requires the use of disequality tests.

The tool also automatically annotates the protocol with the necessary *guess* actions, i.e., the protocol specifier is

GUESS-REACHABILITY(P, K^0_{solved})
Let n be the number of actions occurring in P.
For all $\mathsf{ri}_{\ell_1,\ldots,\ell_n}(\overrightarrow{R},\overrightarrow{R'}) \Leftarrow \mathsf{k}^{\star_1}_{w_1}(X_1,x_1),\ldots,\mathsf{k}^{\star_m}_{w_m}(X_m,x_m) \in K^0_{\mathsf{solved}}$
  Let $c_1,\ldots,c_k$ be fresh names in $\mathcal{N}_{\mathsf{pub}}$ such that $\sigma : \mathrm{vars}(\ell_1,\ldots,\ell_n) \cup \{x_1,\ldots,x_m\} \to \{c_1,\ldots,c_k\}$ is a bijection.
  Let $\omega = \{X_i \mapsto x_i\sigma \mid 1 \le i \le m\}$.
  Check whether $(P,\emptyset) = (P_0,\varphi_0) \xrightarrow{\ell_1\sigma} \ldots (P_{n-1},\varphi_{n-1}) \xrightarrow{\ell_n\sigma} (P_n,\varphi_n) = (\emptyset,\varphi)$ using the recipes $R_j\omega$ and $R'_j\omega$ as a witness for guessing the guessable name in the j^{th} guess action. If this execution is valid, return **Yes**.
Return **No**.

Fig. 5. Test for checking guess reachability

not required to include these actions. Intuitively, these guess actions have to be inserted at all possible steps. To limit the number of interleavings we implement a few straightforward optimisations: guess actions are only inserted before inputs when an output occurred since the last guess of the given weak name; when generating the interleavings we always favour outputs as they increase the attacker's knowledge which is a sound optimisation when verifying reachability properties.

VIII. CASE STUDIES

In this section we demonstrate that our tool can be effectively used to analyse protocols based on weak secrets. Weak secrets may either be low-entropy secrets (the guessable names introduced in Section II) or the result of applying a function that produces a short output, e.g., truncating the output of a hash function. The ISO/IEC 9798-6:2010 standard [23] suggests that "short" corresponds to 16-20 bits. We will first describe how we can model such *weak* hash functions. Weak secrets are often used in protocols relying on a (low capacity) out-of-band channel. We therefore also explain how out-of-band channels can be encoded in our formalism.

A. Weak functions

Sometimes short verification codes need to be transmitted. One way for generating these codes is to use *weak* hash functions whose output is short enough to be, e.g., copied or compared by a human. However, when the output of the function is short, an attacker may efficiently find *collisions*. As AKISS supports user specified equational theories, we encode this property in the equational theory. While, strong cryptographic hash functions are modelled as free symbols, the property that collisions may be found by the attacker on a keyed hash function is reflected using equations:

$$\begin{aligned} \mathsf{sh}(k',\mathsf{bf}(k',\mathsf{sh}(k,m))) &= \mathsf{sh}(k,m) \\ \mathsf{sh}(\mathsf{bf}(m',\mathsf{sh}(k,m)),m') &= \mathsf{sh}(k,m) \end{aligned}$$

The first equation models that given a short hash $\mathsf{sh}(k,m)$ and an arbitrary key k', the attacker may find by *brute force* a message $m' = \mathsf{bf}(k',\mathsf{sh}(k,m))$ such that $\mathsf{sh}(k,m) = \mathsf{sh}(k',m')$. The second equation allows to fix the message m' and find a key $k' = \mathsf{bf}(m',\mathsf{sh}(k,m))$ such that $\mathsf{sh}(m,k) = \mathsf{sh}(m',k')$. A similar modelling was also used by Chothia *et al.* in [12] for verifying commitment protocols in ProVerif.

The ISO/IEC 9798-6:2010 standard [23] distinguishes two types of short hash functions, which they call *check value functions* and *message digests*. While both generate short hashes, they differ on the keys they take as input: check value functions take as second argument a weak key and message digests a strong key. Our model does reflect these two types of hash functions, simply depending on whether the second argument is a weak name or not. The requested security property for short hash functions is that an attacker may not find two messages m, m' such that $\mathsf{sh}(m,k) = \mathsf{sh}(m',k)$ for a non-negligible fraction of the key space. This property is indeed reflected by our modelling as the equation $\mathsf{sh}(m,x) = \mathsf{sh}(m',x)$ does not hold for arbitrary keys, unless $m = m'$.

B. Out-of-band channels

When out-of-band channels are used in protocols we may consider different kinds of channels, depending on whether the communication is supposed to be confidential, authentic, and synchronous. To model these different kind of channels, we allow the use of *private channels*, similar to the applied pi calculus. However, we only allow static channel names, i.e., channel names are not transmitted dynamically during the protocol execution. In that case, private channels are merely syntactic sugar for generating the appropriate traces.

We consider the following types of channels:

- Public channels: they offer no guarantee and correspond to normal output actions in our calculus.
- Private channels: communications are confidential, authentic and synchronous; this corresponds to a usual communication using a private channel.
- Asynchronous, private channels: such communications offer confidentiality and authentication, but may be delayed by the attacker; asynchrony is encoded as usual by performing the output in parallel, i.e., if c_{ap} is an asynchronous private channel, we replace $\mathbf{out}(c_{ap},u).P$ with $P \mid \mathbf{out}(c_{\mathsf{priv}},u).0$ where c_{priv} is a private channel.
- Asynchronous, authentic, but non-confidential (or simply authentic) channels can be encoded similarly; if c_{aa} is an asynchronous authentic channel, we replace $\mathbf{out}(c_{aa},u).P$ with $\mathbf{out}(c_{\mathsf{pub}},u).P \mid \mathbf{out}(c_{\mathsf{priv}},u).0$ where c_{pub} is a public channel and c_{priv} is a private channel.

Which channel is the most appropriate depends on the context and the nature of the out-of-band channel.

C. Protocols with manual data transfer

The ISO/IEC 9798-6:2010 standard [23] presents several protocols for transferring authenticated data among two devices. The mechanisms differ in the assumptions on the input/output capabilities of the devices, the used of hash functions and the actions a user has to perform.

Devices may have a standard or simple input or output interface. While standard interfaces allow to enter/display alphanumeric strings, a simple interface generally only consists in a button or a light to confirm or infirm success. Hash functions may be short hash functions, cryptographic hash functions or MACs. Humans may need to either compare or copy short strings. Some protocols also require the human to generate a short random string.

To illustrate our work, we will detail below mechanisms 2 and 4 of [23].

a) Manual authentication mechanism 2: The protocol assumes two devices that have both a standard output interface and a simple input interface. The protocol uses a check-value function, i.e., a short hash function which uses a weak key. The protocol assumes that both devices are in proximity of a human who is able to compare short strings displayed on both device's output interface.

The protocol is described in Figure 6. Initially, data d_A and d_B are provided to devices A, respectively B. The goal of the protocol is to check whether the devices agree on the data. We assume that the data can be chosen freely by the attacker. Once they have received their data the devices signal to the user that they are ready. When both devices are ready, the user informs device A that it may start, e.g., by pressing the button of the simple input interface. Device A generates a fresh, weak key k and transmits it (on an insecure channel) to device B. Both devices display the key k and the short hash of their data with k on their output interface. The user compares the output of both devices and enters either accept or reject depending on whether the displayed codes coincide or not.

The protocol is easily encoded in the process calculus serving as the input language of AKISS. The process modelling the authentication mechanism is entirely displayed in Figure 7 where k is a weak name, and channels ah_1, ah_2, ha_1, ha_2, bh_1, bh_2, and hb are private channels used for encoding authentic channels as explained in Section VIII-B. We briefly explain below the process B given in Figure 7. Initially, d_B is provided to device B. Therefore, B initially waits for an input, and answers by outputting the constant ready on an authentic (but non-confidential) channel. To model these channel properties we rely on the encoding explained in Section VIII-B. Then, device B is waiting for another message on a classical (i.e. insecure) channel. B answers to this message by sending the key it just received as well as the short hash it has computed. Both are sent on another authentic channel (encoded in a similar way than before). Once this is done, it simply waits for an answer from the user (this is modelled through a communication on the private channel hb).

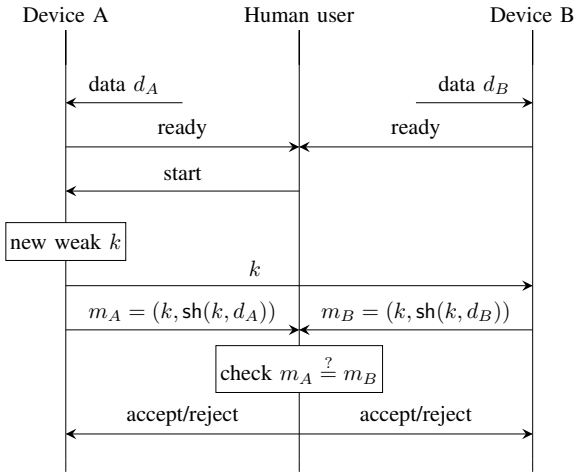

Fig. 6. Manual authentication mechanism 2 from [23]

In case, the received value is the public constant ok (which indicated that the user has accepted this transaction) device B will end this session normally (modelled through the event end). If the event end is reached authentication should hold. The two other processes A and H are modelled in a rather similar way. The operator :: that occurs in process H is used to model the two synchronisations that appear from the user's point of view. Indeed, the user is supposed to receive the ready message coming from both devices before he proceeds. Similarly, the user is supposed to wait for the short strings of both devices and compare them before he accepts/rejects.

scenario		# traces	time	status
one session	Auth. A	368	$<$ 8s	ok
	Auth. B	168	$<$ 4s	ok
without start	Auth. A	360/592	$<$ 8s	attack
	Auth. B	337/736	$<$ 8s	attack
without start + h strong	Auth. A	592	$<$ 4s	ok
	Auth. B	736	$<$ 5s	ok
two sessions	Auth. A	48852	59m	ok
	Auth. B	10794	8m30s	ok

We verify the correspondance property that states that whenever Device B finishes the protocol with data d then Device A must have started the protocol with the same data before, by annotating the processes with begin and end events.

Our tool easily verifies that this property indeed holds. We can also show that if the synchronisation message 'start' is removed we find an attack as expected: indeed the attacker may learn k and forge a value $d' = \mathsf{bf}(k', \mathsf{sh}(k, m))$ to be sent to A in the begin of the protocol. Note that the attacker can not chose an arbitrary value for d' as both k and the short hash are compared by the human. We have also verified that, although this scenario is unrealistic, this attack is not possible

$$
\begin{aligned}
A \;&\hat{=}\; \mathbf{in}(c, x_d).(\mathbf{out}(ah_1, \text{ready}) \\
&\qquad || \,\mathbf{out}(c, \text{ready}).\mathbf{in}(ha_1, x_s).[x_s = \text{start}].\mathbf{out}(c, k).(\\
&\qquad\; \mathsf{begin}(x_d).\mathbf{out}(ah_2, \langle k, \mathsf{sh}(k, x_d)\rangle) \,||\, \mathbf{out}(c, \langle k, \mathsf{sh}(k, x_d)\rangle).\mathbf{in}(ha_2, x).[x = \text{ok}])) \\
B \;&\hat{=}\; \mathbf{in}(c, y_d).(\mathbf{out}(bh_1, \text{ready}) \\
&\qquad || \,\mathbf{out}(c, \text{ready}).\mathbf{in}(c, y_k).(\mathbf{out}(bh_2, \langle y_k, \mathsf{sh}(y_k, y_d)\rangle) \\
&\qquad\qquad\qquad || \,\mathbf{out}(c, \langle y_k, \mathsf{sh}(y_k, y_d)\rangle).\mathbf{in}(hb, y).[y = \text{ok}].\mathsf{end}(y_d))) \\
H \;&\hat{=}\; (\mathbf{in}(ah_1, z_a).[z_a = \text{ready}]. \,||\, \mathbf{in}(bh_1, z_b).[z_b = \text{ready}]) :: (\mathbf{out}(ha_1, \text{start}) \,||\, \mathbf{out}(c, \text{start}). \\
&\quad (\mathbf{in}(ah_2, z'_a) \,||\, \mathbf{in}(bh_2, z'_b)) :: [z'_a = z'_b].(\mathbf{out}(ha_2, \text{ok}) \,||\, \mathbf{out}(c, \text{ok}) \,||\, \mathbf{out}(hb, \text{ok}) \,||\, \mathbf{out}(c, \text{ok})))
\end{aligned}
$$

Fig. 7. Manual authentication mechanism 2 from [23] (one session Auth. B)

if we use a strong hash function instead of the short hash. Finally we also checked that the protocol is still secure in the case of two sequential sessions. Given that the protocol uses a human as an intermediate we think that parallel sessions are not a meaningful scenario.

Most of those verifications have been performed in a few seconds on a standard laptop. Actually, only the last experiments (the ones reported on the last line of each table) have been done on a 20 core Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10GHz. We also indicate the number of traces (due to all the possible interleaving of the actions) that have been considered by the tool in order to conclude. Note that in case an attack has been found, the tool stops the exploration before considering all these traces, and this is also indicated in the table.

The fact that the performances degrade when considering an additional session is not surprising and is clearly correlated with the number of traces that have to be considered to carry out the analysis. This is the usual and well-known state-space explosion problem inherent of the concurrent nature of the systems we analyse. As briefly explained in Section VII-C, we have implemented a few straightforward optimisations to mitigate this issue, but more needs to be done in this direction.

b) Manual authentication mechanism 4: This mechanism is decribed in Figure 8. It assumes a simple input interface and standard output interface for device A and a simple output and standard input interface for device B.

scenario		# traces	time	status
one session	Auth. A	35	$<$ 1s	ok
	Auth. B	35	$<$ 1s	ok
without start	Auth. A	16/70	$<$ 1	attack
	Auth. B	21/125	$<$ 1s	attack
k weak	Auth. A	51/175	$<$ 1s	attack
	Auth. B	51/175	$<$ 1s	attack
two sessions	Auth. A	38290	2m	ok
	Auth. B	43876	2m	ok

The protocol relies on a standard hash function and supposes a user who can copy a short random string from device A to device B. As in the previous protocol we assume that data d_A and d_B are given to the devices and that the aim is to ensure

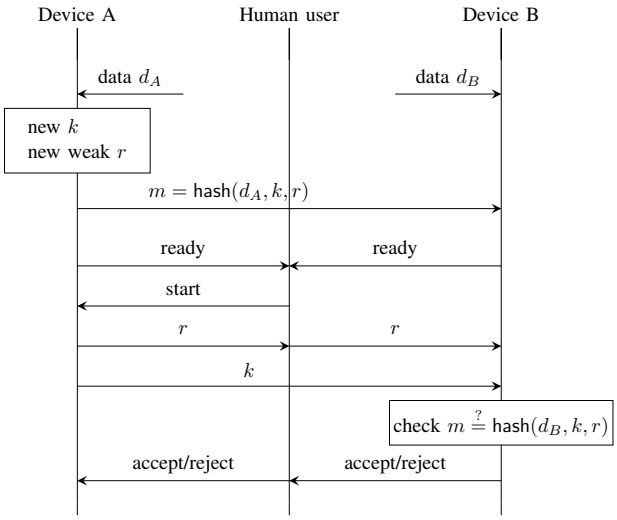

Fig. 8. Manual authentication mechanism 4 from [23]

agreement on these data. Device A generates a strong nonce k, a weak nonce r and sends $\mathsf{hash}(d_A, k, r)$ to B over an insecure network. Both devices then notify the user that they are ready and the user informs A that it may start. A displays r to the user who copies r into B. Moreover, A sends k to B over the insecure network. Device B recomputes the hash and compares it to the previously received value. If both values are identical it outputs accept and reject otherwise. The user enters the displayed result into device A.

We may note that, despite the high number of traces, answers are relatively fast. We consider here an empty equational theory, and therefore the saturation process is faster than for mechanism 2.

IX. Conclusion

In this paper we presented a novel symbolic model for reasoning about protocols that use short authentication strings, possibly generated as the result of a weak hash function. We propose a new procedure for verifying protocols in this model and have implemented the procedure in the AKISS tool. The new capabilities of the tool are illustrated by analysing

several protocols from the ISO/IEC 9798-6:2010 standard for authentication mechanisms for manual data transfer.

As future work we plan to perform additional case studies, as well as checking *injective* correspondance properties. Another possible generalisation is to adapt our framework to verify equivalence properties. The resulting framework might be used for analysing vote anonymity in protocols relying on short vote codes, such as in Pretty Good Democracy [27] or Du-Vote [22].

REFERENCES

[1] https://technologypartner.visa.com/Library/3DSecure.aspx.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, 2001.

[3] https://github.com/LudovicRobin/akiss/.

[4] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. 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 *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, LNCS, pages 281–285. Springer, 2005.

[5] D. A. Basin, C. Cremers, and S. Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. *Journal of Computer Security*, 21(6):817–846, 2013.

[6] D. A. Basin, S. Radomirovic, and L. Schmid. Modeling human errors in security protocols. In *Proceedings of the 29th Computer Security Foundations Symposium (CSF'16)*, pages 325–340. IEEE Computer Society Press, 2016.

[7] B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*, Tallinn, Estonia, July 2005.

[8] B. Blanchet, B. Smyth, and V. Cheval. *Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2016.

[9] Specification of the bluetooth system. Technical report, Bluetooth Special Interest Group, 2001.

[10] Simple pairing whitepaper. Technical report, Bluetooth Special Interest Group, 2006.

[11] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 2016. To appear.

[12] T. Chothia, B. Smyth, and C. Staite. Automatically checking commitment protocols in proverif without false attacks. In *Proceedings of the 4th International Conference on Principles of Security and Trust (POST'15)*, volume 9036 of *Lecture Notes in Computer Science*, pages 137–155. Springer, Apr. 2015.

[13] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *LNCS*, pages 294–307. Springer, 2005.

[14] R. Corin, J. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. *ENTCS*, 121:47–63, 2005.

[15] S. Delaune and F. Jacquemard. Decision procedures for the security of protocols with probabilistic encryption against offline dictionary attacks. *Journal of Automated Reasoning*, 36(1-2):85–124, Jan. 2006.

[16] S. Delaune, S. Kremer, and L. Robin. Formal verification of protocols based on short authenticated strings (extended version). Research report, Inria Nancy - Grand Est, 2017. https://hal.inria.fr/hal-01528603.

[17] D. Dolev and A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[18] C. Ellison. Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399, 2007. http://eprint.iacr.org/2007/399.

[19] 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.

[20] S. Escobar, R. Sasse, and J. Meseguer. Folding variant narrowing and optimal variant termination. *J. Log. Algebr. Program.*, 81(7-8):898–928, 2012.

[21] Google 2-step. https://www.google.com/landing/2step/.

[22] G. S. Grewal, M. Ryan, L. Chen, and M. Clarkson. Du-vote: Remote electronic voting with untrusted computers. In *Proc. 28th Computer Security Foundations Symposium Conference (CSF'15)*, pages 155–169. IEEE Computer Society, 2015.

[23] ISO/IEC 9798-6:2010 information technology – security techniques – entity authentication – part 6: Mechanisms using manual data transfer, 2010. This standard was last reviewed and confirmed in 2016.

[24] G. Lowe. Analysing protocols subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.

[25] D. Pavlovic and C. A. Meadows. Deriving ephemeral authentication using channel axioms. In *Proc. 17th International Workshop on Security Protocols (2009). Revised Selected Papers*, volume 7028 of *Lecture Notes in Computer Science*, pages 240–261. Springer, 2013.

[26] A. Roscoe, T. Smyth, and L. Nguyen. Model checking cryptographic protocols subject to combinatorial attack. 2012.

[27] P. Y. A. Ryan and V. Teague. Pretty good democracy. In *Proc. 17th International Workshop on Security Protocols. Revised Selected Papers*, pages 111–130, 2009.

[28] B. Schmidt, S. Meier, C. Cremers, and D. Basin. The tamarin prover for the symbolic analysis of security protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.