Automating security analysis: symbolic equivalence of constraint systems *

Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune

LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

Abstract. We consider security properties of cryptographic protocols, that are either trace properties (such as confidentiality or authenticity) or equivalence properties (such as anonymity or strong secrecy).
Infinite sets of possible traces are symbolically represented using *deducibility constraints*. We give a new algorithm that decides the trace equivalence for the traces that are represented using such constraints, in the case of signatures, symmetric and asymmetric encryptions. Our algorithm is implemented and performs well on typical benchmarks. This is the first implemented algorithm, deciding symbolic trace equivalence.

1 Introduction

Security protocols are small distributed programs aiming at some security goal, though relying on untrusted communication media. Formally proving that such a protocol satisfies a security property (or finding an attack) is an important issue, in view of the economical and social impact of a failure.

Starting in the 90s, several models and automated verification tools have been designed. For instance both protocols, intruder capabilities and security properties can be formalized within first-order logic and dedicated resolution strategies yield relevant verification methods [18, 21, 6]. Another approach, initiated in [19], consists in symbolically representing the traces using deducibility constraints. Both approaches were quite successful in finding attacks/proving security protocols. There are however open issues, that concern the extensions of the methods to larger classes of protocols/properties [11]. For instance, most efforts and successes only concerned, until recently, *trace properties*, i.e., security properties that can be checked on each individual sequence of messages corresponding to an execution of the protocol. A typical example of a trace property is the *confidentiality*, also called *weak secrecy*: a given message m should not be deducible from any sequence of messages, that corresponds to an execution of the protocol. Agreement properties, also called *authenticity properties*, are other examples of trace properties.

There are however security properties that cannot be stated as properties of a single trace. Consider for instance a voter casting her vote, encrypted with a public key of a server. Since there are only a fixed, known, number of possible

* This work has been partially supported by the ANR project SeSur AVOTÉ.

plaintexts, the confidentiality is not an issue. A more relevant property is the ability to relate the voter's identity with the plaintext of the message. This is a property in the family of *privacy* (or *anonymity*) properties [15]. Another example is the *strong secrecy*: m is strongly secret if replacing m with any m' in the protocol, would yield another protocol that is indistinguishable from the first one: not only m itself cannot be deduced, but the protocol also does not leak any piece of m. These two examples are not trace properties, but *equivalence properties*: they can be stated as the indistinguishability of two processes. In the present paper, we are interested in automating the proofs of equivalence properties. As far as we know, there are only three series of works that consider the automation of equivalence properties for security protocols[1].

The first one [7] is an extension of the first-order logic encoding of the protocols and security properties. The idea is to overlap the two processes that are supposedly equivalent, forming a *bi-process*, then formalize in first-order logic the simultaneous moves (the single move of the bi-process) upon reception of a message. This method checks a stronger equivalence than observational equivalence, hence it fails on some simple (cook up) examples of processes that are equivalent, but their overlapping cannot be simulated by the moves of a single bi-process. The procedure might also not terminate or produce false attacks, but considers an unbounded number of protocol instances.

The second one [3] (and [14]) assumes a fixed (bounded) number of sessions. Because of the infinite number of possible messages forged by an attacker, the number of possible traces is still infinite. The possible traces of the two processes are symbolically represented by two deducibility constraints. Then [3] provides with a decision procedure, roughly checking that the solutions, *and the recipes that yield the solutions* are identical for both constraints. This forces to compute the solutions and the associated recipes and yields an unpractical algorithm.

The third one [17, 9] is based on an extension of the small attack property of [20]. They show that, if two processes are not equivalent, then there must exist a small witness of non-equivalence. A decision of equivalence can be derived by checking every possible small witness. As in the previous method, the main problem is the practicality. The number of small witnesses is very large as all terms of size smaller than a given bound have to be considered. Consequently, neither this method nor the previous one have been implemented.

We propose in this paper another algorithm for deciding equivalence properties. As in [3, 9], we consider *trace equivalence*, which coincides with observational equivalence for determinate processes [14]. In that case, the equivalence problem can be reduced to the symbolic equivalence of finitely many pairs of deducibility constraints, each of which represents a set of traces (see [14]). We consider signatures, pairing, symmetric and asymmetric encryptions, which is slightly less general than [3, 9], who consider arbitrary subterm-convergent theories. The main idea of our method is to simultaneously solve pairs of constraints, instead of solving each constraint separately and comparing the solutions, as

[1] [16] gives a logical characterization of the equivalence properties. It is not clear if this can be of any help in deriving automated decision procedures.

in [3]. These pairs are successively split into several pairs of systems, while preserving the symbolic equivalence: roughly, the father pair is in the relation if, and only if, all the sons pairs are in the relation. This is not fully correct, since, for termination purposes, we need to keep track of some earlier splitting, using additional predicates. Such predicates, together with the constraint systems, yield another notion of equivalence, which is preserved upwards, while the former is preserved downwards. When a pair of constraints cannot be split any more, then the equivalence can be trivially checked.

A preliminary version of the algorithm has been implemented and works well (within a few seconds) on all benchmarks. The same implementation can also be used for checking the static equivalence and for checking the constraints satisfiability. We also believe that it is easier (w.r.t. [3, 9]) to extend the algorithm to a more general class of processes (including disequality tests for instance) and to avoid the detour through trace equivalence. This is needed to go beyond the class of determinate processes.

We first state precisely the problem in Section 2, then we give the algorithm, actually the transformation rules, in Section 3. We sketch the correctness and termination proofs in Section 4 and provide with a short summary of the experiments in Section 5. Detailed proofs of the results can be found in [8].

2 Equivalence properties and deducibility constraints

We use the following straightfoward example for illustrating some definitions:

Example 1. Consider the following very simple handshake protocol:

$$A \to B : \mathsf{enc}(N_A, K_{AB})$$
$$B \to A : \mathsf{enc}(f(N_A), K_{AB})$$

The agent A sends a random message N_A to B, encrypted with a key K_{AB}, that is shared by A and B only. The agent B replies by sending $f(N_A)$ encrypted with the same key. The function f is any function, for instance a hash function.

Consider only one session of this protocol: a sends $\mathsf{enc}(n_a, k_{ab})$ and waits for $\mathsf{enc}(f(n_a), k_{ab})$. The agent b is expecting a message of the form $\mathsf{enc}(x, k_{ab})$. The variable x represents the fact that b does not know in advance what is this randomly generated message. Then he replies by sending out $\mathsf{enc}(f(x\sigma), k_{ab})$. All possible executions are obtained by replacing x with any message $x\sigma$ such that the attacker can supply with $\mathsf{enc}(x\sigma, k_{ab})$ and then with $\mathsf{enc}(f(n_a), k_{ab})$. This is represented by the following constraint:

$$C := \begin{cases} a,\ b,\ \mathsf{enc}(n_a, k_{ab}) \stackrel{?}{\vdash} \mathsf{enc}(x, k_{ab}) \\ a,\ b,\ \mathsf{enc}(n_a, k_{ab}),\ \mathsf{enc}(f(x), k_{ab}) \stackrel{?}{\vdash} \mathsf{enc}(f(n_a), k_{ab}) \end{cases}$$

Actually, C has only one solution: x has to be replaced by n_a. There is no other way for the attacker to forge a message of the form $\mathsf{enc}(x, k_{ab})$.

2.1 Function symbols and terms

We will use the set of function symbols $\mathcal{F} = \mathcal{N} \cup \mathcal{C} \cup \mathcal{D}$ where:

- $\mathcal{C} = \{\mathsf{enc},\ \mathsf{aenc},\ \mathsf{pub},\ \mathsf{sign},\ \mathsf{vk},\ \langle\ \rangle\}$ is the set of *constructors*;
- $\mathcal{D} = \{\mathsf{dec},\ \mathsf{adec},\ \mathsf{check},\ \mathsf{proj}_1,\ \mathsf{proj}_2\}$ is the set of *destructors*;
- $\mathcal{N}$ is a set of constants, called *names*.

In addition, $\mathcal{X}$ is a set of variables x, y, z,... The *constructor terms* (resp. *ground constructor terms*) are built on $\mathcal{C}$, $\mathcal{N}$ and $\mathcal{X}$ (resp. $\mathcal{C}, \mathcal{N}$). The term rewriting system below is convergent: we let $t{\downarrow}$ be the normal form of t.

$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pub}(y)), y) \to x \qquad \mathsf{proj}_1(\langle x, y\rangle) \to x \qquad \mathsf{dec}(\mathsf{enc}(x, y), y) \to x$$
$$\mathsf{check}(\mathsf{sign}(x, y), \mathsf{vk}(y)) \to x \qquad \mathsf{proj}_2(\langle x, y\rangle) \to y$$

A (ground) *recipe* records the attacker's computation. It is used as a witness of how some deduction has been performed. Formally, it is a term built on $\mathcal{C}, \mathcal{D}$ and a set of special variables $\mathcal{AX} = \{ax_1, \ldots, ax_n, \ldots\}$, that can be seen as pointers to the hypotheses, or known messages. Names are excluded from recipes: names that are known to the attacker must be given explicitly as hypotheses.

Example 2. Given $\mathsf{enc}(a, b)$ and b, the recipe $\zeta = \mathsf{dec}(ax_1, ax_2)$ is a witness of how to deduce a: $\zeta\{ax_1 \mapsto \mathsf{enc}(a, b); ax_2 \mapsto b\}{\downarrow} = a$.

The recipes are generalized, including possibly variables that range over recipes: (general) recipes are terms built on $\mathcal{C}, \mathcal{D}, \mathcal{AX}$ and $\mathcal{X}_r$, a set of recipe variables, that are written using capital letters $X, X_1, X_2, \ldots$.

We denote by $var(u)$ is the set of variables of any kind that occur in u.

2.2 Frames

The *frame* records the messages that have been sent by the participants of the protocol; it is a symbolic representation of a set of sequences of messages. The frame is also extended to record some additional informations on attacker's deductions. Typically $\mathsf{dec}(X, \zeta), i \triangleright u$ records that, using a decryption with the recipe ζ, on top of a recipe X, allows to get u (at stage i). After recording this information in the frame, we may forbid the attacker to use a decryption on top of X, forcing him to use this "direct access" from the frame.

Definition 1. *A* frame ϕ *is a sequence* $\zeta_1, i_1 \triangleright u_1, \ldots, \zeta_n, i_n \triangleright u_n$ *where* $u_1, \ldots, u_n$ *are constructor terms,* $i_1, \ldots, i_n \in \mathbb{N}$*, and* $\zeta_1, \ldots, \zeta_n$ *are general recipes. The* domain *of the frame* ϕ*, denoted* $dom(\phi)$*, is the set* $\{\zeta_1, \ldots, \zeta_n\} \cap \mathcal{AX}$*. It must be equal to* $\{ax_1, \ldots, ax_m\}$ *for some* m *that is called the* size *of* ϕ*. A frame is* closed *when* $u_1, \ldots, u_n$ *are ground terms and* $\zeta_1, \ldots, \zeta_n$ *are ground recipes.*

Example 3. The messages of Example 1 are recorded in a frame of size 4.

$$\{ax_1, 1 \triangleright a,\ ax_2, 2 \triangleright b,\ ax_3, 3 \triangleright \mathsf{enc}(n_a, k_{ab}),\ ax_4, 4 \triangleright \mathsf{enc}(f(x), k_{ab})\}.$$

A frame ϕ defines a substitution $\{ax \mapsto u \mid ax \in dom(\phi), ax \triangleright u \in \phi\}$. A closed frame is *consistent* if, for every $\zeta \triangleright u \in \phi$, we have that $\zeta\phi{\downarrow} = u$.

2.3 Deducibility constraints

The following definitions are consistent with [12]. We generalize however the usual definition, including equations between recipes, for example, in order to keep track of some choices in our algorithm.

Definition 2. *A* deducibility constraint *(sometimes called simply* constraint *in what follows) is either* $\bot$ *or consists of:*

1. *a subset* S *of* $\mathcal{X}$ *(the free variables of the constraint);*
2. *a frame* ϕ*, whose size is some* m*;*
3. *a sequence* $X_1, i_1 \overset{?}{\vdash} u_1;\ \ldots;\ X_n, i_n \overset{?}{\vdash} u_n$ *where*
   - $X_1, \ldots, X_n$ *are distinct variables in* $\mathcal{X}_r$*,* $u_1, \ldots, u_n$ *are constructor terms, and* $0 \leq i_1 \leq \ldots \leq i_n \leq m$*.*
   - *for every* $0 \leq k \leq m$*,* $var(ax_k\phi) \subseteq \bigcup_{i_j<k} var(u_j)$*;*
4. *a conjunction* E *of equations and disequations between terms;*
5. *a conjunction* E' *of equations and disequations between recipes.*

The variables X_i represent the recipes that might be used to deduce the right hand side of the deducibility constraint. The indices indicate which initial segment of the frame can be used. We use this indirect representation, instead of the seemingly simpler notation of Example 1, because the transformation rules that will change the frame don't need then to be reproduced on all relevant left sides of deducibility constraints.

Example 4. Back to Example 1, the deducibility constraint is formally given by $S = \{x, y\}$, $E = E' = \emptyset$, the frame ϕ as in Example 3 and the sequence:

$$D \ = \ X_1, 3 \overset{?}{\vdash} \mathsf{enc}(x, k_{ab});\ \ X_2, 4 \overset{?}{\vdash} \mathsf{enc}(f(n_a), k_{ab}).$$

For sake of simplicity, in what follows, we will forget about the first component (the free variables). This is justified by an invariant of our transformation rules: initially all variables are free and each time new variables are introduced, their assignment is determined by an assignment of the free variables.

Definition 3. *A* solution *of a deducibility constraint* $C = (\phi, D, E, E')$ *consists of a mapping* σ *from variables to ground constructor terms and a substitution* θ *mapping* $\mathcal{X}_r$ *to ground recipes, such that:*

- *for every* $\zeta, i \rhd u \in \phi$*,* $var(\zeta\theta) \subseteq \{ax_1, \ldots, ax_i\}$ *and* $\zeta\theta(\phi\sigma)\downarrow = u\sigma\downarrow$ *(i.e. the frame is consistent after instanciating the variables);*
- *for every* $X_i, j \overset{?}{\vdash} u_i$ *in* D*,* $var(X_i\theta) \subseteq \{ax_1, \ldots, ax_j\}$ *and* $X_i\theta(\phi\sigma)\downarrow = u_i\sigma\downarrow$*;*
- *for every equation* $u \overset{?}{=} v$ *(resp.* $u \overset{?}{\neq} v$*) in* E*,* $u\sigma\downarrow = v\sigma\downarrow$ *(resp.* $u\sigma\downarrow \neq v\sigma\downarrow$*);*
- *for every equation* $\zeta \overset{?}{=} \zeta'$ *(resp.* $\zeta \overset{?}{\neq} \zeta'$*) in* E'*,* $\zeta\theta = \zeta'\theta$ *(resp.* $\zeta\theta \neq \zeta'\theta$*).*

$\mathsf{Sol}(C)$ *is the set of solutions of* C*. By convention,* $\mathsf{Sol}(\bot) = \emptyset$*.*

Example 5. Coming back to Example 4, a solution is (σ, θ) with:

- $\sigma = \{x \mapsto n_a,\ y \mapsto \langle a, \mathsf{enc}(n_a, k_{ab})\rangle\}$, and
- $\theta = \{X_1 \mapsto ax_3,\ X_2 \mapsto ax_4,\ X_3 \mapsto \langle ax_1, ax_3\rangle\}$.

Each solution of a constraint corresponds to a possible execution of the protocol, together with the attacker's actions that yield this execution. For instance an attack on the confidentiality of a term s can be modeled by adding $X, m \overset{?}{\vdash} s$ to the constraint system (X is a fresh variable and m is the size of the frame). This represents the derivability of s from the messages sent so far. Note that there might be several attacker's recipes yielding the same trace.

Example 6. Consider another very simple example: the Encrypted Password Transmission protocol [13], which is informally described by the rules:

$$A \to B : \langle N_A, \mathsf{pub}(K_A)\rangle$$
$$B \to A : \mathsf{aenc}(\langle N_A, P\rangle, \mathsf{pub}(K_A))$$

Assume that a first sends a message whereas b is waiting for a message of the form $\langle x, \mathsf{pub}(k_a)\rangle$. Then b responds by sending $\mathsf{aenc}(\langle x, p\rangle, \mathsf{pub}(k_a))$. The corresponding deducibility constraint is (S, ϕ, D, E, E') where $S = \{x, y\}$, $E = E' = \emptyset$, and the sequences ϕ and D are as follows:

$$\phi = \begin{cases} ax_1, 1 \rhd \mathsf{pub}(k_a);\ ax_2, 2 \rhd \mathsf{pub}(k_b); \\ ax_3, 3 \rhd \langle n_a, \mathsf{pub}(k_a)\rangle; \\ ax_4, 4 \rhd \mathsf{aenc}(\langle x, p\rangle, \mathsf{pub}(k_a)) \end{cases} \quad D = \begin{cases} X_1, 3 \overset{?}{\vdash} \langle x, \mathsf{pub}(k_a)\rangle \\ X_2, 4 \overset{?}{\vdash} \mathsf{aenc}(\langle n_a, y\rangle, \mathsf{pub}(k_a)) \end{cases}$$

There are several solutions. For instance, the "honest solution" (σ_h, θ_h) is given by $\sigma_h = \{x \mapsto n_a, y \mapsto p\}$ and $\theta_h = \{X_1 \mapsto ax_3, X_2 \mapsto ax_4\}$. Another solution is (σ, θ) where $\sigma = \{x \mapsto \mathsf{pub}(k_a), y \mapsto n_a\}$ and $\theta = \{X_1 \mapsto \langle ax_1, ax_1\rangle, X_2 \mapsto \mathsf{aenc}(\langle \mathsf{proj}_1(ax_3), \mathsf{proj}_1(ax_3)\rangle, ax_1)\}$.

2.4 Static equivalence

Two sequences of terms are *statically equivalent* if, whatever an attacker observes on the first sequence, the same observation holds on the second sequence [2]:

Definition 4. *Two closed frames ϕ and ϕ' having the same size m are* statically equivalent*, which we write $\phi \sim_s \phi'$, if*

1. *for any ground recipe ζ such that $var(\zeta) \subseteq \{ax_1, \ldots, ax_m\}$, we have that*
   $$\zeta\phi{\downarrow} \text{ is a constructor term if, and only if, } \zeta\phi'{\downarrow} \text{ is a constructor term}$$
2. *for any ground recipes ζ, ζ' such that $var(\{\zeta, \zeta'\}) \subseteq \{ax_1, \ldots, ax_m\}$, and the terms $\zeta\phi{\downarrow}$, $\zeta'\phi{\downarrow}$ are constructor terms, we have that*
   $$\zeta\phi{\downarrow} = \zeta'\phi{\downarrow} \text{ if, and only, if } \zeta\phi'{\downarrow} = \zeta'\phi'{\downarrow}.$$

Example 7. Consider the frames $\phi_1 = \{ax_1 \rhd a, ax_2 \rhd \mathsf{enc}(a, b), ax_3 \rhd b\}$ and $\phi_2 = \{ax_1 \rhd a, ax_2 \rhd \mathsf{enc}(c, b), ax_3 \rhd b\}$. $\phi_1 \not\sim_s \phi_2$ since choosing $\zeta = \mathsf{dec}(ax_2, ax_3)$ and $\zeta' = ax_1$ yields $\zeta\phi_1{\downarrow} = \zeta'\phi_1{\downarrow} = a$ while $\zeta\phi_2{\downarrow} \neq \zeta'\phi_2{\downarrow}$.

On the other hand, $\{ax_1 \rhd a, ax_2 \rhd \mathsf{enc}(a, b)\} \sim_s \{ax_1 \rhd a, ax_2 \rhd \mathsf{enc}(c, b)\}$ since, intuitively, there is no way to open the ciphertexts or to construct them, hence no information on the content may leak.

2.5 Symbolic equivalence

Now we wish to check static equivalence on any possible trace. This is captured by the following definition:

Definition 5. *Let C and C' be two constraints whose corresponding frames are ϕ and ϕ'. C is* symbolically equivalent *to C', $C \approx_s C'$, if:*
- for all $(\theta, \sigma) \in \mathsf{Sol}(C)$, there exists σ' such that $(\theta, \sigma') \in \mathsf{Sol}(C')$, and $\phi\sigma \sim_s \phi'\sigma'$,
- for all $(\theta, \sigma') \in \mathsf{Sol}(C')$, there exists σ such that $(\theta, \sigma) \in \mathsf{Sol}(C)$, and $\phi\sigma \sim_s \phi'\sigma'$.

Example 8. As explained for instance in [3], the security of the handshake protocol against offline guessing attacks can be modeled as an equivalence property between two samples of the protocol instance, one in which, at the end of the protocol, the key is revealed and the other in which a random number is revealed instead. This amounts to check the symbolic equivalence of the two constraints:

- $C_1 = (\phi \cup \{ax_5, 5 \triangleright k_{ab}\}, D \cup \{X_3, 5 \overset{?}{\vdash} y\}, \emptyset, \emptyset)$, and
- $C_2 = (\phi \cup \{ax_5, 5 \triangleright k\}, D \cup \{X_3, 5 \overset{?}{\vdash} y\}, \emptyset, \emptyset)$

where D is as in Example 4 and ϕ is as in Example 3.

The constraints C_1 and C_2 are *not* symbolically equivalent: considering the assignment $\sigma = \{x \mapsto n_a, y \mapsto n_a\}$, there is a recipe $X_3\theta = \mathsf{dec}(ax_3, ax_5)$ yielding this solution, while any solution σ' of C_2 maps x to n_a and, if $X_3\theta = \mathsf{dec}(ax_3, ax_5)$, we must have $y\sigma'{\downarrow} = \mathsf{dec}(\mathsf{enc}(n_a, k_{ab}), k)$, which is not possible since this is not a constructor term.

Any trace equivalence problem can be expressed as an instance of the equivalence of an *initial pair of constraints*, that is a pair of the form (ϕ_1, D_1, E_1, E_1'), (ϕ_2, D_2, E_2, E_2') in which:

- $E_1' = E_2' = \emptyset$, and E_1, E_2 only contain equations;
- $\phi_1 = \{ax_1, 1 \triangleright u_1, \ldots, ax_m, m \triangleright u_m\}$, and $D_1 = X_1, i_1 \overset{?}{\vdash} s_1;\ \ldots;\ X_n, i_n \overset{?}{\vdash} s_n$;
- $\phi_2 = \{ax_1, 1 \triangleright v_1, \ldots, ax_m.m \triangleright v_m\}$, and $D_2 = X_1, i_1 \overset{?}{\vdash} t_1;\ \ldots;\ X_n, i_n \overset{?}{\vdash} t_n$.

Or else it is a pair as above, in which one of the components is replaced with $\bot$.

In particular, the number of components in the frame and in the deducibility part are respectively identical in the two constraints, when none of them is $\bot$. *This will be an invariant in all our transformation rules.* Hence we will always assume this without further mention. This is unchanged by the transformations, unless the constraint becomes $\bot$. We keep the notation m for the size of the frames. Finally, the consistency of the frame after instanciation (the first condition of Definition 3) is satisfied for all solutions of initial constraints and is again an invariant, hence we will not care of this condition.

As explained in [14], such initial constraints are sufficient for our applications. The case where one of the component is $\bot$ solves the satisfiability problem for the constraint: the constraint solving procedure of [12] solves this specific instance.

3 Transformation rules

The main result of this paper is a decision procedure for symbolic equivalence of an initial pair of constraints:

Theorem 1. *Given an initial pair (C, C'), it is decidable whether $C \approx_s C'$.*

This result in itself is already known (e.g. [3, 9]), but, as claimed in the introduction, the known algorithms cannot yield any reasonable implementation. We propose here a new algorithm/proof, which is implemented. As pointed in [14], this yields a decision algorithm for the observational equivalence of simple processes without replication nor else branch. The class of simple processes captures most existing protocols.

The decision algorithm works by rewriting pairs of constraints, until a trivial failure or a trivial success is found. These rules are branching: they rewrite a pair of constraints into two pairs of constraints. Transforming the pairs of constraints therefore builds a binary tree. Termination requires to keep track of some information, that is recorded using flags, which we describe first. In Section 4, we show that the tree is then finite: the rules are terminating. The transformation rules are also correct: if all leaves are success leaves, then the original pair of constraints is equivalent. They are finally complete: if the two original constraints are equivalent then any of two pairs of constraints resulting from a rewriting steps are also equivalent.

3.1 Flags

The flags are additional constraints that restrict the recipes. We list them here, together with (a sketch of) their semantics.

Constraints $X, i \overset{?}{\vdash}_F u$ may be indexed with a set F consisting of propositions $\texttt{NoCons}_f$ where f is a constructor. Any solution (θ, σ) such that $X\theta$ is headed with f is then excluded. Expressions $\zeta, j \triangleright_F u$ in a frame are indexed with a set F consisting of:

- $\texttt{NoCons}_f$ (as above) discards the solutions (θ, σ) such that a subterm of a recipe allows to deduce $u\sigma$ using f as a last step.
- $\texttt{NoDest}_f(i)$ where f is a destructor and $i \leq m$ discards the solutions (θ, σ) such that there exists $X, j \overset{?}{\vdash} v$ with $j \leq i$ and $\zeta'_2, \ldots, \zeta'_n$ where $f(\zeta\theta, \zeta'_2, \ldots, \zeta'_n)$ occurs as a subterm in $X\theta$, unless we use a shortcut explicitly given in the frame.
- $\texttt{NoUse}$. The corresponding elements of the frame cannot be used in any recipe, and avoids shifting the indices.

3.2 The rules

The rules are displayed in Figure 1 for single constraints. We explain in Section 3.3 how they are applied to pairs of constraints (an essential feature of our

algorithm). A simple idea would be to guess the top function symbol of a recipe and replace the recipe variable with the corresponding instance. When the head symbol of a recipe is a constructor and the corresponding term is not a variable, this is nice, since the constraint becomes simpler. This is the purpose of the rule Cons. When the top symbol of a recipe is a destructor, the constraint becomes more complex, introducing new terms, which yields non-termination.

Our strategy is different for destructors: we switch roughly from the top position of the recipe to the *redex position*. Typically, in case of symmetric encryption, if a ciphertext is in the frame, we will guess whether the decryption key is deducible, and at which stage.

The Cons rule simply guesses whether the top symbol of the recipe is a constructor f. Either it is, and then we can split the constraint, or it is not and we add a flag forbidding this. The rule Axiom also guesses whether a trivial recipe can be applied. If so, the constraint can simply be removed. Otherwise, it means that the right-hand-side of the deducibility constraint is different from the members of the frame. The Dest rule is more tricky. If v is a non-variable member of the frame, that can be unified with a non variable subterm of a left side of a rewrite rule (for instance v is a ciphertext), we guess whether the rule can be applied to v. This corresponds to the equation $u_1 \stackrel{?}{=} v$, that yields an instance of w, the right member of the rewrite rule, provided that the rest of the left member is also deducible: we get constraints $X_2, i \stackrel{?}{\vdash} u_2; \ldots; X_n, i \stackrel{?}{\vdash} u_n$. The flag `NoDest` is added in any case to the frame, since we either already applied the destructor, and this application result is now recorded in the frame by $f(\zeta, X_2, \ldots, X_n), i \triangleright w$, or else it is assumed that f applied to v will not yield a redex.

The remaining rules cover the comparisons that an attacker could perform at various stages. The equality rules guess equalities between right sides of deducibility constraints and/or members of the frame. If a member of the frame is deducible at an early stage, then this message does not bring any new information to the attacker: it becomes useless, hence the `NoUse` flag.

Finally, the last rule is the only rule that is needed to get in addition a static equivalence decision algorithm, as in [1]. Thanks to this rule, if a subterm of the frame is deducible, then there will be a branch in which it is deduced.

3.3 How to use the transformation rules

In the previous section we gave rules that apply on a single constraint. We explain here how they are extended to pairs of constraints. If one of the constraint is $\bot$, then we proceed as if there was a single constraint. Otherwise, the indices i (resp. i_1, i_2) and the recipes X, ζ (resp. $X_1, X_2, \zeta_1, \zeta_2$) matching the left side of the rules *must be identical in both constraints*: we apply the rules at the same positions in both constraints.

We have to explain now what happens when, on a given pair (C, C') a rule can be applied on C and not on C' (or the converse).

$$\underline{\text{Cons}} : X, i \overset{?}{\vdash}_F f(t_1, \ldots, t_n) \rightarrow \begin{array}{l} X_1, i \overset{?}{\vdash}_F t_1; \cdots; X_n, i \overset{?}{\vdash}_F t_n; X \overset{?}{=} f(X_1, \ldots, X_n) \\ X, i \overset{?}{\vdash}_{F+\texttt{NoCons}_f} f(t_1, \ldots, t_n) \end{array}$$

If $\texttt{NoCons}_f \notin F$ and $X_1, \ldots X_n$ are fresh variables.

$$\underline{\text{Axiom}} : X, i \overset{?}{\vdash}_F v \rightarrow \begin{array}{l} u \overset{?}{=} v;\ X \overset{?}{=} \zeta \\ X, i \overset{?}{\vdash}_F v;\ X \overset{?}{\neq} \zeta \end{array}$$

If $v \notin \mathcal{X}$, ϕ contains $\zeta, j \rhd_G u$ with $\texttt{NoUse} \notin G$, and $i \geq j$.

$$\underline{\text{Dest}} : \zeta, y \rhd_G v \rightarrow \begin{array}{l} X_2, i \overset{?}{\vdash} u_2;\ \cdots;\ X_n, i \overset{?}{\vdash} u_n;\ u_1 \overset{?}{=} v;\ \zeta, j \rhd_{G+\texttt{NoDest}_f(m)} v; \\ \qquad\qquad f(\zeta, X_2, \ldots, X_n), i \rhd w \\ \zeta, j \rhd_{G+\texttt{NoDest}_f(i)} v \end{array}$$

If $v \notin \mathcal{X}$, $\texttt{NoUse} \notin G$, there is a rewrite rule $f(u_1, \ldots, u_n) \rightarrow w$, $k < i$ whenever $\texttt{NoDest}_f(k) \in G$ and i is minimal such that $j \leq i$ and there is some constraint $X, i \overset{?}{\vdash} w$ ($i = m$ if there is no such constraint).

$$\underline{\text{Eq-left-left}} : \zeta_1, i_1 \rhd_{F_1} u_1;\ \zeta_2, i_2 \rhd_{F_2} u_2 \rightarrow \begin{array}{l} \zeta_1, i_1 \rhd_{F_1} u_1;\ \zeta_2, i_2 \rhd_{F_2} u_1;\ u_1 \overset{?}{=} u_2 \\ \zeta_1, i_1 \rhd_{F_1} u_1;\ \zeta_2, i_2 \rhd_{F_2} u_2;\ u_1 \overset{?}{\neq} u_2 \end{array}$$

If $\texttt{NoUse} \notin F_1 \cup F_2$ and $i_1 \leq i_2$.

$$\underline{\text{Eq-right-right}} : X_2, i_2 \overset{?}{\vdash} u_2 \rightarrow \begin{array}{l} X_1 = X_2;\ u_1 \overset{?}{=} u_2 \\ X_2, i_2 \overset{?}{\vdash} u_2;\ u_1 \overset{?}{\neq} u_2 \end{array}$$

If $X_1, i_1 \overset{?}{\vdash} u_1;$ and $i_1 \leq i_2$.

$$\underline{\text{Eq-left-right}} : \zeta, j \rhd_G v \rightarrow \begin{array}{l} \zeta, j \rhd_{G+\texttt{NoUse}} u;\ u \overset{?}{=} v \\ \zeta, j \rhd_G v;\ u \overset{?}{\neq} v \end{array}$$

If $X, i \overset{?}{\vdash}_F u;$, $\texttt{NoUse} \notin G$ and $j > i$.

$$\underline{\text{Ded-subterms}} : \zeta, i \rhd_F f(u_1, \ldots, u_n) \rightarrow \begin{array}{l} X_1, m \overset{?}{\vdash} u_1;\ \cdots;\ X_n, m \overset{?}{\vdash} u_n; \\ \qquad\qquad \zeta, i \rhd_{F+\texttt{NoCons}_f} u \\ \zeta, i \rhd_{F+\texttt{NoCons}_f} f(u_1, \ldots, u_n) \end{array}$$

If $\texttt{NoCons}_f, \texttt{NoUse} \notin F$ and $X_1, \ldots, X_n$ are fresh variables.

All rules assume that the equations have a mgu and that this mgu is eagerly applied to the resulting constraint without yielding any trivial disequation.

Fig. 1. Transformation rules

Example 9. Let $C = (\phi, D, E, E')$ and $C' = (\phi, D', E, E')$ where $E = E' = \emptyset$, $\phi = ax_1, 1 \triangleright a$, $D = X, 1 \overset{?}{\vdash} \mathsf{enc}(x_1, x_2)$, and $D' = X, 1 \overset{?}{\vdash} x$. The rule Cons can be applied on C and not on C'. However, we have to consider solutions where $\mathsf{enc}(x_1, x_2)\sigma$ and $x\sigma'$ are both obtained by a construction. Hence, it is important to enable this rule on both sides. For this, we first apply the substitution $x \mapsto \mathsf{enc}(y_1, y_2)$ where y_1, y_2 are fresh variables. This yields the two pairs of constraints (C_1, C_1') and (C_2, C_2') (forgetting about equations):

- $C_1 = (\phi, X_1, 1 \overset{?}{\vdash} x_1; X_2, 1 \overset{?}{\vdash} x_2)$ and $C_1' = (\phi, X_1, 1 \overset{?}{\vdash} y_1;\ X_2, 1 \overset{?}{\vdash} y_2)$;
- $C_2 = (\phi, X, 1 \overset{?}{\vdash}_{\mathtt{NoCons}_{\mathsf{enc}}} \mathsf{enc}(x_1, x_2))$ and $C_2' = (\phi, X, 1 \overset{?}{\vdash}_{\mathtt{NoCons}_{\mathsf{enc}}} x)$.

Therefore, the rule Cons, (this is similar for Ded-subterms), when applied to pairs of constraints comes in three versions: either the rule is applied on both sides or, if $X, i \overset{?}{\vdash} f(t_1, \ldots, t_n)$ (resp. $\zeta \triangleright f(t_1, \ldots, t_n)$) is in C, and $X, i \overset{?}{\vdash} x$ (resp. $\zeta \triangleright x$) is in C', we may apply the rule on the pair of constraints, adding to C' the equation $x \overset{?}{=} f(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are fresh variables. The third version is obtained by switching C and C'. This may introduce new variables, that yield a termination issue, which we discuss in Section 4.1. Similarly, the rules Axiom and Dest assume that $v \notin \mathcal{X}$. This has to be satisfied by C or C'. In case of the rule Dest, this means that the variables of the rewrite rule might not be immediately eliminated: this may also introduce new variables. For the rules Eq-left-left, Eq-right-right and Eq-left-right, we require that at least one new non-trivial equality (or disequality) is added to one of the two constraints (otherwise there is a trivial loop).

For all rules, if a rule is applicable on one constraint and not the other, we do perform the transformation, however replacing a constraint with $\bot$ when a condition becomes false or meaningless. Furthermore, we also replace a constraint C with $\bot$ when:

- the rule Dest cannot be applied on C; and
- C contains a constraint $X, i \overset{?}{\vdash} v$ such that v is not a variable and the rules Cons and Axiom cannot be applied to it.

Altogether this yields a transformation relation $(C, C') \to (C_1, C_1'), (C_2, C_2')$ on pairs of constraints: a node labeled (C, C') has two sons, respectively labeled (C_1, C_1') and (C_2, C_2').

Our algorithm can be stated as follows:

- Construct, from an initial pair of constraints (C_0, C_0') a tree, by applying as long as possible a transformation rule to a leaf of the tree.
- If, at some point, there is a leaf to which no rule is applicable and that is labeled $(C, \bot)$ or $(\bot, C)$ where $C \neq \bot$, then we stop with $C_0 \not\approx_s C_0'$.
- Otherwise, if the construction of the tree stops without reaching such a failure, return $C_0 \approx_s C_0'$.

Our algorithm can also be used to decide static equivalence of frames, as well as the (un)satisfiability of a constraint. Furthermore, in case of failure, a witness of the failure can be returned, using the equations of the non-$\bot$ constraint.

4 Correctness, completeness and termination

4.1 Termination

In general, the rules might not terminate, as shown by the following example:

Example 10. Consider the initial pair of contraints (C, C') given below:

$$C = \begin{cases} a \overset{?}{\vdash} \mathsf{enc}(x_1, x_2) \\ a, b \overset{?}{\vdash} x_1 \end{cases} \qquad C' = \begin{cases} a \overset{?}{\vdash} y_1 \\ a, b \overset{?}{\vdash} \mathsf{enc}(y_1, y_2) \end{cases}$$

We may indeed apply Cons yielding (on one branch):

$$C_1 = \begin{cases} a \overset{?}{\vdash} x_1 \\ a \overset{?}{\vdash} x_2 \\ a, b \overset{?}{\vdash} x_1 \end{cases} \qquad C'_1 = \begin{cases} a \overset{?}{\vdash} z_1 \\ a \overset{?}{\vdash} z_2 \\ a, b \overset{?}{\vdash} \mathsf{enc}(\mathsf{enc}(z_1, z_2), y_2) \end{cases} \quad \text{and } y_1 \overset{?}{=} \mathsf{enc}(z_1, z_2)$$

Then, again using Cons, we get back as a subproblem the original constraints.

Fortunately, there is a simple complete strategy that avoids this behavior, by breaking the symmetry between the two constraints components. We assume in the following that, applying

- Cons to (C, C') where $X, i \overset{?}{\vdash} x \in C$ and $X, i \overset{?}{\vdash} f(t_1, \ldots, t_n) \in C'$,
- Ded-subterms to (C, C') where $\zeta, j \rhd x \in C$ and $\zeta, j \rhd f(t_1, \ldots, t_n) \in C'$,
- Dest to (C, C') where $X, i \overset{?}{\vdash} u; \zeta, j \rhd x \ \in C$ and $X, i \overset{?}{\vdash} u'; \zeta, j \rhd v' \ \in C'$

are only allowed when no other rule can be applied.

There is however no such restriction, when we switch the elements of the pair. If we come back to Example 10, we still apply the same transformation rule to the pair (C, C'), but we cannot apply Cons to (C_1, C'_1) since Eq-right-right can be applied to the constraint C_1, yielding a failure: $C \not\approx_s C'$.

Lemma 1. *With the above strategy, the transformation rules are terminating on any initial pair of constraint systems.*

Idea of the proof: as long as no new first-order variable is introduced, the set of first-order terms appearing in the constraint is roughly bounded by the subterms of the constraint. (This relies on the properties of the rewrite system). Loops are then prevented by the flags. Now, because of the eager application of substitutions, the only cases in which new first-order variables are introduced are the above cases of applications of Cons, Ded-subterms and Dest. Until new variables are introduced in the right constraints, the above argument applies:

the sequence of transformations is finite. Then, according to the strategy, when new variables are introduced on the right constraint, no other rule may apply. This implies that the left constraint (considered in isolation) is irreducible: it is of the form $X_1, i_1 \overset{?}{\vdash} x_1, \ldots, X_n, i_n \overset{?}{\vdash} x_n, \ldots$ where $x_1, \ldots x_n$ are distinct variables (which we call a *solved constraint*). From this point onwards, the rules DEST, DED-SUBTERMS will never be applicable and therefore, no element will be added to the frames. Then, either usable elements of the frames are strictly decreasing (using a EQ-LEFT-RIGHT) or else we preserve the property of being solved on the left. In the latter case, the first termination argument can be applied to the right constraint.

4.2 Correctness

The transformation rules yield a finite tree labeled with pairs of constraints.

Lemma 2. *If all leaves of a tree, whose root is labeled with (C_0, C_0') (a pair of initial constraints), are labeled either with $(\bot, \bot)$ or with some (C, C') with $C \neq \bot, C' \neq \bot$, then $C_0 \approx_s C_0'$.*

The idea of the proof is to first analyse the structure of the leaves. We introduce a restricted symbolic equivalence $\approx_s^r$ such that $C \approx_s^r C'$ for any leaf whose two label components are distinct from $\bot$. Roughly, this restricted equivalence will only consider the recipes that satisfied the additional constraints induced by the flags. Then we show that $\approx_s^r$ is preserved upwards in the tree: for any transformation rule, if the two pairs of constraints labeling the sons of a node are respectively in $\approx_s^r$, then the same property holds for the father. Finally, $\approx_s^r$ coincides with $\approx_s$ on the initial constraints (that contain no flag).

4.3 Completeness

We prove that the symbolic equivalence is preserved by the transformation rules, which yields:

Lemma 3. *If (C_0, C_0') is a pair of initial constraints such that $C_0 \approx_s C_0'$, then all leaves of a tree, whose root is labeled with (C_0, C_0'), are labeled either with $(\bot, \bot)$ or with some (C, C') with $C \neq \bot$ and $C' \neq \bot$.*

5 Implementation and experiments

An Ocaml implementation of an early version of the procedure described in this paper, as well as several examples, are available at `http://www.lsv.ens-cachan.fr/~cheval/programs/index.php` (around 5000 lines of Ocaml). Our implementation closely follows the transformation rules that we described. For efficiency reasons, a strategy on checking the rules applicability has been designed in addition.

We checked the implementation on examples of static equivalence problems, on examples of satisfiability problems, and on symbolic equivalence problems that come from actual protocols. On all examples the tool terminates in less than a second (on a standard laptop). Note that the input of the algorithm is a pair of constraints: checking the equivalence of protocols would require in addition an interleaving step, that could be expensive.

We have run our tool on the following family of examples presented in [5]:

$$\phi_n = \{ax_1 \triangleright t_n^0, ax_2 \triangleright c_0, ax_3 \triangleright c_1\} \text{ and } \phi'_n = \{ax_1 \triangleright t_n^1, ax_2 \triangleright c_0, ax_3 \triangleright c_1\}$$

where $t_0^i = c_i$ and $t_{n+1}^i = \langle \mathsf{enc}(t_n^i, k_n^i), k_n^i \rangle$, $i \in \{0, 1\}$. In these examples, the size of the distinguishing tests increase exponentially while the sizes of the frames grow linearly. As KiSs [10], our tool outperforms YAPA [4] on such examples.

For symbolic equivalences, we cannot compare with other tools (there is no such tools); we simply tested the program on some home made benchmarks as well as on the handshake protocol, several versions of the encrypted password transmission protocol, the encrypted key exchange protocol [13], each for the offline guessing attack property. We checked also the strong secrecy for the corrected Dennin-Sacco key distribution protocol. Unfortunately we cannot (yet) check anonymity properties for e-voting protocols, as we would need to consider more cryptographic primitives.

6 Conclusion

We presented a new algorithm for deciding symbolic equivalence, which performs well in practice. There is still some work to do for extending the results and the tool. First, we use trace equivalence, which requires to consider all interleavings of actions; for each such interleaving, a pair of constraints is generated, which is given to our algorithm. This requires an expensive overhead (which is not implemented), that might be unnecessary. Instead, we wish to extend our algorithm, considering pairs of sets of constraints and use a symbolic bisimulation. This looks feasible and would avoid the detour through trace equivalence. This would also allow drop the determinacy assumption on the protocols and to compare our method with ProVerif [7].

We considered only positive protocols; we wish to extend the algorithm to non-positive protocols, allowing disequality constraints from the start. Finally, we need to extend the method to other cryptographic primitives, typically blind signatures and zero-knowledge proofs.

Acknowledgments. We wish to thank Sergiu Bursuc for fruitful discussions.

References

1. M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1–2):2–32, 2006.

2. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, 2001.
3. M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. of 12th ACM Conference on Computer and Communications Security*, 2005.
4. M. Baudet. YAPA (Yet Another Protocol Analyzer), 2008. `http://www.lsv.ens-cachan.fr/~baudet/yapa/index.html`.
5. M. Baudet, V. Cortier, and S. Delaune. YAPA: A generic tool for computing intruder knowledge. In *Proc. of 20th International Conference on Rewriting Techniques and Application (RTA'09)*, LNCS, 2009.
6. B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *Proc. of 20th International Conference on Automated Deduction (CADE'05)*, 2005.
7. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.
8. V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: symbolic equivalence of constraint systems. Technical report, `http://www.lsv.ens-cachan.fr/~cheval/programs/technical-report.pdf`, 2010.
9. Y. Chevalier and M. Rusinowitch. Decidability of symbolic equivalence of derivations. Unpublished draft, 2009.
10. Ş. Ciobâcă. KiSs, 2009. `http://www.lsv.ens-cachan.fr/~ciobaca/kiss`.
11. H. Comon-Lundh. Challenges in the automated verification of security protocols. In *Proc. of 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, volume 5195 of *LNAI*, pages 396–409, Sydney, Australia, 2008. Springer-Verlag.
12. H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties of cryptographic protocols. application to key cycles. *Transaction on Computational Logic*, 11(2), 2010.
13. R. Corin, J. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. *Electr. Notes Theor. Comput. Sci.*, 121:47–63, 2005.
14. V. Cortier and S. Delaune. A method for proving observational equivalence. In *Proc. of 22nd Computer Security Foundations Symposium (CSF'09)*, pages 266–276. IEEE Comp. Soc. Press, 2009.
15. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
16. U. Fendrup, H. Hüttel, and J. N. Jensen. Modal logics for cryptographic processes. *Theoretical Computer Science*, 68, 2002.
17. H. Huttel. Deciding framed bisimulation. In *4th International Workshop on Verification of Infinite State Systems INFINITY'02*, pages 1–20, 2002.
18. C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
19. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. of 8th ACM Conference on Computer and Communications Security*, 2001.
20. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. In *Proc. of 14th Computer Security Foundations Workshop*, 2001.
21. C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proc. of 16th Conference on Automated Deduction*, volume 1632, pages 314–328. LNCS, 1999.