Constraint solving techniques and enriching the model with equational theories

Hubert Comon-Lundh[1], Stéphanie Delaune[1] and Jonathan Millen[2]
[1]LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France
[2]MITRE Corporation

Abstract. Derivability constraints represent in a symbolic way the infinite set of possible executions of a finite protocol, in presence of an arbitrary active attacker. Solving a derivability constraint consists in computing a simplified representation of such executions, which is amenable to the verification of any (trace) security property. Our goal is to explain this method on a non-trivial combination of primitives.

In this chapter we explain how to model the protocol executions using derivability constraints, and how such constraints are interpreted, depending on the cryptographic primitives and the assumed attacker capabilities. Such capabilities are represented as a deduction system that has some specific properties. We choose as an example the combination of exclusive-or, symmetric encryption/decryption and pairing/unpairing. We explain the properties of the deduction system in this case and give a complete and terminating set of rules that solves derivability constraints. A similar set of rules has been already published for the classical Dolev-Yao attacker, but it is a new result for the combination of primitives that we consider. This allows to decide trace security properties for this combination of primitives and arbitrary finite protocols.

1. Introduction

A protocol specifies a set of *roles*, each of which is a template for a finite sequence of actions that send or receive messages. Each role may be instantiated any number of times; the instances are *sessions*. A *trace* of a protocol is a global sequence of actions that is an interleaving of a finite number of sessions. A protocol has many possible traces, depending on how many sessions there are, and on the particular interleaving. Even when such an ordering of actions is fixed, there are still many (actually infinitely many) possible instances of a trace with the same sequence of actions, because the intruder may affect the content of the messages by intercepting sent messages and forging received messages.

In this chapter, we introduce *derivability constraints*. Such constraints represent in a symbolic and compact way which trace instances are possible, when an interleaving of actions is fixed. Then the existence of an attack can be expressed as the satisfiability of the derivability constraints, together with the negation of the security goal. For instance, if the security goal is the confidentiality of some data s, then the protocol is secure if the derivability contraints, together with the derivability of s, is unsatisfiable. Hence, deciding the satisfiability of a derivability constraint (together with some other formula) yields, as a particular case, an alternative to the decision algorithm described in the previous chapter. We may however, in addition, consider other security properties, that can be expressed as properties of symbolic traces. Typical examples of such properties include agreement properties or timing properties.

Derivability constraints and their satisfiability were first introduced in [33]. Since then, the approach has been followed by many papers, which show the decidability of the problem in many different settings, depending on the cryptographic primitives and the supposed properties that they satisfy (*e.g.* exclusive-or [19,11], some algebraic properties of modular exponentiation [10,34,15,8], monoidal equational theories [24]).

In this chapter, we explain a method for simplifying derivability constraints when the security primitives consist in exclusive-or, symmetric encryption/decryption and pairing/unpairing. In principle, the same method can be applied to numerous other cryptographic primitives, adapting however the underlying set of simplification rules. The procedure that we describe in this chapter is actually a generalization of the known procedures for such primitives [19,11]: we provide with a constraint simplification algorithm that transforms a constraint into finitely many equivalent and simpler ones, called *solved forms*. This allows us not only to decide the existence of a solution, but also to represent all solutions. Such a feature is used in [16] for deciding trace properties such as authentication and key cycles in security protocols, and also in [26] for deciding game-theoretic security properties such as abuse-freeness. As far as we know, the result presented here is new. Some proofs that are not detailed in this chapter can be found in [18].

Finally, we claim that our decision procedure is simple: we only give a few transformation rules that are applied to constraints until they are solved. The difficult part is then the design of a complete and terminating strategy. In this introductory chapter, we do not try to get the best performance. There are many possible optimizations that we discuss only briefly. We prove, however, the correctness, completeness and termination of the constraint solving method, along the same lines as [16], but extending the primitives with an exclusive-or operator.

Before introducing the derivability constraints in Section 3, we discuss in Section 2 the intruder capabilities. The main idea of the constraint solving technique is to search for an intruder strategy, only considering strategies that are "optimal". In other words, an intruder may have several ways to compute a given message, some of which are simpler. Then, when solving the derivability constraints, we only look for the last step of an intruder's proof that is "optimal", until the constraint is solved.

Outline. In Section 2, we review the various ways of describing the intruder's capabilities. In Section 3, we introduce the constraint solving approach and its relationship with the security analysis of protocols. In Section 4, we give a more detailed exposition of the constraint solving method, in the case of symmetric encryption, pairing, and exclusive-or.

2. Intruder capabilities

2.1. Messages

In the formal setting that is considered in this chapter, messages are *terms* that are built from a set of *function symbols* $\mathcal{F}$. These function symbols allow us to represent cryptographic primitives. Here is a sampling of typical function symbols. We will not use all of them.

- pairing and projections: $\langle x, y\rangle$, $\pi_1(x)$, and $\pi_2(x)$;
- symmetric encryption/decryption: $\{|x|\}^{\mathsf{s}}_y$, and $\{|x|\}^{-\mathsf{s}}_y$;
- asymmetric encryption/decryption: $\{|x|\}^{\mathsf{a}}_y$, and $\{|x|\}^{-\mathsf{a}}_y$;
- private and public keys for asymmetric encryption: $\mathtt{dk}(x)$, and $\mathtt{pk}(x)$.
- signature and signature check: $[x]_y$, and $[x]^{-}_y$;
- signature key and verification key for signature: $\mathtt{sk}(x)$, and $\mathtt{vk}(x)$;
- hash function: $\mathtt{hash}(x)$;
- exclusive-or: $x \oplus y$;
- random numbers, symmetric keys: n, r, k, $\ldots$

The set of terms $\mathcal{T}(\mathcal{F})$ (or messages) is untyped in our definitions. A typed version can be encoded using tags and possibly additional function symbols and rules. We assume that typing and type-checking is performed in an explicit way, which we believe is the most conservative solution.

We may need to consider messages with unknown (arbitrary) parts; given a set of *variables* $\mathcal{X}$, the set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the set of terms built from $\mathcal{F}$ and the variables in $\mathcal{X}$. We denote by $vars(t)$ the set of variables that occurs in t. We also use *substitutions*. A substitution $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is the simultaneous replacement of x_i with t_i for every $1 \leq i \leq n$. We require that no x_i may occur in any t_j. We denote by $t\sigma$ the term that results from the application of the substitution σ to the term t. Occasionally it is convenient to regard σ as a conjunction of equations $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$. We denote by $\mathtt{top}(u)$ the top symbol of the term u, *i.e.* the function symbol that occurs at its root position.

2.2. Deductions

In a formal security analysis that follows the Dolev-Yao model [25], the intruder is assumed capable of intercepting all messages in the network, and deriving new messages from prior messages by decomposing and composing them. The ability of an intruder to create a message from others can be inferred either from relations in an equational theory or from deduction rules expressing possible intruder derivations. For example, the intruder may derive a from $\{|a|\}^{\mathsf{s}}_k$ and the key k either by noticing that $\{|\{|a|\}^{\mathsf{s}}_k|\}^{-\mathsf{s}}_k = a$ or by applying the deduction rule:

$$\frac{\{|x_1|\}^{\mathsf{s}}_{x_2} \qquad x_2}{x_1}$$

Deduction rules state that any instance of the conclusion can be computed by the intruder from a corresponding instance of the premisses.

Example 1 *A possible set of deduction rules for asymmetric encryption, signatures and pairing is described below. There are several small variants of these rules, which we do not describe here.*

$$\frac{x \quad y}{\langle x, y\rangle}\ (\mathsf{P}) \qquad \frac{\langle x, y\rangle}{x}\ (\mathsf{U}_1) \qquad \frac{\langle x, y\rangle}{y}\ (\mathsf{U}_2)$$

$$\frac{x \quad y}{\{\!|x|\!\}^{\mathsf{a}}_{y}}\ (\mathsf{AE}) \qquad \frac{\{\!|x|\!\}^{\mathsf{a}}_{\mathtt{pk}(y)} \quad \mathtt{dk}(y)}{x}\ (\mathsf{AD}) \qquad \frac{x}{\mathtt{pk}(x)}\ (\mathsf{PK})$$

$$\frac{x \quad y}{[x]_y}\ (\mathsf{S}) \qquad \frac{[x]_{\mathtt{sk}(y)} \quad \mathtt{vk}(y)}{x}\ (\mathsf{V}) \qquad \frac{x}{\mathtt{vk}(x)}\ (\mathsf{VK})$$

The balance between equations or deduction rules to specify intruder capabilities depends on the rest of the formalization, and on the algorithms that will be used in the analysis. There is no choice if the message algebra is a free algebra that has no destructors. In particular, in a free algebra, decryption $\{\!|x|\!\}^{-\mathsf{s}}_{y}$ and projection $\pi_i(\langle x, y\rangle)$ operations are not available, and intruder decomposition capabilities are necessarily modeled by deduction rules. The loss of generality entailed by the use of a free algebra, and the cure for it, is discussed in [32] and [30]. As we will see below, there are cryptographic primitives such as exclusive-or (see Example 2) that cannot be expressed fully and satisfactorily with deduction rules. We need to model them by the means of an equational theory. An equational theory E is a set of equations between terms. Given two terms u and v, we write $u =_{\mathsf{E}} v$ if the equation $u = v$ is a consequence of E.

Example 2 *The equational theory $\mathsf{E}_\oplus$ for the exclusive-or operator is defined by the following equations:*

$$\begin{array}{cc} x \oplus (y \oplus z) = (x \oplus y) \oplus z & x \oplus y = y \oplus x \\ x \oplus x = 0 & x \oplus 0 = x \end{array}$$

The symbol $\oplus$ is a binary function symbol whereas 0 is a constant. The two first equations modeled the associativity and commutativity properties of the $\oplus$ symbol whereas the two last ones modeled the nilpotency and the fact that 0 is a neutral element.

For instance, consider three distinct constant symbols a, b, and c. We have that $(a \oplus b) \oplus b =_{\mathsf{E}_\oplus} a$ whereas $a \oplus b \neq_{\mathsf{E}_\oplus} a \oplus c$.

Having decided upon the split between deduction rules and a set of equations E, we can define the *derivability relation*, denoted $T \vdash u$, that represents the intruder capability to derive a message u from a set of available messages T.

Definition 1 (derivability relation) *The* derivability relation $\vdash$ *is defined as the least relation such that, when T is a finite set of terms and s and t are terms, we have that:*

- *$T \vdash s$ when $s \in T$;*
- *$T \vdash s$ when there is a term t such that $t =_{\mathsf{E}} s$ and $T \vdash t$;*
- *$T \vdash s$ if there is a deduction rule $\frac{u_1 \quad \ldots \quad u_n}{u}$ and a substitution σ such that $s = u\sigma$ and $T \vdash u_i\sigma$ for every $i \in \{1, \ldots, n\}$.*

Example 3 *Let $T = \{\{\!|a|\!\}^{\mathsf{a}}_{\mathtt{pk}(b)}, \mathtt{vk}(b), [\mathtt{dk}(b)]_{\mathtt{sk}(b)}\}$. Using the rules given in Example 1, we may model the deduction of a from the set of terms T as follows:*

$$\frac{\{\!|a|\!\}^{\mathsf{a}}_{\mathtt{pk}(b)} \qquad \dfrac{[\mathtt{dk}(b)]_{\mathtt{sk}(b)} \qquad \mathtt{vk}(b)}{\mathtt{dk}(b)}}{a}$$

In many cases, equations like $\{\!|\{\!|x|\!\}^{\mathsf{s}}_y|\!\}^{-\mathsf{s}}_y = x$ can be oriented to form term rewriting rules, in this case $\{\!|\{\!|x|\!\}^{\mathsf{s}}_y|\!\}^{-\mathsf{s}}_y \to x$. If the resulting term rewriting system is *convergent*, any term u has a unique normal form $u\!\downarrow$, which is obtained by applying the rewriting rules in any order and as long as possible. In that case, $u\!\downarrow$ is a canonical representative of the equivalence class of u with respect to $=_{\mathsf{E}}$, *i.e.* $u =_{\mathsf{E}} v$ if, and only if, $u\!\downarrow\, = v\!\downarrow$. If, in addition, there is no premisse of any deduction rule that overlaps a left side of a rewriting rule in a non-trivial way, we may simply apply a normalization step after each deduction step.

In some cases, it is not possible to get a finite convergent rewriting system from the set of equations, nor to turn the equations into finitely many deduction rules. A typical example is the set of equations given in Example 2 and that allows one to model the exclusive-or operator. The associativity and commutativity properties of the $\oplus$ symbol prevent us from getting a convergent rewriting system. Usually, such symbols are considered as varyadic: we may write $u_1 \oplus \cdots \oplus u_n$, since the parentheses (and the ordering) on $u_1, \dots, u_n$ are irrelevant. Keeping such a flat representation is useful, since the AC properties consist only in rearranging the arguments of a $\oplus$ operator, without changing the structure of the message. This requires, however, an infinite (yet recursive) set of deduction rules, relying on an extended rewriting system.

From now on, we use only the deduction rules, rewrite rules, and equations displayed in Figure 1. For simplicity, we only keep symmetric encryption and pairing, and do not consider asymmetric encryption and signatures. Notice that some exclusive-or equations have been oriented into rewrite rules; this set of rules is convergent (modulo the equations): every term t has a unique normal form modulo associativity and commutativity, which we write $t\!\downarrow$. The set of equations then only consists of permutative equations (on flattened terms). We will omit the index E in $=_{\mathsf{E}}$, leaving implicit both the flattening and the possible permutation of arguments.

Example 4 *Let* $T = \{\{\!|a|\!\}^{\mathsf{s}}_{a\oplus b},\ a \oplus \{\!|c|\!\}^{\mathsf{s}}_b,\ b \oplus \{\!|c|\!\}^{\mathsf{s}}_b\}$. *We show that* $T \vdash c$, *using the rules described in Figure 1. First, we show that* $T \vdash \{\!|c|\!\}^{\mathsf{s}}_b$ *and* $T \vdash b$. *Indeed, the two derivations* π_1 *and* π_2 *described below are witnesses of these facts.*

$$\pi_1 = \left\{ \frac{a \oplus \{\!|c|\!\}^{\mathsf{s}}_b \qquad \dfrac{\{\!|a|\!\}^{\mathsf{s}}_{a\oplus b} \qquad \dfrac{a \oplus \{\!|c|\!\}^{\mathsf{s}}_b \qquad b \oplus \{\!|c|\!\}^{\mathsf{s}}_b}{a \oplus b}\ (\mathsf{XOR})}{a}\ (\mathsf{SD})}{\{\!|c|\!\}^{\mathsf{s}}_b}\ (\mathsf{XOR}) \right.$$

Deduction rules:

$$\frac{x_1 \quad \cdots \quad x_n}{(x_1 \oplus \cdots \oplus x_n)\!\downarrow}\ (\mathsf{XOR}) \quad \text{for any } n \in \mathbb{N}$$

$$\frac{x_1 \quad x_2}{\langle x_1, x_2\rangle}\ (\mathsf{P}) \qquad \frac{\langle x_1, x_2\rangle}{x_1}\ (\mathsf{U}_1) \qquad \frac{\langle x_1, x_2\rangle}{x_2}\ (\mathsf{U}_2)$$

$$\frac{x_1 \quad x_2}{\{\!|x_1|\!\}^{\mathsf{s}}_{x_2}}\ (\mathsf{SE}) \qquad \frac{\{\!|x_1|\!\}^{\mathsf{s}}_{x_2} \quad x_2}{x_1}\ (\mathsf{SD})$$

Rewrite rules: $\quad x \oplus x \oplus y \ \to\ y \qquad x \oplus x \ \to\ 0 \qquad x \oplus 0 \ \to\ x$

Equations: $x_1 \oplus \cdots \oplus x_n = x_{\tau(1)} \oplus \cdots \oplus x_{\tau(n)}$ for any $n \in \mathbb{N}$ and any permutation τ.

Figure 1. Deduction rules, rewriting rules, and equations for encryption, pairing, and exclusive-or.

$$\pi_2 = \left\{ \dfrac{\dfrac{\{\!|a|\!\}^{\mathsf{s}}_{a\oplus b} \qquad \dfrac{a \oplus \{\!|c|\!\}^{\mathsf{s}}_b \quad b \oplus \{\!|c|\!\}^{\mathsf{s}}_b}{a \oplus b}\ (\mathsf{XOR})}{a}\ (\mathsf{SD}) \qquad \dfrac{a \oplus \{\!|c|\!\}^{\mathsf{s}}_b \quad b \oplus \{\!|c|\!\}^{\mathsf{s}}_b}{a \oplus b}\ (\mathsf{XOR})}{b}\ (\mathsf{XOR}) \right.$$

Now, it is easy to see that $T \vdash c$.

2.3. Proofs

The intruder's deductions are represented as tree proofs, as in the previous example. We formalize these notions here.

Definition 2 (proof) *A* proof π *with respect to a set of deduction rules $\mathcal{I}$ (and a convergent rewriting system $\mathcal{R}$) is a tree whose nodes are labeled with terms and such that, if a node is labeled with a term t and its sons are labeled with terms $t_1, \ldots, t_n$, then there is a deduction rule $\dfrac{s_1, \ldots, s_n}{s} \in \mathcal{I}$ and a substitution σ such that $s_i\sigma = t_i$ for every $1 \leq i \leq n$ and $s\sigma\!\downarrow = t$.*

The *hypotheses* $\mathsf{hyp}(\pi)$ of a proof π are the labels of the leaves of π. Its *conclusion* $\mathsf{conc}(\pi)$ is the label of the root of π. The last deduction rule $\mathsf{last}(\pi)$ is the instance of the deduction rule that yields the root. We say that π is a proof of $T \vdash u$ if $\mathsf{hyp}(\pi) \subseteq T$ and $\mathsf{conc}(\pi) = u$. Finally, $\mathsf{step}(\pi)$ is the set of all labels of π. A *subproof* of π is a subtree of π. It is also a proof.

Example 5 *In Example 4, the proof π_1 is such that $\mathsf{step}(\pi_1) = T \cup \{a \oplus b, a, \{\!|c|\!\}^{\mathsf{s}}_b\}$, $\mathsf{conc}(\pi_1) = \{\!|c|\!\}^{\mathsf{s}}_b$, $\mathsf{hyp}(\pi_1) = T = \{\{\!|a|\!\}^{\mathsf{s}}_{a\oplus b},\ a \oplus \{\!|c|\!\}^{\mathsf{s}}_b,\ b \oplus \{\!|c|\!\}^{\mathsf{s}}_b\}$, and $\mathsf{last}(\pi_1)$ is an instance of the* XOR *deduction rule. More precisely, we have that:*

$$\mathsf{last}(\pi_1) = \frac{a \oplus \{\!|c|\!\}^{\mathsf{s}}_b \qquad a}{\{\!|c|\!\}^{\mathsf{s}}_b}\ (\mathsf{XOR}).$$

The instances of the deduction rules (U_1), (U_2), and (SD) and instances of the (XOR) for which the conclusion is not headed with $\oplus$ are called *decompositions*. More generally, an instance of a deduction rule is a decomposition if its conclusion is a subterm of the premisses and is irrelevant in the rewriting/equational steps. By convention, if π is reduced to a leaf, then we also say that $\mathsf{last}(\pi)$ is a decomposition.

Example 6 *The following are instances of deduction rules, that are decompositions:*

$$\frac{\{\!|a \oplus b|\!\}^{\mathsf{s}}_{c} \quad c}{a \oplus b}\ (\mathsf{SD}) \qquad\qquad \frac{\{\!|a|\!\}^{\mathsf{s}}_{b} \oplus b \quad b \oplus c \quad c}{\{\!|a|\!\}^{\mathsf{s}}_{b}}\ (\mathsf{XOR})$$

while the following are instances of deduction rules, that are not decompositions:

$$\frac{a \oplus b \quad b \oplus c}{a \oplus c}\ (\mathsf{XOR}) \qquad\qquad \frac{\{\!|a \oplus b|\!\}^{\mathsf{s}}_{k} \oplus a \quad \{\!|a \oplus b|\!\}^{\mathsf{s}}_{k} \oplus b}{a \oplus b}\ (\mathsf{XOR})$$

An instance of a deduction rule is a *composition* if its premisses are subterms of the conclusion. Typical examples of compositions are all the instances of the deduction rules (SE) and (P). Note that some instances of the deduction rule (XOR) are compositions, and some others are never compositions, nor decompositions.

Example 7 *Consider the following three instances of the deduction rule* (XOR)*:*

$$\frac{a \oplus b \quad b}{a} \qquad\qquad \frac{a \quad b}{a \oplus b} \qquad\qquad \frac{a \oplus b \quad b \oplus c}{a \oplus c}$$

The first instance is a decomposition, the second instance is a composition whereas the third one is neither a composition nor a decomposition.

2.4. Normal proofs

Typical results concerning the deduction rules show that if a term can be derived then there is a *normal proof* of it. The notion of normal proof will depend on the intruder deduction system, but it has the property to avoid any unnecessary detour. Normal proofs allow us to restrict the search space when looking for an attack.

We define below the normal proofs for the deduction system given in Figure 1. We simplify the proofs according to the rules presented in Figure 2. These rules simply gather together successive (XOR) deduction rules and otherwise, they only remove useless parts of the proofs. They are (strongly) terminating: a *normal proof* is a proof, that is irreducible with respect to the rules of Figure 2. For other equational axioms or inference rules, there are also similar simplification and gathering rules [24]. There is however no general procedure that yields an appropriate notion of normal proofs for arbitrary equational theories.

"Locality" is a subformula property that holds on normal proofs. In the definition that follows, $\mathsf{St}(T)$ is the set of subterms of T. Let $u = u_1 \oplus \ldots \oplus u_n$ be a term such that $\mathsf{top}(u_i) \neq \oplus$ for every $i \in \{1, \ldots, n\}$. Then, the strict direct subterms of u, also called the *factors* of u and denoted $\mathsf{fact}(u)$, are the individual arguments u_i only.

Definition 3 (local proof) *A* local proof π *of* $T \vdash u$ *is a proof in which*

$$\dfrac{u_1 \cdots u_i \; u \; u_{i+1} \cdots u_j \; u \; u_{j+1} \cdots u_n}{v}\ (\mathsf{XOR}) \quad\Rightarrow\quad \dfrac{u_1 \cdots u_i \; u_{i+1} \cdots u_j \; u_{j+1} \cdots u_n}{v}\ (\mathsf{XOR})$$

$$\dfrac{u_1 \cdots u_i \qquad \dfrac{v_1 \cdots v_m}{v}\ (\mathsf{XOR}) \qquad u_{i+1} \cdots u_n}{u}\ (\mathsf{XOR}) \quad\Rightarrow\quad \dfrac{u_1 \cdots u_i \; v_1 \cdots v_m \; u_{i+1} \cdots u_n}{u}\ (\mathsf{XOR})$$

$$\dfrac{\dfrac{\pi_1}{u_1}\ \cdots\ \dfrac{\pi_i}{u_i}\ \ \dfrac{\pi}{u}\ \ \dfrac{\pi_{i+1}}{u_{i+1}}\ \cdots\ \dfrac{\pi_n}{u_n}}{u}\ (\mathsf{XOR}) \quad\Rightarrow\quad \dfrac{\pi}{u}$$

$$\dfrac{\dfrac{\dfrac{\pi}{u} \quad \dfrac{\pi_1}{u_1}}{\langle u, u_1 \rangle}\ (\mathsf{P})}{u}\ (\mathsf{U}_1) \quad\Rightarrow\quad \dfrac{\pi}{u} \qquad\qquad \dfrac{\dfrac{\dfrac{\pi_1}{u_1} \quad \dfrac{\pi}{u}}{\langle u_1, u \rangle}\ (\mathsf{P})}{u}\ (\mathsf{U}_2) \quad\Rightarrow\quad \dfrac{\pi}{u}$$

$$\dfrac{\dfrac{\dfrac{\pi}{u} \quad \dfrac{\pi_1}{u_1}}{\{\!|u|\!\}^{\mathsf{s}}_{u_1}}\ (\mathsf{SE}) \qquad \dfrac{\pi_1'}{u_1}}{u}\ (\mathsf{SD}) \quad\Rightarrow\quad \dfrac{\pi}{u}$$

Figure 2. Proof normalization rules

- *either* $\mathsf{last}(\pi)$ *is a decomposition and* $\mathsf{step}(\pi) \subseteq \mathsf{St}(T)$
- *or else* $\mathsf{step}(\pi) \subseteq \mathsf{St}(T \cup \{u\})$.

This general property of proof systems is ensured by our proof normalization process:

Lemma 1 *If π is a normal proof of $T \vdash u$ then π is a local proof of $T \vdash u$.*

Proof:
Let π be a normal proof of $T \vdash u$. Let us prove that π is local by induction on the size of π, *i.e.* its number of nodes.

Base case: If π is reduced to a leaf, then $u \in T$ and π is a local proof of $T \vdash u$.

Induction step: We distinguish two cases depending on whether $\mathsf{last}(\pi)$ is a decomposition or not.

1. If $\mathsf{last}(\pi)$ is not a decomposition, either $\mathsf{hyp}(\mathsf{last}(\pi))$ are subterms of its conclusion (*e.g.* an instance of (SE) or (P)) in which case we can simply use the induction hypothesis, or else we have that

$$\pi = \left\{ \frac{\pi_1 \quad \cdots \quad \pi_n}{u} \ (\mathsf{XOR}) \right.$$

with $\mathsf{conc}(\pi_i) \notin \mathsf{St}(u)$ for some i.
Let $u_j = \mathsf{conc}(\pi_j)$ for $j \in \{1, \dots, n\}$. We have that $u = (u_1 \oplus \cdots \oplus u_n)\downarrow$. By proof normalization, for every j, either $\mathsf{last}(\pi_j)$ is not a decomposition, and then $\mathsf{top}(u_j) \neq \oplus$ or else $\mathsf{last}(\pi_j)$ is a decomposition and, by induction hypothesis, $\mathsf{step}(\pi_j) \subseteq \mathsf{St}(T)$. Consider an index k such that u_k is maximal (with respect to the subterm relation) in the set $\{u_1, \dots, u_n\}$.
If $\mathsf{last}(\pi_k)$ is not a decomposition, then $\mathsf{top}(u_k) \neq \oplus$. Furthermore, thanks to the rewriting rules for $\oplus$, we are in one of the following cases:

   - u_k is a strict subterm of some u_j. This is ruled out by the maximality assumption on u_k.
   - $u_k = u_j$ for some $j \neq k$. This is ruled out by the proof normalization rules.
   - $u_k = u$. This is ruled out by the proof normalisation rules
   - $u_k \in \mathsf{fact}(u\downarrow)$ is a strict subterm of u (*i.e.*, it does not disappear in the normalisation of $u_1 \oplus \ldots \oplus u_n$).

   Since only the last case is possible, every maximal term in $\{u_1, \dots, u_n\}$, that is not obtained by a decomposition, is a strict subterm of u and therefore, by induction hypothesis, $\mathsf{step}(\pi_k) \subseteq \mathsf{St}(u_k) \cup \mathsf{St}(T) \subseteq \mathsf{St}(u) \cup \mathsf{St}(T)$.
It follows that for every maximal term in $\{u_1, \dots, u_n\}$, we have that $\mathsf{step}(\pi_k) \subseteq \mathsf{St}(u) \cup \mathsf{St}(T)$. Then, for any term u_i, there is a maximal term u_k such that $u_i \in \mathsf{St}(u_k)$ and therefore $\mathsf{St}(u_i) \cup \mathsf{St}(T) \subseteq \mathsf{St}(u_k) \cup \mathsf{St}(T) \subseteq \mathsf{St}(u) \cup \mathsf{St}(T)$.
It follows that, for every i, $\mathsf{step}(\pi_i) \subseteq \mathsf{St}(u) \cup \mathsf{St}(T)$, hence $\mathsf{step}(\pi) \subseteq \mathsf{St}(u) \cup \mathsf{St}(T)$.
2. Assume now that $\mathsf{last}(\pi)$ is a decomposition. We consider all possible rules for $\mathsf{last}(\pi)$.

   Case 1: The proof π ends with an instance of (U_1) (or (U_2)), *i.e.*

$$\pi = \left\{ \frac{\pi_1}{u} \ (\mathsf{U_i}) \right.$$

   with $\mathsf{conc}(\pi_1)$ is either a pair $\langle u, v\rangle$ or a pair $\langle v, u\rangle$. In both cases, in order to get a term whose top symbol is a pairing, $\mathsf{last}(\pi_1)$ must be either a pairing rule or a decomposition rule. The first case is ruled out since π is a normal proof. Hence we can apply the induction hypothesis and conclude that $\mathsf{step}(\pi_1) \subseteq \mathsf{St}(T)$. It follows that $\mathsf{step}(\pi) \subseteq \mathsf{St}(T)$.

   Case 2: The proof π ends with an instance of (SD), *i.e.*

$$\pi = \left\{ \frac{\pi_1 \quad \pi_2}{u} \ (\mathsf{SD}) \right.$$

   with $\mathsf{conc}(\pi_1) = \{\!|u|\!\}^{\mathsf{s}}_v$, $\mathsf{conc}(\pi_2) = v$. Since $\mathsf{conc}(\pi_1)$ is headed with an encryption symbol, $\mathsf{last}(\pi_1)$ must be either an instance of (SE) or a decomposition. The first case is ruled out since π is a normal proof, hence $\mathsf{last}(\pi_1)$ is a decomposition. By induction hypothesis $\mathsf{step}(\pi_1) \subseteq \mathsf{St}(T)$. In particular, we have that $v \in \mathsf{St}(\mathsf{conc}(\pi_1)) \subseteq \mathsf{St}(T)$. Now, by induction hypothesis, we have that $\mathsf{step}(\pi_2) \subseteq \mathsf{St}(T \cup \{v\}) \subseteq \mathsf{St}(T)$. It follows that $\mathsf{step}(\pi) \subseteq \mathsf{St}(T)$.

Case 3: The proof π ends with an instance of (XOR), *i.e.*

$$\pi = \left\{ \frac{\pi_1 \quad \cdots \quad \pi_n}{u} \ (\mathsf{XOR}) \right.$$

Let $u_j = \mathsf{conc}(\pi_j)$ for every $j \in \{1, \ldots, n\}$. By hypothesis, we know that $\mathsf{last}(\pi)$ is a decomposition, thus $\mathsf{top}(u) \neq \oplus$ and $u \in \mathsf{fact}(u_j)$ for some $j \in \{1, \ldots, n\}$.
For every j, $\mathsf{last}(\pi_j)$ cannot be an instance of the (XOR) deduction rule, because π is a normal proof. Therefore, if $\mathsf{last}(\pi_j)$ is not a decomposition, then $\mathsf{top}(u_j) \neq \oplus$. It must then be a subterm of some u_k, $k \neq j$ (actually a strict subterm since π is a normal proof). Thus, the maximal terms in $\{u_1, \ldots, u_n\}$ with respect to the subterm relation, are terms u_j, such that $\mathsf{last}(\pi_j)$ is a decomposition. By induction hypothesis, it follows that, for every $j \in \{1, \ldots, n\}$, we have that $\mathsf{step}(\pi_j) \subseteq \mathsf{St}(T)$ and therefore we have that $\mathsf{step}(\pi) \subseteq \mathsf{St}(T)$.

□

As a consequence, the derivation problem, *i.e.* given T and u, the problem of deciding whether u is derivable from T or not, is in PTIME. We may indeed run a fixed point computation algorithm on the set $\mathsf{St}(T \cup \{u\})$. It will terminate in polynomial time, as long as the one-step deducibility relation is decidable in PTIME, which is the case for our rules in Figure 1, as well as in many other cases (see [24]). There are several other such results for some classes of equational theories (*e.g.* [24,9]). An overview of equational properties that are relevant to security protocols is given in [21].

3. Derivation constraints: definitions and examples

3.1. Introduction with an example

Consider the flawed handshake protocol example described in Figure 3, temporarily returning to a larger set of operations. We want to check, for a fixed number of sessions (let us call this the *bounded scenario* case), whether or not there is an attack on the protocol that compromises the secret s, which the participant B generates and wishes to share with the participant A. This security property turns out to be a trace property, so that we can check it when we know which protocol traces are possible.

In the bounded scenario case, the number of messages that are exchanged is bounded. However, as explained in Chapter **??**, there are still an infinite number of possible message instances. Hence the verification cannot be performed by a simple enumeration of scenarii. The solution is to use a symbolic representation of sequences of messages, treating uninstantiated parameters as variables.

Let us show how it works on the handshake protocol described in Figure 3, when there is one instance of the role A (with agent parameters a and c) and one instance of

$$\begin{tikzcd} A \arrow[r, "\{\![k]_{\mathtt{sk}(A)}\}\!\}^{\mathsf{a}}_{\mathtt{pk}(B)}"] \arrow[d, dash] & B \arrow[d, dash] \\ \circ & \circ \arrow[l, "\{\!|s|\!\}^{\mathsf{s}}_{k}"'] \end{tikzcd}$$

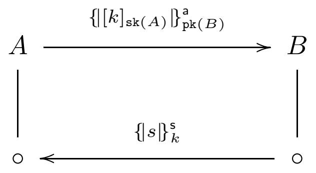

Figure 3. Flawed handshake protocol

the role B (with agent parameters b and a). The *roles* are simply threads, in which some names are hidden: this corresponds to local random generation. To be more precise, we consider two threads expressed in a spi-calculus style, with pattern matching:

$$A(a,c) : \nu k.\ \mathtt{send}(\{\!|[k]_{\mathtt{sk}(a)}|\!\}^{\mathsf{a}}_{\mathtt{pk}(c)}).\ \mathtt{get}(\{\!|x|\!\}^{\mathsf{s}}_{k}).$$
$$B(b,a) : \nu s.\ \mathtt{get}(\{\!|[z]_{\mathtt{sk}(a)}|\!\}^{\mathsf{a}}_{\mathtt{pk}(b)}).\ \mathtt{send}(\{\!|s|\!\}^{\mathsf{s}}_{z}).$$

Note that, in this scenario, the name a occurs free in both $A(a,c)$ and in $B(b,a)$. It cannot be renamed in any of the two threads: the name is *shared* by these threads. Let us see informally what the thread $A(a,c)$ does in more detail. First it has a, c as parameters: it is a program executed on behalf of an agent a, who wishes to communicate with c. Given a and c, it generates a fresh key k (the νk construction). Then it constructs the first encrypted message and sends it. Next, it waits for some message containing the encryption of some value x with k. When the message is received, it tries to decrypt it with k. If this succeeds, then it retrieves the message x. Otherwise, if the decryption fails, the program aborts (this is not specified here). The thread $B(b,a)$ works in a similar way: first it waits for a message encrypted with its own public key, and extracts a value z signed by a. If the signature checks, it sends back the nonce (secret) s encrypted with the (supposed) key z.

Then, these roles are composed, using parallel composition, name hiding and possibly replication, to build *scenarii*. For instance, $A(a,c) \| B(b,a)$ is a scenario where there is one instance of the role A (with agent parameters a and c) and one instance of the role B (with agent parameters b and a).

The operational semantics of such threads is described (in slightly different flavors) in several papers such as [1,39,37]. The important features are that each of these threads is executed concurrently in a hostile environment: messages that are sent over the network can be intercepted and modified by an intruder. Therefore, what an agent gets is the message that has been forged by the intruder from his current knowledge and the messages that circulated on the network. It is out of the scope of this chapter to define a precise operational semantics of such a process algebra (see Chapter **??**). It is not always sufficient to consider a small system consisting of only one thread per role, for security analysis. Conditions under which such small systems are sufficient are given by Lowe and others [29,35,2].

In this scenario, we assume that the keys $\mathtt{pk}(a), \mathtt{pk}(b), \mathtt{pk}(c), \mathtt{vk}(a), \mathtt{vk}(b), \mathtt{vk}(c)$ are public, hence available to all agents, including the intruder. The other keys are private: they are normally held by their owners only. In this example, we suppose that c is compromised: this means that the secret keys $\mathtt{dk}(c)$ and $\mathtt{sk}(c)$ are also available to the intruder. Finally, the intruder knows the names of the agents, namely a, b and c. Thus, the total initial intruder knowledge is the set of terms:

$$T_{\mathsf{init}} = \{\mathtt{pk}(a), \mathtt{pk}(b), \mathtt{pk}(c), \mathtt{vk}(a), \mathtt{vk}(b), \mathtt{vk}(c), \mathtt{dk}(c), \mathtt{sk}(c), a, b, c\}.$$

The next step is to construct the parallel composition of the two sessions. The parallel composition has no more than six possible traces, which are the various interleavings of the two ordered message actions of each of the two sessions. In general, if there are two sessions of length m and n, respectively, the number of interleavings is the number of combinations $C(m+n, m)$. However, not all of the interleavings are *solvable*. A trace is solvable if, and only if, there is a substitution for its variables such that the message u in each $\mathtt{get}(u)$ action is derivable by the intruder from the intruder's initial knowledge plus the messages that have been sent in prior $\mathtt{send}$ actions. This condition generates a *derivation constraint* for each received message.

Coming back to our running example, one of the interleaving is:

$$\mathtt{send}(\{\![k]_{\mathtt{sk}(a)}]\!\}^{\mathsf{a}}_{\mathtt{pk}(c)}), \mathtt{get}(\{\![z]_{\mathtt{sk}(a)}]\!\}^{\mathsf{a}}_{\mathtt{pk}(b)}), \mathtt{send}(\{\!|s|\!\}^{\mathsf{s}}_{z}), \mathtt{get}(\{\!|x|\!\}^{\mathsf{s}}_{k})$$

Is it solvable? There are two $\mathtt{get}$ actions, hence two derivation constraints:

$$\mathcal{C} = \begin{cases} T_{\mathsf{init}},\ \{\![k]_{\mathtt{sk}(a)}]\!\}^{\mathsf{a}}_{\mathtt{pk}(c)} \overset{?}{\vdash} \{\![z]_{\mathtt{sk}(a)}]\!\}^{\mathsf{a}}_{\mathtt{pk}(b)} \\ T_{\mathsf{init}},\ \{\![k]_{\mathtt{sk}(a)}]\!\}^{\mathsf{a}}_{\mathtt{pk}(c)},\ \{\!|s|\!\}^{\mathsf{s}}_{z} \overset{?}{\vdash} \{\!|x|\!\}^{\mathsf{s}}_{k} \end{cases}$$

Note that the set of terms on the left of each derivation constraint is actually a union $T_{\mathsf{init}} \cup ...$, but it simplifies the notation to write these sets as comma-separated lists. The difference between a derivation constraint, denoted by $T \overset{?}{\vdash} u$, and a derivation, denoted by $T \vdash u$, is that the constraint asks for a substitution σ such that $T\sigma \vdash u\sigma$.

Example 8 *The constraints in $\mathcal{C}$ are simultaneously satisfied by the substitution*

$$\sigma = \{z \mapsto k,\ x \mapsto s\}.$$

The fact that the second constraint is satisfied is obvious. Indeed, a normal proof of $T_{\mathsf{init}},\ \{\![k]_{\mathtt{sk}(a)}]\!\}^{\mathsf{a}}_{\mathtt{pk}(c)},\ \{\!|s|\!\}^{\mathsf{s}}_{k} \vdash \{\!|s|\!\}^{\mathsf{s}}_{k}$ is actually reduced to a leaf. The fact that the first constraint is satisfied takes some effort to see. A normal proof witnessing this fact is described below.

$$\dfrac{\dfrac{\{\![k]_{\mathtt{sk}(a)}]\!\}^{\mathsf{a}}_{\mathtt{pk}(c)} \qquad \mathtt{sk}(c)}{[k]_{\mathtt{sk}(a)}} \qquad \mathtt{pk}(b)}{\{\![k]_{\mathtt{sk}(a)}]\!\}^{\mathsf{a}}_{\mathtt{pk}(b)}}$$

We still have to ask whether this trace violates the security goal of the protocol. It does, because the intruder can obtain k from $[k]_{\mathtt{sk}(a)}$ and then s from $\{\!|x|\!\}^{\mathsf{s}}_{k}$. There is a trick to help us figure this out: we add a role $C(s) = \mathtt{get}(s)$ to the protocol and the scenario. An artificial role of this kind, introduced to test a secrecy property, is called a *listener*. This means that the action $\mathtt{get}(s)$ must be included in each trace. This causes a new constraint to be generated, of the form $T_{\mathsf{init}}, ... \overset{?}{\vdash} s$.

A substitution satisfying all the constraints, including this one, will demonstrate that the secret s can be read by the intruder. This reduces detection of secrecy violations to the basic constraint solution problem.

Before proceeding with the method for solving constraints, we remark that there are some easy optimizations that relieve us from considering every possible trace. If both of the traces ....`get`(u). `send`(u').... and ....`send`(u'). `get`(u).... are possible, we can discard the first one, because any substitution that satisfies the constraints arising from the first will also satisfy the second. Also, if a trace has consecutive `send` actions, their order is irrelevant, so one particular order is sufficient. This means that, when counting (or generating) interleavings, one need only consider the number and order of `get` actions in each session. By a similar argument, a listener action may always be placed last in a trace.

The basic constraint solving approach, therefore, has two stages: the first is to generate a sufficient set of traces, and the second is to generate and attempt to solve the constraint system arising from each trace. An alternative approach is used in [20]: while generating traces, attempt to solve the current partial constraint system each time a `get` action is added to a trace. If there is no solution, the partial trace need not be extended, eliminating a subtree of trace continuations. This approach can lead to more solution attempts in the worst case, but in practice it usually saves considerable time. Constraint differentiation [4] offers another kind of optimization.

3.2. Derivation constraints

We can now formally define the notion of a derivation constraint system, which is not specific to our set of primitives/deduction rules.

Definition 4 (derivation constraint system) *A* derivation constraint system $\mathcal{C}$ *consists of a conjunction of equations and a sequence of derivation constraints:*

$$T_1 \overset{?}{\vdash} u_1 \wedge \cdots \wedge T_n \overset{?}{\vdash} u_n \wedge s_1 \overset{?}{=} t_1 \wedge \ldots \wedge s_m \overset{?}{=} t_m$$

where $T_1, \ldots, T_n$ are finite sets of terms. Moreover, we assume that:

- Monotonicity: $\emptyset \subsetneq T_1 \subseteq T_2 \cdots \subseteq T_n$,
- Determinacy: *for every $i \in \{1, \ldots, n\}$, for every $u \in T_i$, we have that $\{u_1, \ldots, u_{i-1}\} \cup \mathcal{P} \vdash u$, where $\mathcal{P}$ is a finite set of ground terms.*

We just saw, with the handshake protocol, how a sequence of derivation constraints arises from a candidate trace of a protocol scenario. A constraint sequence arising this way always satisfies the two properties mentioned above, for some finite set $\mathcal{P}$ of ground terms, that represents the initial knowledge of the agents.

The first condition states that the intruder knowledge is increasing along the protocol execution. The second condition states that the message u, which is emitted at some stage, can be computed from all data that are possibly available at this stage. We do not commit here to any particular program that computes u, but only state that such a program must exist, if we assume that the corresponding agent has access to all data. This is a reasonable assumption, that only rules out non-deterministic choices and possibly some encodings of complex operations.

Let us remark that the monotonicity and determinacy conditions imply two important properties:

- *Origination:* for every $i \in \{1, \ldots, n\}$, for every $x \in \mathit{vars}(T_i)$, there must be a $j < i$ such that $x \in \mathit{vars}(u_j)$. In particular, we have that $\mathit{vars}(T_1) = \emptyset$. Indeed, if $x \notin \{u_1, \cdots, u_{i-1}\}$, then $x \notin \mathit{vars}(u)$ for any term u that can be deduced from $\{u_1, \cdots, u_{i-1}\} \cup \mathcal{P}$.
- *Stability by instantiation:* If $\mathcal{C}$ is a derivation constraint system, then, for every substitution σ, we have that $\mathcal{C}\sigma$ is also a derivation constraint system.

These two properties are actually what we need, and they could replace the determinacy property, yielding a more general (yet less natural) definition of constraint systems (see also [34]).

Example 9 *Consider the following sets of derivation constraints:*

$$\begin{aligned}
\mathcal{S}_1 &= a \overset{?}{\vdash} x \ \wedge \ b \overset{?}{\vdash} y \\
\mathcal{S}_2 &= a,\, b \overset{?}{\vdash} x \ \wedge \ a,\, b,\, \{\!|y|\!\}^{\mathsf{s}}_a \overset{?}{\vdash} x \ \wedge \ a,\, b,\, \{\!|y|\!\}^{\mathsf{s}}_a, x \overset{?}{\vdash} y \\
\mathcal{S}_3 &= a,\, b \overset{?}{\vdash} \{\!|x|\!\}^{\mathsf{s}}_y \ \wedge \ a,\, b,\, x \overset{?}{\vdash} y \\
\mathcal{S}_4 &= a,\, b \overset{?}{\vdash} \{\!|\langle x, y\rangle|\!\}^{\mathsf{s}}_k \ \wedge \ a,\, b,\, x \overset{?}{\vdash} y
\end{aligned}$$

The set $\mathcal{S}_1$ is not a derivation constraint system since it violates the monotonicity property. The set $\mathcal{S}_2$ is not a derivation constraint system, since it violates determinacy (actually, it violates origination). The set $\mathcal{S}_3$ is not a derivation constraint system since x is not derivable from $\{\!|x|\!\}^{\mathsf{s}}_y \cup \mathcal{P}$ for any set of ground terms $\mathcal{P}$. Intuitively, the agent receiving $\{\!|x|\!\}^{\mathsf{s}}_y$ cannot retrieve y and cannot retrieve x without having the key. This set of derivation constraints could still be handled since it satisfies the origination and any instance of it also satisfies the origination. Lastly, $\mathcal{S}_4$ is a derivation constraint system. In particular, we have that $\{\!|\langle x, y\rangle|\!\}^{\mathsf{s}}_k, k \vdash x$.

In addition to these constructions, we may need to introduce new variables along with constraint simplifications. In order to keep the set of solutions, we may add existential quantifiers for these new variables, in front of a derivation constraint system.

Actually, we may consider some additional formulas with only slight modifications of the constraint solving technique. Typically, we may wish to add disequalities, either because our threads contain conditionals or because we wish to express agreement properties. The constraint solving algorithm is not modified, until the end; we only need to add a simple test as a last step (see [16]). Similarly, membership constraints (expressing for instance typing information) or timing constraints can be added with minor changes.

Definition 5 (solution) *Given a convergent set of rewrite rules (possibly modulo associativity and commutativity) and a set of derivation rules that describe the intruder capabilities (as in Figure 1), a* solution *of a set of derivation constraints $\mathcal{C} = T_1 \overset{?}{\vdash} u_1 \wedge \cdots \wedge T_n \overset{?}{\vdash} u_n$, together with a set of equations $\mathcal{E}$, is a substitution σ of its free variables such that, for every i, $T_i\sigma\!\downarrow\ \vdash u_i\sigma\!\downarrow$ and, for every equation $u \overset{?}{=} v$ in $\mathcal{E}$, we have that $u\sigma\!\downarrow = v\sigma\!\downarrow$.*

Example 10 *Consider the following derivation constraint system (with no equation):*

$$\mathcal{C} = \begin{cases} a \oplus \{\!|c|\!\}^{\mathsf{s}}_{b},\ b \oplus \{\!|c|\!\}^{\mathsf{s}}_{b} \overset{?}{\vdash} x \\ \{\!|a|\!\}^{\mathsf{s}}_{x},\ a \oplus \{\!|c|\!\}^{\mathsf{s}}_{b},\ b \oplus \{\!|c|\!\}^{\mathsf{s}}_{b} \overset{?}{\vdash} c \end{cases}$$

The substitution $\{x \mapsto a \oplus b\}$ is a solution. A witness of the fact that σ satisfies the last derivation constraint is given in Example 4.

3.3. Solved constraints

A variable x is *solved* in a derivation constraint system $\mathcal{C}$ if it occurs as a member of an equation $x \overset{?}{=} u$, and nowhere else in the derivation constraint system. A variable x is *pre-solved* if it occurs in a derivation constraint $T \overset{?}{\vdash} x$ such that $x \notin vars(T)$ and does not occur in any term u such that $T' \overset{?}{\vdash} u$ is a derivation constraint such that $T' \subsetneq T$. A derivation constraint system $\mathcal{C}$ is *solved* if it is of the form

$$z_1 \overset{?}{=} t_1 \wedge \ldots \wedge z_m \overset{?}{=} t_m \wedge T_1 \overset{?}{\vdash} x_1 \wedge \ldots \wedge T_n \overset{?}{\vdash} x_n$$

where $z_1, \ldots, z_m$ are solved variables and $x_1, \ldots, x_n$ are pre-solved variables. In particular, this implies that $x_1, \ldots, x_n$ are distinct variables.

Lemma 2 *A solved constraint system has always at least one solution (actually, infinitely many solutions if it is not a trivial system).*

Proof:
Let $T_1 \overset{?}{\vdash} x_1, \ldots T_n \overset{?}{\vdash} x_n$ be the derivation constraints that occur in the system, and assume that $T_1 \subseteq \ldots \subseteq T_n$. We construct a solution by induction on n.

Base case: $n = 0$. In such a case, the equational part defines a substitution that is a solution of the constraint.

Induction step: If $n \geq 1$, first choose $x_1\sigma = t_1 \in T_1$ and replace x_1 with t_1 in the remainder of the constraint. After removing the constraint $T_1 \overset{?}{\vdash} t_1$, the resulting system is still a derivation constraint system and it is still solved (since the variables x_i are distinct). Then we apply the induction hypothesis. □

This lemma can be easily extended, considering additional disequality (resp. membership) constraints [16]. For instance, if we wish to decide a security property that can be expressed with equalities (typically an agreement property), then, as explained in introduction, we consider the derivation constraint system, together with the negation of these equalities. It is satisfiable if and only if the security property can be violated. If the derivation constraint is in solved form and there is no trivial disequality, then the above lemma implies that there is a solution, hence the security property is violated. In other words, it suffices to simplify the derivation constraints into solved forms (which may yield some replacements of the variables in the disequality part) and then check the satisfiability of the disequality part.

$$
\begin{array}{llll}
 & & \mathsf{R}_3: & T \overset{?}{\vdash} u \rightsquigarrow \exists x.\ T \overset{?}{\vdash} \langle x, u\rangle \\
\mathsf{R}_1: & T \overset{?}{\vdash} \langle u_1, u_2\rangle \rightsquigarrow T \overset{?}{\vdash} u_1 \wedge T \overset{?}{\vdash} u_2 & \mathsf{R}_4: & T \overset{?}{\vdash} u \rightsquigarrow \exists x.\ T \overset{?}{\vdash} \langle u, x\rangle \\
\mathsf{R}_2: & T \overset{?}{\vdash} \{\!|u_1|\!\}^{\mathsf{s}}_{u_2} \rightsquigarrow T \overset{?}{\vdash} u_1 \wedge T \overset{?}{\vdash} u_2 & \mathsf{R}_5: & T \overset{?}{\vdash} u \rightsquigarrow \exists x.\ T \overset{?}{\vdash} \{\!|u|\!\}^{\mathsf{s}}_{x} \wedge T \overset{?}{\vdash} x
\end{array}
$$

$$
\begin{array}{rl}
\mathsf{R}_6: & T \overset{?}{\vdash} x \ \wedge\ T' \overset{?}{\vdash} x \rightsquigarrow T \overset{?}{\vdash} x \ \text{ if } T \subseteq T' \\
\mathsf{R}_7: & T \overset{?}{\vdash} u \rightsquigarrow t \overset{?}{=} u \ \text{ if } t \in T
\end{array}
$$

Figure 4. Naive constraint reduction rules

This idea can be applied to other security properties than agreement or confidentiality, as long as they can be expressed with a formula on the same variables as the variables of the derivation constraints.

4. Solving a derivation constraint system

4.1. Constraint reduction

Constraint reduction (see [33]) is a relatively straightforward and efficient way to solve a derivation constraint system arising from a candidate protocol trace. It solves a system iteratively by reducing individual constraints until the system is solved.

Each reduction step should be *sound* in the sense that any solution of the reduced constraint system is a solution of the original system. At each step, several reductions may be possible. Hence, reduction sequences form a branching tree. Some paths may lead to a failure, while others yield solved forms, hence solutions. A desirable property is the *completeness* of the reduction rules: if σ is a solution of $\mathcal{C}$, then there is a possible reduction of $\mathcal{C}$ into a constraint system $\mathcal{C}'$, of which σ is a solution. In other words, there is a path in the tree that will yield a solved constraint system of which σ is a solution.

We could simply get a sound and complete set of constraint reduction rules by guessing the last intruder computation step. For instance in a system with only symmetric encryption and pairing, given a constraint $T \overset{?}{\vdash} u$ such that u is not a variable, for any solution σ, the last step in the proof of $T\sigma \vdash u\sigma$ must be a pairing (this is only possible if u is a pair) or an encryption (this is only possible if u is a ciphertext), or a decryption or a projection, or a trivial step (this is only possible if $u\sigma$ belongs to $T\sigma$). This yields a naive set of reduction rules, that is displayed in Figure 4.

The two first rules correspond to pairing and encryption. The three following rules correspond to projections and decryption. Then, the last one is actually the only one that builds a solution: we guess here that the proof is completed, as (the instance of) u belongs to (the corresponding instance of) T. In addition to such rules, we need an equality constraint solving rule:

$$t \overset{?}{=} u \ \wedge\ \mathcal{C} \to \sigma \wedge \mathcal{C}\sigma \qquad \text{if } \sigma = \mathsf{mgu}(t, u)$$

We do not include it in Figure 4 since this rule is a deterministic simplification rule: it may be applied to any constraint without considering any alternative simplification. By convention, $\sigma = \perp$ is a failure if the two terms are not unifiable.

Unfortunately, such rules are too naive: while sound and complete, they will not terminate, as the three rules that correspond to projections and decryption (namely R_3, R_4 and R_5) can be repeatedly applied any number of times. That is why we do not aim at reproducing any possible intruder's computation, but only some of its computations, and at least one for each resulting term. This results in using the same set of rules as the above one, however restricting the three rules R_3, R_4 and, R_5; while keeping completeness. This is, in essence, what is performed in the decision procedures based on constraint solving.

We now give a complete and terminating procedure, including primitives such as exclusive-or.

4.2. A constraint solving procedure for exclusive-or

We display in Figure 5 a set of constraint simplification rules for a theory of exclusive-or, pairing and symmetric encryption. The simplification rules exactly reflect the intruder capabilities. They must be applied "don't know": this is a non-deterministic procedure, in which all possible applicable rules must be considered if we wish to get a completeness result.

$$\begin{array}{lll}
\mathsf{Ax}: & T \overset{?}{\vdash} t \;\rightsquigarrow\; t \overset{?}{=} u & \text{if } u \in T \\
\mathsf{C}: & T \overset{?}{\vdash} f(t_1, \ldots, t_n) \;\rightsquigarrow\; T \overset{?}{\vdash} t_1 \wedge \ldots \wedge T \overset{?}{\vdash} t_n & \text{if } f \neq \oplus \\
\mathsf{C}_\oplus: & T \overset{?}{\vdash} u \oplus v \;\rightsquigarrow\; T \overset{?}{\vdash} u \wedge T \overset{?}{\vdash} v & \\
\mathsf{D}_{\pi_1}: & T \overset{?}{\vdash} t \;\rightsquigarrow\; T \overset{?}{\vdash} \langle u, v\rangle \wedge t \overset{?}{=} u & \text{if } \langle u, v\rangle \in \mathsf{St}(T) \\
\mathsf{D}_{\pi_2}: & T \overset{?}{\vdash} t \;\rightsquigarrow\; T \overset{?}{\vdash} \langle u, v\rangle \wedge t \overset{?}{=} v & \text{if } \langle u, v\rangle \in \mathsf{St}(T) \\
\mathsf{D}_{\mathsf{dec}}: & T \overset{?}{\vdash} t \;\rightsquigarrow\; T \overset{?}{\vdash} \{\!|u|\!\}^{\mathsf{s}}_v \wedge T \overset{?}{\vdash} v \wedge t \overset{?}{=} u & \text{if } \{\!|u|\!\}^{\mathsf{s}}_v \in \mathsf{St}(T) \\
\mathsf{D}_\oplus: & T \overset{?}{\vdash} t \;\rightsquigarrow\; T \overset{?}{\vdash} v_1 \wedge \ldots \wedge T \overset{?}{\vdash} v_n \wedge t \overset{?}{=} v_1 \oplus \ldots \oplus v_n & \\
& & \text{if } v_1, \ldots, v_n \in \mathsf{St}(T) \\
\mathsf{Geq}: & T \overset{?}{\vdash} u \oplus v \;\rightsquigarrow\; T \overset{?}{\vdash} w \oplus v \wedge u \overset{?}{=} w & \text{if } w \in \mathsf{St}(T) \cup \mathsf{St}(v) \\
& & \text{and } \mathsf{top}(u) \in \{\{\!|\_|\!\}^{\mathsf{s}}_{\_}, \langle \_, \_\rangle\}
\end{array}$$

Figure 5. Constraint transformation rules

In addition to these rules, equational constraints are simplified and the resulting substitutions are applied to the rest of the constraint:

$$\mathsf{U}: \qquad \mathcal{C} \wedge \mathcal{E} \;\rightarrow\; \bigvee_{\sigma \in \mathsf{mgu}(\mathcal{E})} \mathcal{C}\sigma\!\downarrow \wedge\ \sigma$$

Here, $\mathsf{mgu}(\mathcal{E})$ denotes a complete set of most general unifiers of $\mathcal{E}$ modulo $\mathsf{E}_\oplus$. Such a set is finite and has even more properties, as we will see in the Lemma 8. Contrary to the case of encryption and pairing only, the set of minimal unifiers, though finite, may not be reduced to a singleton.

Example 11 *The equation* $\langle x, x\rangle \oplus \langle y, y\rangle \overset{?}{=} \langle a, a\rangle \oplus \langle b, b\rangle$ *has two most general unifiers* $\sigma_1 = \{x \mapsto a, y \mapsto b\}$ *and* $\sigma_2 = \{x \mapsto b, y \mapsto a\}$.

In addition, we simplify the constraints with the rules

$$\mathsf{S}_1 : T \overset{?}{\vdash} u \ \wedge \ T' \overset{?}{\vdash} u \ \rightarrow \ T \overset{?}{\vdash} u \qquad\qquad \text{if } T \subseteq T'$$

$$\begin{aligned} \mathsf{S}_2 : T \overset{?}{\vdash} x \ \wedge \ T_1 \overset{?}{\vdash} u_1 \ \cdots \ \wedge \ T_n \overset{?}{\vdash} u_n & \\ \rightarrow \ T \overset{?}{\vdash} x \quad \wedge \quad T_1, x \overset{?}{\vdash} u_1 \quad \cdots \quad \wedge \ T_n, x \overset{?}{\vdash} u_n & \\ \text{if } T_1, \ldots, T_n \text{ are all left hand sides such that } T \subsetneq T_i, \text{ and } x \notin T_i. & \end{aligned}$$

We also use two simplifications of right members of constraints of the form $x \oplus t$:

$$\begin{aligned} &\mathsf{XR}_1 : T' \overset{?}{\vdash} x \wedge T \overset{?}{\vdash} x \oplus t \ \rightarrow \ T' \overset{?}{\vdash} x \wedge T \overset{?}{\vdash} t \qquad\qquad \text{if } T' \subseteq T \\ &\mathsf{XR}_2 : T \overset{?}{\vdash} x \oplus t \ \rightarrow \ \exists y.\, T \overset{?}{\vdash} y \wedge x \overset{?}{=} y \oplus t \\ &\qquad\qquad \text{if } x \notin \mathit{vars}(T) \cup \mathit{vars}(t) \text{ and } \mathsf{XR}_1 \text{ cannot be applied} \end{aligned}$$

By convention, newly quantified variables should not occur previously in the constraint.

The rules of Figure 5 concerning pairing and symmetric encryption are very similar to the naive rules of Figure 4. The only differences are the side constraints, that impose the newly introduced terms to be already subterms of the constraint. We will show that this is complete. On the other hand, this gives an upper bound on the possible right members of the constraints, allowing us to derive a terminating strategy.

Example 12 *Consider the constraint system given in Example 10 and for which* $\sigma = \{x \mapsto a \oplus b\}$ *is a solution. Let* $T_1 = \{a \oplus \{\!|c|\!\}^{\mathsf{s}}_b,\ b \oplus \{\!|c|\!\}^{\mathsf{s}}_b\}$ *and* $T_2 = T_1 \cup \{\{\!|a|\!\}^{\mathsf{s}}_x\}$. *We have the following transformation sequence that leads to a derivation constraint system in solved form for which* σ *is still a solution. For sake of simplicity, trivial equations are omitted.*

$$\left\{\begin{array}{l} T_1 \overset{?}{\vdash} x \\ T_2 \overset{?}{\vdash} c \end{array}\right. \rightarrow_{\mathsf{S}_2} \left\{\begin{array}{l} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} c \end{array}\right. \rightsquigarrow_{\mathsf{D}_{dec}} \left\{\begin{array}{l} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} \{\!|c|\!\}^{\mathsf{s}}_b \\ T_2, x \overset{?}{\vdash} b \end{array}\right. \rightsquigarrow_{\mathsf{D}_\oplus} \left\{\begin{array}{l} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} a \oplus \{\!|c|\!\}^{\mathsf{s}}_b \\ T_2, x \overset{?}{\vdash} a \\ T_2, x \overset{?}{\vdash} b \end{array}\right.$$

$$\rightsquigarrow_{\mathsf{Ax}} \begin{cases} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} a \\ T_2, x \overset{?}{\vdash} b \end{cases} \rightsquigarrow_{\mathsf{D}_\oplus} \begin{cases} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} a \\ T_2, x \overset{?}{\vdash} a \oplus \{\!|c|\!\}^{\mathsf{s}}_b \\ T_2, x \overset{?}{\vdash} b \oplus \{\!|c|\!\}^{\mathsf{s}}_b \end{cases} \rightsquigarrow_{\mathsf{Ax}} \begin{cases} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} a \\ T_2, x \overset{?}{\vdash} b \oplus \{\!|c|\!\}^{\mathsf{s}}_b \end{cases} \rightsquigarrow_{\mathsf{Ax}}$$

$$\begin{cases} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} a \end{cases} \rightsquigarrow_{\mathsf{D}_{dec}} \begin{cases} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} \{\!|a|\!\}^{\mathsf{s}}_x \\ T_2, x \overset{?}{\vdash} x \end{cases} \rightarrow_{\mathsf{S}_1} \begin{cases} T_1 \overset{?}{\vdash} x \\ T_2, x \overset{?}{\vdash} \{\!|a|\!\}^{\mathsf{s}}_x \end{cases} \rightsquigarrow_{\mathsf{Ax}} T_1 \overset{?}{\vdash} x$$

The rules can actually be refined, for instance preventing from applying successively twice $\mathsf{D}_\oplus$ on a same constraint (two consecutive $\oplus$ rules never occurs in a normal proof). Some of the rules (for instance Geq) might even be unnecessary.

It is not very difficult to show that the simplification rules $\mathsf{S}_1, \mathsf{S}_2, \mathsf{XR}_1, \mathsf{XR}_2, \mathsf{U}$ preserve the set of solutions, hence we may eagerly (and deterministically) apply these rules:

Lemma 3 (soundness and completeness of simplification rules) *The simplification rules* $\mathsf{U}, \mathsf{S}_1, \mathsf{S}_2, \mathsf{XR}_1, \mathsf{XR}_2$ *are sound and complete: they preserve the set of solutions.*

Then the rules of the Figure 5 are applied to simplified derivation constraints. It remains now to prove three main properties:

1. The rules do transform the derivation constraints into derivation constraints, and they do not introduce new solutions. This is stated in the Lemma 4.
2. If a derivation constraint system is not in solved form, then, for every solution σ, there is a rule that can be applied and that yields another constraint, of which σ is a solution; this is the *completeness* result, which we eventually prove in Corollary 1. This shows that every solution of a derivation constraint can be eventually retrieved from a solved form.
3. There is a complete strategy such that there is no infinite sequence of transformations. We will prove this last result Section 4.4.

Lemma 4 (soundness of transformation rules) *The rules of Figure 5 are sound: they transform a constraint system $\mathcal{C}$ into a constraint system $\mathcal{C}'$ such that any solution of $\mathcal{C}'$ is also a solution of $\mathcal{C}$.*

4.3. Completeness of the transformation rules

The completeness would be straightforward for a set of naive rules such as the rules displayed in the Figure 4. We imposed however additional restrictions, typically that, for decompositions, we may only consider the subterms of T. This reflects the fact that we may restrict our attention to normal proofs. However, our terms now contain variables. Thanks to Lemma 1, if π is a normal proof of $T\sigma\!\downarrow\, \vdash u\sigma\!\downarrow$ such that $\mathsf{last}(\pi)$ is a decomposition, then $\mathsf{step}(\pi) \in \mathsf{St}(T\sigma\!\downarrow)$. This does not necessarily mean that $\mathsf{hyp}(\mathsf{last}(\pi))$ are instances of terms in $\mathsf{St}(T)$, because $\mathsf{St}(T\sigma\!\downarrow) \not\subseteq \mathsf{St}(T)\sigma\!\downarrow$. The main difficulty for showing the completeness is to get an analog of Lemma 1, however lifted to terms with variables. This is what we start in Lemma 5, however with a slightly weaker conclusion.

Lemma 5 *Let T be a finite set of terms, u be a ground term in normal form and σ be a substitution mapping the variables of T to ground terms in normal form. We assume moreover that, for every $x \in vars(T)$, there is a $T_x \subsetneq T$ and a proof $\pi_{x,\sigma}$ of $T_x\sigma\!\downarrow\, \vdash x\sigma$, such that $\mathsf{last}(\pi_{x,\sigma})$ is not a decomposition and, for every $y \in vars(T_x)$, we have that $T_y \subsetneq T_x$. Let π be a normal proof of $T\sigma\!\downarrow\, \vdash u$.*

If $\mathsf{last}(\pi)$ is a decomposition, then

- *either there is $x \in vars(T)$ and a subproof π' of $\pi_{x,\sigma}$ such that $\mathsf{conc}(\pi') = u$ and $\mathsf{last}(\pi')$ is not a decomposition*
- *or else $u \in \mathsf{St}(T)\sigma\!\downarrow$*

For every $u_f \in \mathsf{fact}(u)$, we have that either $T\sigma\!\downarrow\, \vdash u_f$, or else $u_f \in \mathsf{St}(T)\sigma\!\downarrow$.

It is actually not sufficient to consider normal proofs to show the completeness of our transformation rules. We have to consider normal proofs that are *alien-free*. In an alien-free proof, we control the premisses of the instances of the XOR deduction rule.

A proof π of $T\sigma\!\downarrow\, \vdash u$ is *alien-free* if, for every subproof $\dfrac{\pi_1 \quad \cdots \quad \pi_n}{v}$ (XOR) of π, for every i, either $\mathsf{conc}(\pi_i) \in \mathsf{St}(T)\sigma\!\downarrow$ or $\mathsf{conc}(\pi_i) \in \mathsf{fact}(v)$.

Lemma 6 *Let T be a finite set of terms, σ be a substitution mapping the variables of T to ground terms in normal form and u be a ground term in normal form. We assume that, for every variable $x \in vars(T)$, there is a $T_x \subsetneq T$ and a normal proof $\pi_{x,\sigma}$ of $T_x\sigma\!\downarrow\, \vdash x\sigma$ such that $\mathsf{last}(\pi_{x,\sigma})$ is not a decomposition and, for every $y \in vars(T_x)$, we have that $T_y \subsetneq T_x$. We assume moreover here that $vars(T) \subseteq T$ and $vars(T_x) \subseteq T_x$ for every variable x.*

If π_0 is a normal proof of $T\sigma\!\downarrow\, \vdash u$, then there is a normal alien-free proof π of $T\sigma\!\downarrow\, \vdash u$.

Example 13 *Consider the following derivation constraint system.*

$$\begin{array}{r} a,\, b,\, c \oplus d,\, f,\, g \overset{?}{\vdash} x \\ a,\, b,\, c \oplus d,\, f,\, g,\, c \oplus e,\, k,\, \{\!|x \oplus d|\!\}^{\mathsf{s}}_k, x \overset{?}{\vdash} \langle a, b\rangle \oplus e \end{array}$$

Let $\sigma = \{x \mapsto \langle a,b\rangle \oplus c \oplus d \oplus \langle f,g\rangle\}$, $T_x = \{a,\, b,\, c \oplus d,\, f,\, g\}$, and $T = T_x \cup \{c \oplus e,\, k,\, \{\!|x \oplus d|\!\}^{\mathsf{s}}_k, x\}$. We have that $\{x\} = vars(T) \subseteq T$, and $\emptyset = vars(T_x) \subseteq T_x$. The proof $\pi_{x,\sigma}$ described below is a proof of $T_x\sigma\!\downarrow\, \vdash x\sigma$ that does not end with a decomposition.

$$\pi_{x,\sigma} = \left\{ \quad \dfrac{c \oplus d \qquad \dfrac{a \quad b}{\langle a, b\rangle} \qquad \dfrac{f \quad g}{\langle f, g\rangle}}{c \oplus d \oplus \langle a, b\rangle \oplus \langle f, g\rangle} \right.$$

Let π_0 be the following proof of $T\sigma \vdash \langle a, b\rangle \oplus e$.

$$\pi_0 = \left\{ \quad \dfrac{\dfrac{\{\!|x \oplus d|\!\}^{\mathsf{s}}_k\sigma{\downarrow} \quad k}{(x \oplus d)\sigma{\downarrow}} \qquad c \oplus e \qquad \dfrac{f \quad g}{\langle f, g\rangle}}{\langle a, b\rangle \oplus e} \right.$$

The proof π_0 is a proof in normal form whose last step is not a decomposition. According to Lemma 5, for every $u_f \in \{\langle a,b\rangle, e\}$, we have that either $T\sigma{\downarrow} \vdash u_f$, or $u_f \in \mathsf{St}(T)\sigma{\downarrow}$. Indeed, we have that $T\sigma \vdash \langle a,b\rangle$, and $e \in \mathsf{St}(T)\sigma{\downarrow}$. However, π_0 is not alien-free since $\langle f, g\rangle$ is neither in $\mathsf{St}(T)\sigma$ nor a factor of $\langle a,b\rangle \oplus e$. An alien-free proof π_0' of $T\sigma{\downarrow} \vdash \langle a,b\rangle \oplus e$ is given below:

$$\pi_0' = \left\{ \quad \dfrac{x\sigma \quad c \oplus d \quad \dfrac{a \quad b}{\langle a, b\rangle} \quad c \oplus e \quad \dfrac{\{\!|x \oplus d|\!\}^{\mathsf{s}}_k\sigma{\downarrow} \quad k}{(x \oplus d)\sigma{\downarrow}}}{\langle a, b\rangle \oplus e} \right.$$

Lemma 7 *There is a mapping μ from derivation constraint systems and substitutions to a well-founded ordered set such that, for every derivation constraint system $\mathcal{C}$ and every solution σ of $\mathcal{C}$, either $\mathcal{C}$ is in solved form or else there is a constraint system $\mathcal{C}'$ such that $\mathcal{C} \rightsquigarrow \mathcal{C}'$, σ is a solution of $\mathcal{C}'$, and $\mu(\mathcal{C}, \sigma) > \mu(\mathcal{C}', \sigma)$.*

Note that this lemma does not prove the termination of our rules, since we only show that, *for every solution σ*, the derivation sequences of which σ is a solution is finite. There might be however an infinite sequence of solutions, each yielding a longer derivation sequence than the previous ones.

As a consequence of this lemma, we get a completeness result, stating that, for every solution σ of $\mathcal{C}$, there is a solved system that can be derived from $\mathcal{C}$ and of which σ is a solution:

Corollary 1 (completeness) *If $\mathcal{C}$ is a derivation constraint system and σ is a solution of $\mathcal{C}$, then there is a solved derivation constraint system $\mathcal{C}'$ such that $\mathcal{C} \rightsquigarrow^* \mathcal{C}'$ and σ is a solution of $\mathcal{C}'$.*

4.4. Termination

The termination can be ensured by a complete strategy on the rules application, as long as unification reduces the number of variables. This property is true, at least, for the combination of a free theory and the exclusive-or theory.

Lemma 8 *Suppose that $\mathcal{E}$ is a set of equations over the disjoint combination of the theory of $\{\oplus, 0\}$ and a free theory with (at least) symmetric encryption and pairing. Then either $\mathcal{E}$ has no solution, or every substitution is a solution of $\mathcal{E}$, or else there is a finite complete set Σ of unifiers of $\mathcal{E}$ such that, for any $\sigma \in \Sigma$, we have that $|vars(\mathcal{E}\sigma)| < |vars(\mathcal{E})|$.*

Proof (sketch):
To establish this result, we rely on the procedure described in [3] for combining disjoint equational theories. Given a system of equations, we let $n_{\mathcal{E}}$ be the maximal number of independent equations in $\mathcal{E}$, *i.e.* we only consider equations that are not a logical

consequence of the other equations (with respect to the considered equational theory). Then, we show that the difference between $n_{\mathcal{E}}$ and the number of variables $n_{\mathcal{V}}$ is non-decreasing. This allows us to conclude. $\square$

This says that the number of new variables introduced by the substitution is less than the number of variables eliminated by the substitution. The examples below show that it is sometimes needed to introduce new variables.

Example 14 *We consider two sets that are made up of one equation only.*

- *Let $\mathcal{E} = \{x = f(x \oplus y)\}$. This set has mgu $\sigma = \{x \mapsto f(z), y \mapsto f(z) \oplus z\}$ where z is a new variable. We have that*
$$|vars(\mathcal{E}\sigma)| = |z| = 1 < 2 = |\{x, y\}| = vars(\mathcal{E}).$$
- *Let $\mathcal{E} = \{x = f(x \oplus y) \oplus z\}$. This set has mgu $\sigma = \{x \mapsto f(x') \oplus z, y \mapsto f(x') \oplus x' \oplus z\}$ where x' is a new variable. We have that*
$$|vars(\mathcal{E}\sigma)| = |\{x', z\}| = 2 < 3 = |\{x, y, z\}| = |vars(\mathcal{E})|.$$

We use the following strategy: the rules $\mathsf{U}, \mathsf{S}_1, \mathsf{S}_2, \mathsf{XR}_1, \mathsf{XR}_2$ are applied eagerly (in this order) and then the rules of Figure 5 are applied to a constraint $T \overset{?}{\vdash} u$ such that, for any $T' \subsetneq T$ and every constraint $T' \overset{?}{\vdash} v$, we have that v is a variable that does not occur in T'. According to the results of the previous section, this strategy preserves the completeness.

Let $\mathcal{V}(\mathcal{C})$ be the pair (n, m) such that n is the number of unsolved variables of $\mathcal{C}$ and m is the number of variables of $\mathcal{C}$ that are neither solved nor pre-solved. Such pairs are ordered lexicographically.

Lemma 9 *Let $\mathcal{C}$ and $\mathcal{C}'$ be two derivation constraint systems such that $\mathcal{C} \rightsquigarrow \mathcal{C}'$, we have that $\mathcal{V}(\mathcal{C}) \geq \mathcal{V}(\mathcal{C}')$.*

Proof (sketch):
Let us consider successively the transformation rule, including the simplification rules, for which the statement is not straightforward.

- When applying U, either the equation is trivial (in which case $\mathcal{V}(\mathcal{C})$ remains constant), or the equation is unsatisfiable (in which case $\mathcal{V}(\mathcal{C})$ is decreasing), or else, thanks to Lemma 8, there are more variables that become solved than variables that are introduced. Hence the first component of $\mathcal{V}(\mathcal{C})$ is strictly decreasing.
- When applying XR_2, followed by a replacement of x with $y \oplus t$, the number of unsolved variables is constant (x was unsolved before applying the rule and becomes solved after the replacement with $y \oplus t$). The number of non-pre-solved variables is strictly decreasing since x was not pre-solved before the rule and x, y are both pre-solved after applying the rule.

- The other rules do not introduce new variables: the first component of $\mathcal{V}(\mathcal{C})$ can only decrease or remain constant. According to our strategy, the only situations in which the number of pre-solved variables could decrease is when a transformation rule is applied to a constraint $T \overset{?}{\vdash} x$ where x is a variable. This may happen with the rules $\mathsf{Ax}, \mathsf{D}_{\pi_1}, \mathsf{D}_{\pi_2}, \mathsf{D}_{\mathtt{dec}}, \mathsf{D}_{\oplus}$, that mimic decompositions. But, in these cases, at least one non trivial equation is added, which yields, after applying U, a strict decreasingness of the first component of $\mathcal{V}(\mathcal{C})$.

□

Now, if a transformation preserves $\mathcal{V}(\mathcal{C})$, then there is no simplification by $\mathsf{U}, \mathsf{XR}_1, \mathsf{XR}_2$ that takes place at this step (except trivial equations simplifications), and the side conditions of the rules ensure that no new subterm appears. More precisely, let $\mathcal{T}(\mathcal{C}) = \mathsf{St}(\mathcal{C}) \cup \{u \mid \exists v, u \oplus v \in \mathsf{St}(\mathcal{C})\}$, then we have the following lemma.

Lemma 10 *Let $\mathcal{C}$ and $\mathcal{C}'$ be two derivation constraint systems such that $\mathcal{C} \rightsquigarrow \mathcal{C}'$ and $\mathcal{V}(\mathcal{C}) = \mathcal{V}(\mathcal{C}')$, then $\mathcal{T}(\mathcal{C}') \subseteq \mathcal{T}(\mathcal{C})$.*

It suffices then to cut looping branches in order to get a terminating and complete procedure: according to Lemma 9, from any infinite transformation sequence we may extract an infinite transformation sequence $\mathcal{C}_0 \rightsquigarrow \mathcal{C}_1 \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_n \rightsquigarrow \cdots$ on which $\mathcal{V}(\mathcal{C}_0) = \mathcal{V}(\mathcal{C}_1) = \cdots = \mathcal{V}(\mathcal{C}_n) = \cdots$. Then, according to Lemma 10, the set of subterms in any derivation constraint system of the sequence is bounded by the set of subterms of the original constraint system $\mathcal{C}_0$. Now, each individual constraint $T \overset{?}{\vdash} u$ can appear only once in any constraint system, thanks to S_1. Therefore, there are only finitely many derivation constraint systems that can be built once the set of subterms (more precisely the set $\mathcal{T}(\mathcal{C})$) is fixed. This means that any infinite sequence of transformations must be looping.

Now, remains to justify that cutting the loops still yields a complete procedure. The reason is that, according to the lemma 7, for every solution σ of $\mathcal{C}_0$, there is a sequence $\mathcal{C}_0 \rightsquigarrow \mathcal{C}_1 \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_n$ such that σ is a solution of $\mathcal{C}_n$ and $\mu(\mathcal{C}_0, \sigma) > \mu(\mathcal{C}_n, \sigma)$, hence the sequence does not include any loop, since the ordering on measures is well-founded.

As a conclusion of this section, we get a termination result:

Theorem 1 (Termination) *There is a complete strategy such that any simplification sequence is finite and yields a solved form.*

The termination proof does not give much information about complexity. Furthermore, the strategy is quite rough and can be refined: there is still some work to do, if we wish to get the shortest simplification sequence. For instance we would need to avoid repeating several times the same simplifications (as explained in [16]).

With such an additional work, the simplification of constraints might yield a NP decision procedure for the satisfiability of derivation constraint systems.

4.5. Further results

The constraint solving approach to the security of protocols was introduced in [33]. In the case of symmetric and asymmetric cryptography, it has been shown to yield a NP decision procedure (for the existence of an attack) in [36]. For the same set of cryptographic primitives as the one we have shown in this chapter, but without exclusive-or, a full constraint solving procedure, preserving all solutions, is given in [16]. Such a procedure allows us to decide more trace properties (such as authentication, key cycles, timeliness).

Extensions of decision results (for secrecy) to other primitives have been extensively studied. For exclusive-or there are two concurrent publications [19,11]. However, these results only provide decision results. In this respect, the procedure that we gave in this chapter is a new result, since it preserves all solutions; it could be used for deciding other security properties.

The second most popular equational theories are modeling parts of arithmetic, in particular modular exponentiation. There is no hope to get a general decision result for the full arithmetic, as unification is already undecidable. The first fragment considers some properties of modular exponentiation that are sufficient for modeling some classical protocols [31,10,12]. In case of an Abelian group theory, the constraint solving approach is also proved to yield a decision procedure [38,34]. This is extended, yet considering a richer fragment of arithmetic in [8].

There are many more relevant equational theories, as described in [21]. For instance homomorphism properties are considered in [23,24], blind signatures in [22,6],... Some classes of equational theories are considered, relying on a constraint solving approach in [24,9].

Finally, we complete the tour by mentioning combinations of decision procedures: for disjoint theories [13], hierarchical combinations [14] and more [8].

All these works make use of techniques similar to derivation constraint solving, but they also use a “small attack property”, showing that, if a derivation constraint is satisfiable, then there is a small solution. This kind of result allows us to restrict the set of solutions that has to be considered; the constraint solving rules (or their counterparts) then do not necessarily preserve the solutions, but only preserve the small solutions, hence the satisfiability. In this chapter (as in [16,9]), we went a step further, preserving the set of all solutions.

4.6. Software resources

There are a few specialized theorem provers that can deal with some algebraic properties of cryptographic primitives [5,7,40]. There are also ways of getting rid of some equational axioms, and then using the verifiers that are designed for free term algebras [17,27,28]. Only some [33,4,40] are (currently) really relying on constraint solving techniques. But the recent theoretical advances in this area may, in the next few years, yield new tools based on these techniques.

5. Research directions

Much is known at this stage about the foundations of security protocol analysis. The basic decidability and complexity results are known. Several specialized software tools, and methods for applying more general tools, are available, with staggering improvements in performance compared with a decade ago: seconds instead of hours to analyze most protocols. More attention is needed now to the presentation of the best algorithms, so that future students, tool developers, and analysts will be able to build on clear and useful knowledge rather than tricks hidden in software.

There is always a need to extend analysis approaches to cover more kinds of cryptographic primitives, such as bilinear pairings, used in elliptic curve cryptography, and zero-knowledge proofs. Another persistent demand is to handle more kinds of security goals, such as anonymity, fair exchange, group key management, and properties expressed in terms of observational equivalence.

Observational equivalence deserves a special attention. It allows one to state some stronger security properties, typically that an attacker cannot learn anything relevant, because the process is indistinguishable from an ideal process in which all relevant informations have been shuffled or hidden.

The constraint solving approach is relevant for deciding equivalences of processes. It requires however significant new insights since the constraint solving method that we described in this chapter is complete only w.r.t. *what* an attacker can deduce, but not w.r.t. *how* it can be deduced. On the other hand, if an attacker has different ways to deduce a given message in two different experiments, he could distinguish between them.

At this stage in the development of security protocol analysis, the most useful challenge for the research community might not be research at all, but rather a transfer of the state of the art to the state of practice. A collaboration of researchers, educators, and tool builders may be necessary to create texts, expectations, and software that exhibits the best of what is possible and explains how to use it. After this kind of consolidation step, it will be easier to see what is most important to do next.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.

[2] M. Arapinis, S. Delaune, and S. Kremer. From one session to many: Dynamic tags for security protocols. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, volume 5330 of *LNAI*, pages 128–142. Springer, 2008.

[3] F. Baader and K. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21:211–243, 1996.

[4] D. Basin, S. Mödersheim, and L. Viganò. Constraint differentiation: a new reduction technique for constraint-based analysis of security protocols. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'03)*, pages 335–344. ACM, 2003.

[5] D. Basin, S. Mödersheim, and L. Viganò. Algebraic intruder deductions. In *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *LNAI*, pages 549–564. Springer-Verlag, 2005.

[6] V. Bernat and H. Comon-Lundh. Normal proofs in intruder theories. In *Revised Selected Papers of the 11th Asian Computing Science Conference (ASIAN'06)*, volume 4435 of *LNCS*. Springer, 2008.

[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] S. Bursuc and H. Comon-Lundh. Protocol security and algebraic properties: Decision results for a bounded number of sessions. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA'09)*, volume 5595 of *LNCS*, pages 133–147. Springer, 2009.

[9] S. Bursuc, S. Delaune, and H. Comon-Lundh. Deducibility constraints. In *Proceedings of the 13th Asian Computing Science Conference (ASIAN'09)*, LNCS. Springer, 2009.

[10] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In *Proceedings of the 23rd Conference on*

Foundations of Software Technology and Theoretical Computer Science (FST&TCS'03), volume 2914 of *LNCS*, 2003.

[11] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. *Theoretical Computer Science*, 338(1-3):247–274, 2005.

[12] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Complexity results for security protocols with Diffie-Hellman exponentiation and commuting public key encryption. *ACM Trans. Comput. Log.*, 9(4), 2008.

[13] Y. Chevalier and M. Rusinowitch. Combining Intruder Theories. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, ICALP'05*, volume 3580 of *LNCS*, pages 639–651. Springer, 2005.

[14] Y. Chevalier and M. Rusinowitch. Hierarchical combination of intruder theories. In *Proceedings of the 17th International Conference on Rewriting Techniques and Application (RTA'06)*, volume 4098 of *LNCS*, pages 108–122, 2006.

[15] Y. Chevalier and M. Rusinowitch. Hierarchical combination of intruder theories. *Information and Computation*, 206(2-4):352–377, 2008.

[16] 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.

[17] 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*, 2005.

[18] H. Comon-Lundh, S. Delaune, and J. Millen. Constraint solving techniques and enriching the model with equational theories. Research Report LSV-10-18, Laboratoire Spécification et Vérification, ENS Cachan, France, Nov. 2010.

[19] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*. IEEE Comp. Soc., 2003.

[20] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *Proceedings of the 9th Static Analysis Symposium (SAS'02)*, volume 2477 of *LNCS*, pages 326–341, 2002.

[21] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[22] V. Cortier, M. Rusinowitch, and E. Zalinescu. A resolution strategy for verifying cryptographic protocols with CBC encryption and blind signatures. In *Proceedings of the 7th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'05)*, pages 12–22. ACM press, 2005.

[23] S. Delaune, P. Lafourcade, D. Lugiez, and R. Treinen. Symbolic protocol analysis in presence of a homomorphism operator and *exclusive or*. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, (ICALP'06), Part II*, volume 4052 of *LNCS*, pages 132–143, 2006.

[24] S. Delaune, P. Lafourcade, D. Lugiez, and R. Treinen. Symbolic protocol analysis for monoidal equational theories. *Information and Computation*, 206(2-4):312–351, 2008.

[25] D. Dolev and A. Yao. On the security of public key protocols. In *Proceedings of the 22nd Symposium on Foundations of Computer Science*, pages 350–357, Nashville (USA, Tennessee, 1981. IEEE Comp. Soc. Press.

[26] D. Kähler and R. Küsters. Constraint solving for contract-signing protocols. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *LNCS*, pages 233–247. Springer, 2005.

[27] R. Küsters and T. Truderung. Reducing protocol analysis with XOR to the XOR-free case in the Horn theory based approach. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'08)*, pages 129–138. ACM, 2008.

[28] R. Küsters and T. Truderung. Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF 2009)*, pages 157–171. IEEE Computer Society, 2009.

[29] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2-3):89–146, 1999.

[30] C. Lynch and C. Meadows. On the relative soundness of the free algebra model for public key encryp-

tion. *ENTCS*, 125(1):43–54, 2005.

[31] C. Meadows and P. Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS'02)*, 2002.

[32] J. Millen. On the freedom of decryption. *Information Processing Letters*, 86(6):329–333, 2003.

[33] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2001.

[34] J. Millen and V. Shmatikov. Symbolic protocol analysis with an abelian group operator or Diffie-Hellman exponentiation. *Journal of Computer Security*, 2005.

[35] R. Ramanujam and S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

[36] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. IEEE Comp. Soc. Press, 2001.

[37] S. Schneider. Security properties and CSP. In *IEEE Symposium on Security and Privacy*, pages 174–187, 1996.

[38] V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *Proceedings of the 13th European Symposium on Programming (ESOP'04)*, volume 2986 of *LNCS*, 2004.

[39] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1), 1999.

[40] M. Turuani. The CL-Atse protocol analyser. In *Proceedings of the 17th International Conference on Term Rewriting and Applications (RTA'06)*, volume 4098 of *LNCS*, pages 277–286. Springer, 2006.