Deciding security for protocols with recursive tests*

Mathilde Arnaud[1,2], Véronique Cortier[2], and Stéphanie Delaune[1]

[1] LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France, France
[2] LORIA, CNRS, France

Abstract. Security protocols aim at securing communications over public networks. Their design is notoriously difficult and error-prone. Formal methods have shown their usefulness for providing a careful security analysis in the case of standard authentication and confidentiality protocols. However, most current techniques do not apply to protocols that perform recursive computation *e.g.* on a list of messages received from the network.

While considering general recursive input/output actions very quickly yields undecidability, we focus on protocols that perform *recursive tests* on received messages but output messages that depend on the inputs in a standard way. This is in particular the case of secured routing protocols, distributed right delegation or PKI certification paths. We provide NPTIME decision procedures for protocols with recursive tests and for a bounded number of sessions. We also revisit constraint system solving, providing a complete symbolic representation of the attacker knowledge.

1 Introduction

Security protocols are communication programs that aim at securing communications over public channels like the Internet. It has been recognized that designing a secure protocol is a difficult and error-prone task. Indeed, protocols are very sensitive to small changes in their description and many protocols have been shown to be flawed several years after their publication (and deployment). Formal methods have been successfully applied to the analysis of security protocols, yielding the discovery of new attacks like the famous man-in-the-middle attack in the Needham-Schroeder public key protocol [17] or, more recently, a flaw in Gmail [4]. Many decision procedures have been proposed (*e.g.* [18, 20]) and efficient tools have been designed such as ProVerif [8] and AVISPA [3].

While formal methods have been successful in the treatment of security protocols using standard primitives like encryption and signatures, there are much fewer results for protocols with recursive primitives, that is, primitives that involve iterative or recursive operations. For example, in group protocols, the server

* This work has been partially supported by the project ANR-07-SESU-002 AVOTÉ. The research leading to these results has also received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement number 258865 (ERC ProSecure project).

or the leader typically has to process a request that contains the contributions of each different agent in the group and these contributions are used to compute a common shared key (see *e.g.* the Asokan-Ginzboorg group protocol [6]). Secured versions of routing protocols [9, 14, 12] also require the nodes (typically the node originating the request) to check the validity of the route they receive. This is usually performed by checking that each node has properly signed (or MACed) some part of the route, the whole incoming message forming a chain where each component is a contribution from a node in the path. Other examples of protocols performing recursive operations are certification paths for public keys (see *e.g.* X.509 certification paths [13]) and right delegation in distributed systems [7].

Recursive operations may yield complex computations. Therefore it is difficult to check the security of protocols with recursive primitives and very few decision procedures have been proposed for recursive protocols. One of the first decidability results [16] holds when the recursive operation can be modeled using tree transducers, which forbids any equality test and also forbids composed keys and chained lists. In [21] recursive computation is modeled using Horn clauses and an NEXPTIME procedure is proposed. This is extended in [15] to include the Exclusive Or operator. This approach however does not allow composed keys nor list mapping (where the same operation, *e.g.* signing, is applied to each element of the list). To circumvent these restrictions, another procedure has been proposed [10] to handle list mapping provided that each element of the list is properly tagged. No complexity bound is provided. All these results hold for rather limited classes of recursive operations (on lists of terms). This is due to the fact that even a single input/output step of a protocol may reveal complex information, as soon as it involves a recursive computation. Consequently, recursive primitives very quickly yield undecidability [16].

Our contributions. The originality of our approach consists in considering protocols that perform standard input/output actions (modeled using usual pattern matching) but that are allowed to perform *recursive tests* such as checking the validity of a route or the validity of a chain of certificates. Indeed, several families of protocols use recursivity only for performing sanity checks at some steps of the protocol. This is in particular the case of secured routing protocols, distributed right delegation, and PKI certification paths.

For checking security of protocols with recursive tests (for a bounded number of sessions), we reuse the setting of constraint systems [18, 11] and add tests of membership to recursive languages. As a first contribution, we revisit the procedure of [11] for solving constraint systems and obtain a complete symbolic representation of the knowledge of the attacker, in the spirit of the characterization obtained in [1] in the passive case (with no active attacker). This result holds for general constraint systems and is of independent interest.

Our second contribution is the proposition of (NPTIME) decision procedures for two classes of recursive languages (used for tests): *link-based recursive languages* and *mapping-based languages*. A link-based recursive language contains chains of links where consecutive links have to satisfy a given relation. A typical example is X.509 public key certificates [13] that consist in a chain of signatures of the form: $[[\![\langle A_1, \mathsf{pub}(A_1)\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2)\rangle]\!]_{\mathsf{sk}(A_3)}; \cdots ; [\![\langle A_n, \mathsf{pub}(A_n)\rangle]\!]_{\mathsf{sk}(S)}]$. The purpose of this chain is to authenticate the public key of A_1. The chain begins with the certificate $[\![\langle A_1, \mathsf{pub}(A_1)\rangle]\!]_{\mathsf{sk}(A_2)}$, and each certificate in the chain is signed by the entity identified by the next certificate in the chain. The chain terminates with a certificate signed by a trusted party S.

A mapping-based language contains lists that are based on a list of names (typically names of agents involved in the protocol session) and are uniquely defined by it. Typical examples can be found in the context of routing protocols, when nodes check for the validity of the route. For example, in the SMNDP protocol [12], a route from the source A_0 to the destination A_n is represented by a list $l_{route} = [A_n; \ldots ; A_1]$. This list is accepted by the source node A_0 only if the received message is of the form:

$$[[\![\langle A_n, A_0, l_{route}\rangle]\!]_{\mathsf{sk}(A_1)}; [\![\langle A_n, A_0, l_{route}\rangle]\!]_{\mathsf{sk}(A_2)}; \ldots ; [\![\langle A_n, A_0, l_{route}\rangle]\!]_{\mathsf{sk}(A_n)}].$$

Note that a link $[\![\langle A_n, A_0, l_{route}\rangle]\!]_{\mathsf{sk}(A_i)}$ both depends on the list l_{route} and on its i-th element.

For each of these two languages, we show that it is possible to bound the size of a minimal attack (bounding in particular the size of the lists used in membership tests), relying on the new characterization we have obtained for solutions of constraint systems. As a consequence, we obtained two new NP decision procedures for two classes of languages that encompass most of the recursive tests involved in secured routing protocols and chain certificates. We illustrate our results with several examples of relevant recursive languages. Detailed proofs of our results can be found in [5].

2 Models for security protocols

2.1 Messages

As usual, messages are represented using a term algebra. We consider the *sorted signature* $\mathcal{F} = \{\mathsf{senc}, \mathsf{aenc}, [\![\_]\!]_\_, \langle \_, \_\rangle, \mathsf{h}, ::, [], \mathsf{pub}, \mathsf{priv}, \mathsf{vk}, \mathsf{sk}\}$ with corresponding arities:

- $ar(\mathsf{f}) = \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$ for $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, [\![\_]\!]_\_, \langle \_, \_\rangle\}$,
- $ar(\mathsf{h}) = \mathsf{Msg} \to \mathsf{Msg}$,
- $ar(::) = \mathsf{Msg} \times \mathsf{List} \to \mathsf{List}$, and $ar([]) = \mathsf{List}$,
- $ar(\mathsf{f}) = \mathsf{Base} \to \mathsf{Msg}$ for $\mathsf{f} \in \mathcal{F}_s = \{\mathsf{pub}, \mathsf{priv}, \mathsf{vk}, \mathsf{sk}\}$.

The sort Msg is a supersort of List and Base. The symbol $\langle\rangle$ represents the pairing function, :: is the list constructor, and [] represents the empty list. For the sake of clarity, we write $\langle u_1, u_2, u_3\rangle$ for the term $\langle u_1, \langle u_2, u_3\rangle\rangle$, and $[u_1; u_2; u_3]$ for $u_1{::}(u_2{::}(u_3{::}[]))$. The terms $\mathsf{pub}(A)$ and $\mathsf{priv}(A)$ represent respectively the public and private keys associated to an agent A, whereas the terms $\mathsf{sk}(A)$ and $\mathsf{vk}(A)$ represent respectively the signature and verification keys associated to an agent A. The function symbol senc (resp. aenc) is used to model symmetric (resp.

asymmetric) encryption whereas the term $[\![m]\!]_{\mathsf{sk}(A)}$ represents the message m signed by the agent A.

We consider an infinite set of *names* $\mathcal{N} = \{Rep, Req, N, K, A, S, D, Id\dots\}$ having Base sort. These names typically represent constants, nonces, symmetric keys, or agent names. Moreover, we assume that we have three disjoint infinite sets of variables, one for each sort, denoted $\mathcal{X}_{\mathsf{Base}}$, $\mathcal{X}_{\mathsf{List}}$, and $\mathcal{X}_{\mathsf{Msg}}$ respectively. We write $vars(u)$ for the set of variables occurring in u. A term is *ground* if it has no variables.

We write $st(u)$ for the set of *subterms* of a term u. This notion is extended as expected to sets of terms. *Substitutions* are written $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ with $dom(\sigma) = \{x_1, \dots, x_n\}$. They are assumed to be well-sorted substitutions, that is the sort of each x_i is a supersort of the sort of t_i. Such a substitution σ is *ground* if all the t_i are ground terms. The application of a substitution σ to a term u is written $u\sigma$. A most general unifier of terms u_1 and u_2 is a substitution (when it exists) denoted by $mgu(u_1, u_2)$.

2.2 Intruder capabiblities

The ability of the intruder is modeled by a deduction system described below and corresponds to the usual rules representing attacker abilities (often called Dolev-Yao rules).

$$\frac{u_1 \quad \dots \quad u_n}{\mathsf{f}(u_1, \dots, u_n)}\ \mathsf{f} \in \mathcal{F} \smallsetminus \mathcal{F}_s \qquad \frac{\langle u_1, u_2\rangle}{u_i}\ i\in\{1,2\} \qquad \frac{u_1{::}u_2}{u_i}\ i\in\{1,2\}$$

$$\frac{\mathsf{senc}(u_1, u_2) \quad u_2}{u_1} \qquad \frac{\mathsf{aenc}(u_1, \mathsf{pub}(u_2)) \quad \mathsf{priv}(u_2)}{u_1} \qquad \frac{[\![u_1]\!]_{\mathsf{sk}(u_2)}}{u_1}\ \textit{(optional)}$$

The first inference rule describes the *composition rules*. The remaining inference rules describe the *decomposition* rules. Intuitively, these deduction rules say that an intruder can compose messages by pairing, building lists, encrypting and signing messages provided he has the corresponding keys. Conversely, he can retrieve the components of a pair or a list, and he can also decompose messages by decrypting provided he has the decryption keys. For signatures, the intruder is also able to *verify* whether a signature $[\![m]\!]_{\mathsf{sk}(a)}$ and a message m match (provided he has the verification key $\mathsf{vk}(a)$), but this does not give him any new message. That is why this capability is not represented in the deduction system. We also consider an optional rule that expresses that an intruder can retrieve the whole message from its signature. This property may or may not hold depending on the signature scheme, and that is why this rule is optional. Our results hold in both cases (that is, when the deduction relation $\vdash$ is defined with or without this rule).

A term u is *deducible* from a set of terms T, denoted by $T \vdash u$, if there exists a *proof*, *i.e.* a tree such that the root is labelled with u, the leaves are labelled with $v \in T$ and every intermediate node is an instance of one of the rules of the deduction system.

2.3 Constraint systems

Constraint systems are quite common (see *e.g.* [11, 18]) in modeling security protocols. A constraint system represents in a symbolic and compact way which trace instances of a protocol are possible once an interleaving of actions has been fixed. They are used, for instance, to specify secrecy preservation of security protocols under a particular, finite scenario. Note that, even if the scenario is fixed, there are still many (actually infinitely many) possible instances of it, because the intruder may affect the content of the messages by intercepting sent messages and forging received messages. The behaviour of the attacker is taken into account relying on the inference system presented in Section 2.2. To enforce the intruder capabilities, we also assume he knows an infinite set of names $\mathcal{I}$ that he might use at his will to mount attacks.

Definition 1 (constraint system). *A* constraint system *is a pair* $(\mathcal{C}, \mathcal{I})$ *such that* $\mathcal{I}$ *is a non empty (and possibly infinite) set of names, and* $\mathcal{C}$ *is either* $\perp$ *or a finite conjunction* $\bigwedge_{i=1}^{n} T_i \Vdash u_i$ *of expressions called* deducibility constraints*, where each* T_i *is a finite set of terms, called the* left-hand side *of the constraint and each* u_i *is a term, called the* right-hand side *of the constraint, such that:*

- $T_i \subseteq T_{i+1}$ *for every* i *such that* $1 \leq i < n$*;*
- *if* $x \in vars(T_i)$ *for some* i *then there exists* $j < i$ *such that* $x \in vars(u_j)$*.*

Moreover, we assume that $st(\mathcal{C}) \cap \mathcal{I} = \emptyset$*.*

The second condition in Definition 1 says that each time a new variable is introduced, it first occurs in some right-hand side. The left-hand side of a constraint system usually represents the messages sent on the network, while the right-hand side represents the message expected by the party.

Definition 2 (non-confusing solution). *Let* $(\mathcal{C}, \mathcal{I})$ *be a constraint system where* $\mathcal{C} = \bigwedge_{i=1}^{n} T_i \Vdash u_i$*. A* solution *of* $(\mathcal{C}, \mathcal{I})$ *is a ground substitution* θ *whose domain is* $vars(\mathcal{C})$ *such that* $T_i\theta \cup \mathcal{I} \vdash u_i\theta$ *for every* $i \in \{1, \dots, n\}$*. The empty constraint system is always satisfiable whereas* $(\perp, \mathcal{I})$ *denotes an unsatisfiable constraint system. Furthermore, we say that* θ *is* non-confusing *for* $(\mathcal{C}, \mathcal{I})$ *if* $t_1 = t_2$ *for any* $t_1, t_2 \in st(T_n)$ *such that* $t_1\theta = t_2\theta$*.*

In other words, non-confusing solutions do not map two distinct subterms of a left-hand side of the constraint system to the same term. Later on, we will show that we can restrict ourselves to consider this particular case of solutions.

Constraint systems model protocols that perform pattern matching only. In particular, deducibility constraints cannot ensure that some message is a valid chain of certificates since this cannot be checked using a pattern. Therefore, we extend constraint systems with *language constraints* of the form $u \Subset \mathcal{L}$ where $\mathcal{L}$ can be any language, that is, any set of terms. In particular, $\mathcal{L}$ will typically be a recursively defined set of terms. We provide in Sections 4 and 5 several examples of classes of recursive languages but for the moment $\mathcal{L}$ can be left unspecified.

Definition 3 (language constraint). *Let $\mathcal{L}$ be a language (i.e. a set of terms). An* $\mathcal{L}$-language constraint *associated to some constraint system $(\mathcal{C}, \mathcal{I})$ is an expression of the form $u_1 \Subset \mathcal{L} \wedge \ldots \wedge u_k \Subset \mathcal{L}$ where each u_i is a term such that $vars(u_i) \subseteq vars(\mathcal{C})$ and $st(u_i) \cap \mathcal{I} = \emptyset$.*

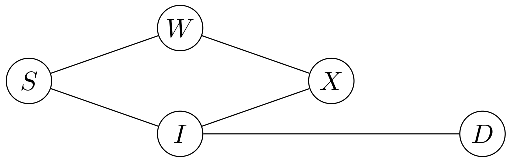
A solution *of a constraint system $(\mathcal{C}, \mathcal{I})$ and of an $\mathcal{L}$-language constraint $\phi = u_1 \Subset \mathcal{L} \wedge \ldots \wedge u_k \Subset \mathcal{L}$ is a ground substitution θ such that θ is a solution of $(\mathcal{C}, \mathcal{I})$ and $u_i\theta \in \mathcal{L}$ for any $1 \leq i \leq k$. We denote $st(\phi) = \{st(u_i) \mid 1 \leq i \leq k\}$.*

2.4 Example: the SMNDP protocol

The aim of the SMNDP protocol [12] is to find a path from a source node S towards a destination node D. Actually, nodes broadcast the route request to their neighbors, adding their name to the current path. When the request reaches the destination, D signs the route and sends the reply back over the network.

More formally, if D receives a request message of the form $\langle Req, S, D, Id, l\rangle$, where Id is a name (the identifier of the request) and l is the path built during the request phase, D will compute the signature $s_0 = [\![\langle D, S, D{::}l\rangle]\!]_{\mathsf{sk}(D)}$ and send back the reply $\langle Rep, D, S, D{::}l, [s_0]\rangle$. All nodes along the route then have to certify the route by adding their own signature. More precisely, during the reply phase, an intermediate node A_i receiving a message of the form $\langle Rep, D, S, l_{route}, [s_{i-1}, \ldots, s_0]\rangle$ would compute $s_i = [\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(A_i)}$ and send the message $\langle Rep, D, S, l_{route}, [s_i, \ldots, s_0]\rangle$. The list of signatures expected by S built over the list $l_{route} = [D, A_1, \ldots, A_n]$ is the list $l_{sign} = [s_n, \ldots, s_0]$ where $s_0 = [\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(D)}$ and $s_i = [\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(A_i)}$ for $1 \leq i \leq n$. We will denote by $\mathcal{L}_{\mathsf{SMNDP}}$ the set of messages of the form $\langle\langle S, D\rangle, \langle l_{route}, l_{sign}\rangle\rangle$.

Consider the following network configuration, where S is the source node, D is the destination node, X is an intermediate (honest) node, W is a node who has been compromised (*i.e.* the intruder knows the secret key $\mathsf{sk}(W)$), and I is a malicious node, *i.e.* a node controlled by the intruder.



An execution of the protocol where D is ready to answer a request and the source is ready to input the final message can be represented by the following constraint system:

$$\mathcal{C} = \begin{cases} T_0 \cup \{u_0, u_1\} \Vdash v_1 \\ T_0 \cup \{u_0, u_1, u_2\} \Vdash v_2 \end{cases}$$

with $T_0 = \{S, D, X, I, W, \mathsf{sk}(I), \mathsf{sk}(W)\}$ the initial knowledge of the intruder
$u_0 = \langle Req, S, D, Id, []\rangle$,
$u_1 = \langle Req, S, D, Id, [X, W]\rangle$,
$u_2 = \langle Rep, D, S, D{::}x_l, [[\![\langle D, S, D{::}x_l\rangle]\!]_{\mathsf{sk}(D)}]\rangle$,
$v_1 = \langle Req, S, D, x_{id}, x_l\rangle$,
$v_2 = \langle Rep, D, S, D{::}x_{route}, x_{sign}\rangle$

Let $\mathcal{I}$ be a non-empty set of names such that $st(\mathcal{C}) \cap \mathcal{I} = \emptyset$. We have that $(\mathcal{C}, \mathcal{I})$ is a constraint system. A solution to $(\mathcal{C}, \mathcal{I}) \wedge \langle\langle S, D\rangle, \langle D{::}x_{route}, x_{sign}\rangle\rangle \Subset \mathcal{L}_{\mathsf{SMNDP}}$ is *e.g.* the substitution $\theta = \{x_{id} \mapsto Id, x_l \mapsto [I; W], x_{route} \mapsto [I; W], x_{sign} \mapsto l_{sign}\}$ where:

- $l_{route} = [D, I, W]$, and
- $l_{sign} = [[\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(W)}; [\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(I)}; [\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(D)}]$.

This solution reflects an attack (discovered in [2]) where the attacker sends to the destination node D the message $\langle Req, S, D, Id, l\rangle$ with a false list $l = [I, W]$. Then D answers accordingly by $\langle Rep, D, S, l_{route}, [[\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(D)}]\rangle$. The intruder concludes the attack by sending to S the message $\langle Rep, D, S, l_{route}, l_{sign}\rangle$. This yields S accepting W, I, D as a route to D, while it is not a valid route.

3 Constraint solving procedure

As a first contribution, we provide a complete symbolic representation of the attacker knowledge, in the spirit of the characterization obtained in [1] in the passive case (that is, when the intruder only eavesdrops on the messages exchanged during the protocol execution). Revisiting the constraint solving procedure proposed in [11], we show that it is possible to compute a finite set $(\mathcal{C}_1, \mathcal{I}), \ldots, (\mathcal{C}_n, \mathcal{I})$ of solved forms whose solutions represent all the solutions of $(\mathcal{C}, \mathcal{I})$. This first result is an easy adaptation of the proof techniques of [11] to our richer term algebra. More importantly, we show that it is sufficient to consider the solutions of $(\mathcal{C}_i, \mathcal{I})$ that are obtained by applying composition rules only.

3.1 Simplification rules

Our procedure is based on a set of simplification rules allowing a general constraint system to be reduced to some simpler ones, called *solved*, on which satisfiability can be easily decided. A constraint system $(\mathcal{C}, \mathcal{I})$ is said to be solved if $\mathcal{C} \neq \bot$ and if each of its constraints is of the form $T \Vdash x$, where x is a variable. Note that the empty constraint system is solved. Solved constraint systems are particularly simple since they always have a solution. Indeed, let $N_0 \in \mathcal{I}$, the substitution τ defined by $x\tau = N_0$ for every variable x is a solution since $T\tau \cup \mathcal{I} \vdash x\tau$ for any constraint $T \Vdash x$ of the solved constraint system.

The *simplification rules* we consider are the following ones:

$$\mathsf{R_{ax}}: \quad (\mathcal{C} \wedge T \Vdash u, \mathcal{I}) \rightsquigarrow (\mathcal{C}, \mathcal{I}) \qquad \text{if } T \cup \{x \mid T' \Vdash x \in \mathcal{C}, T' \subsetneq T\} \vdash u$$

$$\mathsf{R_{unif}}: \quad (\mathcal{C} \wedge T \Vdash u, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}\sigma \wedge T\sigma \Vdash u\sigma, \mathcal{I}) \qquad \text{if } \sigma = mgu(t_1, t_2)$$
$$\text{where } t_1 \in st(T),\ t_2 \in st(T \cup \{u\}), \text{ and } t_1 \neq t_2$$

$$\mathsf{R_{fail}}: \quad (\mathcal{C} \wedge T \Vdash u, \mathcal{I}) \rightsquigarrow (\bot, \mathcal{I}) \qquad \text{if } vars(T \cup \{u\}) = \emptyset \text{ and } T \nvdash u$$

$$\mathsf{R_f}: \quad (\mathcal{C} \wedge T \Vdash \mathsf{f}(u, v), \mathcal{I}) \rightsquigarrow (\mathcal{C} \wedge T \Vdash u \wedge T \Vdash v, \mathcal{I}) \qquad \text{for } \mathsf{f} \in \mathcal{F} \smallsetminus \mathcal{F}_s$$

All the rules are indexed by a substitution (when there is no index then the identity substitution is implicitly considered). We write $(\mathcal{C},\mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}',\mathcal{I})$ if there are $\mathcal{C}_1,\dots,\mathcal{C}_n$ such that $(\mathcal{C},\mathcal{I}) \rightsquigarrow_{\sigma_0} (\mathcal{C}_1,\mathcal{I}) \rightsquigarrow_{\sigma_1} \dots \rightsquigarrow_{\sigma_n} (\mathcal{C}',\mathcal{I})$ and $\sigma = \sigma_n \circ \dots \circ \sigma_1 \circ \sigma_0$. Our rules are similar to those in [11] except for the rule $\mathsf{R}_{\mathsf{unif}}$. We authorize unification with a subterm of u and also with variables.

Soundness and termination are still ensured by [11]. To ensure termination in polynomial time, we consider the strategy $\mathcal{S}$ that consists of applying $\mathsf{R}_{\mathsf{fail}}$ as soon as possible, $\mathsf{R}_{\mathsf{unif}}$ and then R_{f}, beginning with the constraint having the largest right hand side. Lastly, we apply R_{ax} on the remaining constraints. We show that these rules form a complete decision procedure.

Theorem 1. *Let $(\mathcal{C},\mathcal{I})$ be a constraint system. We have that:*

- *Soundness: If $(\mathcal{C},\mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}',\mathcal{I})$ for some constraint system $(\mathcal{C}',\mathcal{I})$ and some substitution σ and if θ is a solution of $(\mathcal{C}',\mathcal{I})$ then $\theta\circ\sigma$ is a solution of $(\mathcal{C},\mathcal{I})$.*
- *Completeness: If θ is a solution of $(\mathcal{C},\mathcal{I})$, then there exist a constraint system $(\mathcal{C}',\mathcal{I})$ in solved form and substitutions σ, θ' such that $\theta = \theta' \circ \sigma$, $(\mathcal{C},\mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}',\mathcal{I})$ following the strategy $\mathcal{S}$, and θ' is a non-confusing solution of $(\mathcal{C}',\mathcal{I})$.*
- *Termination: If $(\mathcal{C},\mathcal{I}) \rightsquigarrow^n_\sigma (\mathcal{C}',\mathcal{I})$ following the strategy $\mathcal{S}$, then n is polynomially bounded in the size of $\mathcal{C}$. Moreover, the number of subterms of $\mathcal{C}'$ is smaller than the number of subterms of $\mathcal{C}$.*

Example 1. Consider our former example of a constraint system (see Section 2.4), we can simplify the constraint system $(\mathcal{C},\mathcal{I})$ following strategy $\mathcal{S}$:

- $\mathsf{R}_{\mathsf{unif}}$: $(\mathcal{C},\mathcal{I})\rightsquigarrow_\sigma(\mathcal{C}_1,\mathcal{I})$ with $\mathcal{C}_1 = \mathcal{C}\sigma$ where $\sigma = \{x_{id} \mapsto Id\}$,
- R_{f}: $(\mathcal{C}_1,\mathcal{I}) \rightsquigarrow^* (\mathcal{C}_2,\mathcal{I})$ with

$$\mathcal{C}_2 = \begin{cases} T_0 \cup \{u_0\sigma, u_1\sigma\} \Vdash \mathit{Req} & \wedge\ T_0 \cup \{u_0\sigma, u_1\sigma, u_2\sigma\} \Vdash \mathit{Rep} \quad \wedge \\ T_0 \cup \{u_0\sigma, u_1\sigma\} \Vdash S & \wedge\ T_0 \cup \{u_0\sigma, u_1\sigma, u_2\sigma\} \Vdash D \quad \wedge \\ T_0 \cup \{u_0\sigma, u_1\sigma\} \Vdash D & \wedge\ T_0 \cup \{u_0\sigma, u_1\sigma, u_2\sigma\} \Vdash S \quad \wedge \\ T_0 \cup \{u_0\sigma, u_1\sigma\} \Vdash \mathit{Id} & \wedge\ T_0 \cup \{u_0\sigma, u_1\sigma, u_2\sigma\} \Vdash x_{route} \quad \wedge \\ T_0 \cup \{u_0\sigma, u_1\sigma\} \Vdash x_l & \wedge\ T_0 \cup \{u_0\sigma, u_1\sigma, u_2\sigma\} \Vdash x_{sign} \end{cases}$$

- R_{ax}: $(\mathcal{C}_2,\mathcal{I}) \rightsquigarrow^* (\mathcal{C}',\mathcal{I})$ with

$$\mathcal{C}' = \begin{cases} T_0 \cup \{u_0\sigma, u_1\sigma\} \Vdash x_l & \wedge\ T_0 \cup \{u_0\sigma, u_1\sigma, u_2\sigma\} \Vdash x_{route} \\ & \wedge\ T_0 \cup \{u_0\sigma, u_1\sigma, u_2\sigma\} \Vdash x_{sign} \end{cases}$$

The constraint system $(\mathcal{C}',\mathcal{I})$ is in solved form, and we have that $\theta = \theta' \circ \sigma$ where $\theta' = \{x_l \mapsto [I;W], x_{route} \mapsto [I;W], x_{sign} \mapsto l_{sign}\}$ is a non-confusing solution of $(\mathcal{C}',\mathcal{I})$.

Compared to [11], we prove in addition that on the resulting solved constraint systems, we can restrict our attention to non-confusing solutions. Intuitively, we exploit the transformation rules such that any possible equality between subterms has already been guessed, thus ensuring that two distinct subterms do not map to the same term. Interestingly, non-confusing solutions of a solved constraint system enjoy a nice characterization.

3.2 A basis for deducible terms

We show that, for any non-confusing solution, any term deducible from the attacker knowledge may be obtained by composition only.

We first associate to each set of terms T the set of subterms of T that may be deduced from $T \cup \mathit{vars}(T)$. Note that on solved constraint systems, these variables are indeed deducible.

$$Sat_v(T) = \{u \in st(T) \mid T \cup \mathit{vars}(T) \vdash u\}$$

Proposition 1. *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system in solved form, θ be a non-confusing solution of $(\mathcal{C}, \mathcal{I})$, T be a left-hand side of a constraint in $\mathcal{C}$ and u be a term such that $T\theta \cup \mathcal{I} \vdash u$. We have that $Sat_v(T)\theta \cup \mathcal{I} \vdash u$ by using composition rules only.*

Proposition 1 states that it is possible to compute from a solved constraint system, a "basis" $Sat_v(T)$ from which all deducible terms can be obtained applying only composition rules. This follows the spirit of [1] but now in the active case.

This characterization is crucial in the remaining of the paper, when considering recursive tests. More generally, we believe that this characterization provides more modularity and could be useful when considering other properties such as checking the validity of a route or authentication properties.

We will also use the notion of *constructive solution* on constraint systems in solved form, which is weaker than the notion of non-confusing solution.

Definition 4 (constructive solution). *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system in solved form. A substitution θ is a* constructive solution *of $(\mathcal{C}, \mathcal{I})$ if for every deducibility constraint $T \Vdash x$ in $\mathcal{C}$, we have that $\mathcal{S}at_v(T)\theta \cup \mathcal{I} \vdash x\theta$ using composition rules only.*

A non-confusing solution of a solved system is a constructive solution, while the converse does not always hold. This notion will be used in proofs as we will transform solutions, preserving the constructive property but not the non-confusing property.

4 Link-based recursive languages

A chain of certificates is typically formed by a list of links such that consecutive links follow a certain relation. For example, the chain of public key certificates $[[\![\langle A_1, \mathsf{pub}(A_1)\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2)\rangle]\!]_{\mathsf{sk}(A_3)}; [\![\langle A_3, \mathsf{pub}(A_3)\rangle]\!]_{\mathsf{sk}(S)}]$ is based on the link $[\![\langle x, \mathsf{pub}(y)\rangle]\!]_{\mathsf{sk}(z)}$, and the names occurring in two consecutive links have to satisfy a certain relation. We provide a generic definition that captures such link-based recursive language.

Definition 5 (link-based recursive language). *Let m be a term built over variables of sort Base. A* link-based recursive language *$\mathcal{L}$ is defined by three terms*

w_0, w_1, w_2 of sort List such that $w_0 = [\mathsf{m}\theta_0^1; \ldots ; \mathsf{m}\theta_0^{k_0}]$, $w_i = \mathsf{m}\theta_i^1 :: \ldots :: \mathsf{m}\theta_i^{k_i} :: x^{\mathsf{m}}$ for $i = 1, 2$, and w_2 is a strict subterm of w_1.

Once w_0, w_1, w_2 are given, the language is recursively defined as follows. A ground term t belongs to the language $\mathcal{L}$ if either $t = w_0\sigma$ for some σ, or there exists σ such that $t = w_1\sigma$, and $w_2\sigma \in \mathcal{L}$.

Intuitively, w_0 is the basic valid chain while w_1 encodes the desired dependence between the links and w_2 allows for a recursive call.

Example 2. As defined in [13], X.509 public key certificates consist in chains of signatures of the form:

$$[[\![\langle A_1, \mathsf{pub}(A_1)\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2)\rangle]\!]_{\mathsf{sk}(A_3)}; \cdots ; [\![\langle A_n, \mathsf{pub}(A_n)\rangle]\!]_{\mathsf{sk}(S)}]$$

where S is some trusted server and each agent A_{i+1} certifies the public key $\mathsf{pub}(A_i)$ of agent A_i. These chained lists are all built from the term $\mathsf{m} = [\![\langle x, \mathsf{pub}(y)\rangle]\!]_{\mathsf{sk}(z)}$ with $x, y, z \in \mathcal{X}_{\mathsf{Base}}$. The set of valid chains of signatures can be formally expressed as the m-link-based recursive language $\mathcal{L}_{\mathsf{cert}}$ defined by:

$$\begin{cases} w_0 = [[\![\langle x, \mathsf{pub}(x)\rangle]\!]_{\mathsf{sk}(S)}], \\ w_1 = [\![\langle x, \mathsf{pub}(x)\rangle]\!]_{\mathsf{sk}(y)} :: [\![\langle y, \mathsf{pub}(y)\rangle]\!]_{\mathsf{sk}(z)} :: x^{\mathsf{m}}, \\ w_2 = [\![\langle y, \mathsf{pub}(y)\rangle]\!]_{\mathsf{sk}(z)} :: x^{\mathsf{m}}. \end{cases}$$

Similarly, link-based recursive languages can also describe delegation rights certificates in the context of distributed access-rights management. In [7] for example, the certificate chains delegating authorization for operation O are of the form:

$$[[\![\langle A_1, \mathsf{pub}(A_1), O\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2), O\rangle]\!]_{\mathsf{sk}(A_3)}; \ldots ; [\![\langle A_n, \mathsf{pub}(A_n), O\rangle]\!]_{\mathsf{sk}(S)}]$$

where S has authority over operation O and each agent A_{i+1} delegates the rights for operation O to agent A_i. These chained lists are all built from the term $\mathsf{m} = [\![\langle x, \mathsf{pub}(y), O\rangle]\!]_{\mathsf{sk}(z)}$ with $x, y, z \in \mathcal{X}_{\mathsf{Base}}$.

Example 3. In the recursive authentication protocol [19], a certificate list consists in a chain of encryptions of the form:

$$\begin{aligned} &[\mathsf{senc}(\langle K_{ab}, B, N_a\rangle, K_a); \mathsf{senc}(\langle K_{ab}, A, N_b\rangle, K_b); \\ &\ \mathsf{senc}(\langle K_{bc}, C, N_b\rangle, K_b); \mathsf{senc}(\langle K_{bc}, B, N_c\rangle, K_c); \ldots ; \mathsf{senc}(\langle K_{ds}, S, N_d\rangle, K_d)] \end{aligned}$$

where S is a trusted server distributing session keys K_{ab}, K_{bc}, ..., K_{ds} to each pair of successive agents via these certificates. These chained lists are all built from the term $\mathsf{m} = \mathsf{senc}(\langle y_1, y_2, y_3\rangle, z)$ with $y_1, y_2, y_3, z \in \mathcal{X}_{\mathsf{Base}}$. The set of valid chains of encryptions in this protocol can be formally expressed as the m-link-based recursive language $\mathcal{L}_{\mathsf{RA}}$ defined by:

$$\begin{cases} w_0 = [\mathsf{senc}(\langle z, S, x\rangle, x_k)], \\ w_1 = \mathsf{senc}(\langle z, x_a, x\rangle, x_{k_b}) :: \mathsf{senc}(\langle z, x_b, y\rangle, x_{k_a}) :: \mathsf{senc}(\langle z', x_c, y\rangle, x_{k_a}) :: x^{\mathsf{m}}, \\ w_2 = \mathsf{senc}(\langle z', x_c, y\rangle, x_{k_a}) :: x^{\mathsf{m}}. \end{cases}$$

We propose a procedure for checking for secrecy preservation for a protocol with link-based recursive tests in NP, for a bounded number of sessions.

Theorem 2. *Let $\mathcal{L}$ be a link-based recursive language. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system and ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$. Deciding whether $(\mathcal{C}, \mathcal{I})$ and ϕ has a solution is in NP.*

The proof of Theorem 2 involves three main steps. First, thanks to Theorem 1, it is sufficient to decide in polynomial (DAG) size whether $(\mathcal{C}, \mathcal{I})$ with language constraint ϕ has a non-confusing solution when $(\mathcal{C}, \mathcal{I})$ is a solved constraint system. Then, we show that we can (polynomially) bound the size of the lists in ϕ. This relies partly on Proposition 1, as it shows that a non-confusing solution is a constructive solution.

Proposition 2. *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system in solved form and ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$ where $\mathcal{L}$ is a link-based recursive language. Let θ be a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ϕ. Then there exists a constructive solution θ' of $(\mathcal{C}, \mathcal{I})$ and ϕ such that $\phi\theta'$ is polynomial in the size of $\mathcal{C}$, and ϕ.*

Proposition 2 is proved by first showing that there is a solution that uses a bounded number of distinct names. Thus there is a finite number of instances of m used in recursive calls, allowing us to cut the lists while preserving the membership to the recursive language.

The third step of the proof of Theorem 2 consists in showing that we can restrict our attention to solutions θ such that $x\theta$ is either a constant or a subterm of $\phi\theta$, by using Lemma 1. This lemma is a generic lemma that shows how any solution can be transformed by projecting some variables on constants. It will be reused in the next section.

Lemma 1. *Let $\mathcal{L}$ be a language, i.e. a set of terms. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system in solved form and ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$. Let θ be a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ϕ. Let N_0 be a name of Base sort in $\mathcal{I}$, and θ' be a substitution such that:*

$$\begin{cases} x\theta' = x\theta & \text{if } x\theta \in st(\phi\theta) \\ x\theta' = [] & \text{if } x \in \mathcal{X}_{\mathsf{List}} \text{ and } x\theta \notin st(\phi\theta) \\ x\theta' = N_0 & \text{if } x \notin \mathcal{X}_{\mathsf{List}} \text{ and } x\theta \notin st(\phi\theta) \end{cases}$$

The substitution θ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ϕ.

5 Routing protocols

Routing protocols typically perform recursive checks to ensure the validity of a given route. However, link-based recursive languages do not suffice to express these checks. Indeed, in routing protocols, nodes aim at establishing and certifying a successful route (*i.e.* a list of names of nodes) between two given nodes that wish to communicate. Each node on the route typically contributes to the routing protocol by certifying that the proposed route is correct, to the best of its knowledge. Thus each contribution contains a list of names (the route). Then the final node receives a list of contributions and needs to check that each contribution contains the same list of names, which has also to be consistent with the whole received message. For example, in the case of the SMNDP protocol [12], the source node has to check that the received message is of the form:

$$[[\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(A_n)}; \ldots ; [\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(A_1)}; [\![\langle D, S, l_{route}\rangle]\!]_{\mathsf{sk}(D)}]$$

where $l_{route} = [D; A_1; \ldots ; A_n]$.

5.1 Mapping-based languages

An interesting property in the case of routing protocols is that (valid) messages are uniquely determined by the list of nodes $[A_1; \ldots ; A_n]$ in addition to some parameters (*e.g.* the source and destination nodes in the case of SMNDP). We propose a generic definition that captures any such language based on a list of names.

Definition 6 (mapping-based language). *Let* b *be a term that contains no name and no* :: *symbol, and such that:*

$$\{w_1, w_1^p, \ldots, w_m^p\} \subseteq \mathit{vars}(\mathsf{b}) \subseteq \{w_1, w_2, w_3, w_1^p, \ldots, w_m^p\}.$$

The variables $w_1^p, \ldots, w_m^p$ *are the parameters of the language, whereas* w_1, w_2, *and* w_3 *are special variables. Let* $\mathcal{P} = \langle P_1, \ldots, P_m\rangle$ *be a tuple of names and* $\sigma_{\mathcal{P}} = \{w_1^p \mapsto P_1, \ldots, w_m^p \mapsto P_m\}$. *Let* $l = [A_1; \ldots ; A_n]$ *be a list of names, the links are defined over* l *recursively in the following manner :*

$$\mathsf{m}_{\mathcal{P}}(i, l) = (\mathsf{b}\sigma_{\mathcal{P}})\{w_1 \mapsto l, w_2 \mapsto A_i, w_3 \mapsto [\mathsf{m}_{\mathcal{P}}(i-1, l); \ldots ; \mathsf{m}_{\mathcal{P}}(1, l)]\}$$

The mapping-based *language (defined by* b*) is the following one:*

$$\mathcal{L} = \{\langle \mathcal{P}, \langle l, l'\rangle\rangle \mid \mathcal{P} = \langle P_1, \ldots, P_m\rangle \text{ is a tuple of names, } l = [A_1; \ldots ; A_n] \text{ a list of names, } n \in \mathbb{N}, \text{ and } l' = [\mathsf{m}_{\mathcal{P}}(n, l); \ldots ; \mathsf{m}_{\mathcal{P}}(1, l)]\}.$$

A mapping-based language is defined by a base shape b. The special variables w_2 and w_3 are optional and may not occur in b. Each element of the language is a triple $\langle \mathcal{P}, \langle l, l'\rangle\rangle$ where l' is a list of links entirely determined by the tuple $\mathcal{P} = \langle P_1, \ldots, P_m\rangle$ and the list l of arbitrary length n. In the list l', each link contains the same parameters $P_1, \ldots, P_m$ (*e.g.* the source and destination nodes), the list l of n names $[A_1; \ldots ; A_n]$ and possibly the current name A_i and the list of previous links, following the base shape b.
We illustrate this definition with two examples of routing protocols.

Example 4 (SMNDP protocol [12]). Recall that in SMNDP, the list of signatures expected by the source node S built over the list $l = [A_1, \ldots, A_n]$ is the list $[s_n, \ldots, s_1]$, where $s_i = [\![\langle D, S, l\rangle]\!]_{\mathsf{sk}(A_i)}$. This language has two parameters, the name of the source w_1^p and the name of the destination w_2^p. The language can be formally described with $\mathsf{b} = [\![\langle w_2^p, w_1^p, w_1\rangle]\!]_{\mathsf{sk}(w_2)}$.

Example 5 (endairA protocol [9]). The difference between SMNDP and endairA lies in the fact that during the reply phase, the intermediate nodes compute a signature over the partial signature list that they receive. In the endairA protocol, the list of signatures expected by the source node S built over the list of nodes $l = [A_1, \ldots, A_n]$ is the list $l'_s = [s_n, \ldots, s_1]$, where $s_i = [\![\langle D, S, l, [s_{i-1}; \ldots; s_1]\rangle]\!]_{\mathsf{sk}(A_i)}$.

This language has two parameters, the name of the source w_1^p and the name of the destination w_2^p. The language can be formally described with $\mathsf{b} = [\![\langle w_2^p, w_1^p, w_1, w_3\rangle]\!]_{\mathsf{sk}(w_2)}$.

5.2 Decision procedure

We propose a procedure for checking for secrecy preservation for a protocol with mapping-based tests in NP, for a bounded number of sessions.

Theorem 3. *Let $\mathcal{L}$ be a mapping-based language. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system and ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$.*
Deciding whether $(\mathcal{C}, \mathcal{I}) \wedge \phi$ has a solution is in NP.

The proof of Theorem 3 involves three main steps. First, thanks to Theorem 1, it is sufficient to decide in polynomial (DAG) size whether $(\mathcal{C}, \mathcal{I})$ with language constraint ϕ has a non-confusing solution when $(\mathcal{C}, \mathcal{I})$ is a solved constraint system. Due to Proposition 1, we deduce that it is sufficient to show that deciding whether $(\mathcal{C}, \mathcal{I}) \wedge \phi$ has a constructive solution is in NP, where $(\mathcal{C}, \mathcal{I})$ is a solved constraint system.

The second and key step of the proof consists in bounding the size of a constructive solution. Note that the requirement on the form of ϕ is not a restriction since any substitution satisfying ϕ will necessarily have this shape.

Proposition 3. *Let $\mathcal{L}$ be a mapping-based language. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system in solved form, ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$, and τ be a constructive solution of $(\mathcal{C}, \mathcal{I}) \wedge \phi$. We further assume that ϕ is of the form $u_1 \Subset \mathcal{L} \wedge \ldots \wedge u_k \Subset \mathcal{L}$ where $u_j = \langle\langle p_1^j, \ldots, p_m^j\rangle, \langle l_j, l'_j\rangle\rangle$.*

There exists a constructive solution τ' of $(\mathcal{C}, \mathcal{I}) \wedge \phi$ such that, for every j, the length of $l_j\tau'$ is polynomially bounded on the size of $\mathcal{C}$ and ϕ.

For each constraint $\langle\langle p_1^j, \ldots, p_m^j\rangle, \langle l_j, l'_j\rangle\rangle \Subset \mathcal{L}$, the list l_j provides constraints on the last elements of the list l'_j, while l'_j provides constraints on the last elements of the list l_j. The main idea of the proof of Proposition 3 is to show that it is possible to cut the middle of the list l_j, modifying the list l'_j accordingly. This is however not straightforward as we have to show that the new substitution is still a solution of the constraint system $(\mathcal{C}, \mathcal{I})$. In particular, cutting part of the list might destroy some interesting equalities, used to deduce terms. Such cases are actually avoided by considering constructive solutions and by cutting at some position in the lists such that none of the elements are subterms of the constraint, which can be ensured by combinatorial arguments.

Proposition 3 allows us to bound the size of $l_j\theta$ for a minimal solution θ, which in turn bounds the size of $l'_j\theta$. The last step of the proof of Theorem 3 consists in showing that any $x\theta$ is bounded by the size of the lists or can be replaced by a constant, by applying Lemma 1.

6 Conclusion

We have provided two new NP decision procedures for (automatically) analysing confidentiality of security protocols with recursive tests, for a bounded number of sessions. The classes of recursive languages we can consider both encompass chained-based lists of certificates and most of the recursive tests performed in the context of routing protocols. These procedures rely on a new characterization of the solutions of a constraint system, extending the procedure for solving constraint systems. We believe that this new characterization is of independent interest and could be used for other families of protocols.

As further work, we plan to implement our procedure, which will require us to optimize it. We also plan to consider larger classes of recursive languages in order to capture *e.g.* the recursive tests performed in the context of group protocols. It would also be interesting to see whether our techniques could be extended for analysing protocols that use such recursive languages not only for performing tests but also as outputs in protocols.

References

1. M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, 2006.
2. T. R. Andel and A. Yasinsac. Automated security analysis of ad hoc routing protocols. In *Proc. of the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'07)*, pages 9–26, 2007.
3. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. of the 17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.
4. A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In *Proc. of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, pages 1–10, 2008.
5. M. Arnaud, V. Cortier, and S. Delaune. Deciding security for protocols with recursive tests. Research Report LSV-11-05, Laboratoire Spécification et Vérification, ENS Cachan, France, Apr. 2011. 46 pages.

6. N. Asokan and P. Ginzboorg. Key agreement in ad hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.
7. T. Aura. Distributed access-rights management with delegation certificates. In *Secure Internet Programming*, volume 1603 of *LNCS*, pages 211–235. 1999.
8. B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *Proc. of the 20th International Conference on Automated Deduction (CADE-20)*, 2005.
9. L. Buttyán and I. Vajda. Towards Provable Security for Ad Hoc Routing Protocols. In *Proc. of the 2nd ACM workshop on Security of ad hoc and sensor networks (SASN'04)*, pages 94–105. ACM, 2004.
10. N. Chridi, M. Turuani, and M. Rusinowitch. Decidable analysis for a class of cryptographic group protocols with unbounded lists. In *Proc. of the 22nd IEEE Computer Security Foundations Symposium (CSF'09)*, pages 277–289, 2009.
11. H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties for cryptographic protocols. Application to key cycles. *ACM Transactions on Computational Logic (TOCL)*, 11(4):496–520, 2010.
12. T. Feng, X. Guo, J. Ma, and X. Li. UC-Secure Source Routing Protocol, 2009.
13. R. Housley, W. Ford, and W. Polk. X.509 certificate and CRL profile, 1998. IETF standard, RFC 2459.
14. Y.-C. Hu, A. Perrig, and D. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. *Wireless Networks*, 11:21–38, 2005.
15. R. Küsters and T. Truderung. On the Automatic Analysis of Recursive Security Protocols with XOR. In *Proc. of the 24th Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *LNCS*, pages 646–657. Springer, 2007.
16. R. Küsters and T. Wilke. Automata-based Analysis of Recursive Cryptographic Protocols. In *Proc. of the 21st Symposium on Theoretical Aspects of Computer Science (STACS'04)*, volume 2996 of *LNCS*, pages 382–393. Springer-Verlag, 2004.
17. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, 1996.
18. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. of the 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 166–175, 2001.
19. L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. of the 10th IEEE Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.
20. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190. IEEE Computer Society Press, 2001.
21. T. Truderung. Selecting theories and recursive protocols. In *Proc. of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *LNCS*, pages 217–232. Springer, 2005.