PROUVÉ
Protocoles cryptographiques : Outils de vérification automatique

Véronique CORTIER, Stéphanie DELAUNE, Cristian ÉNÉ, Francis KLAY, Yassine LAKHNECH, Max MISSONGO, Michael RUSINOWITCH, Laurent VIGNERON, Ralf TREINEN, Indra WATI

Compte rendu de fin de recherche d'opération d'une recherche financée par le ministère de la jeunesse, de l'éducation et de la recherche.

Date : 12 septembre 2008

Décision d'aide numéro 03V358, 03V359, 03V360, 03V361, 03V362

Noms des organismes bénéficiaires :
- CNRS - École Normale Supérieure de Cachen
- CRIL Technology
- France Télécom
- INRIA Lorraine
- Université Grenoble 1, Joseph Fourier

Coordinateur général : Michael RUSINOWITCH
INRIA Lorraine
Technopole de Nancy-Brabois
615, rue du jardin Botanique
54602 Villers-les-Nancy

Responsable scientifique : Ralf TREINEN
Laboratoire Spécification et Vérification
ENS de Cachan
61, avenue du Président Wilson
94235 CACHAN Cedex

Numéro d'identification

Résumé

Un protocole cryptographique est un programme distribué exécuté par des agents, et dont les parties communiquent via un canal qui est ouvert à l'écoute, ou à la manipulation par des tiers. Un tel programme peut se servir de certaines primitives cryptographiques (comme par exemple le chiffrement, ou la signature) pour assurer les propriétés de sécurité souhaitées, comme par exemple le secret des messages échangés, ou l'authentification des agents.

L'objectif du projet Prouvé était d'avancer l'état de l'art dans la vérification automatique et symbolique de protocoles cryptographiques. Cette technique de vérification permet de trouver des failles de conception logique d'un protocole, comme par exemple la vulnérabilité par une attaque de « l'homme au milieu ».

Les contributions essentielles du projets concernent le développement d'un langage formel pour la spécification de protocoles, la mise au point et l'intégration de plusieurs outils de vérification automatique, et des nouvelles techniques de vérification automatique qui permettent de prendre en compte des propriétés réelles de primitives cryptographiques. Ces contributions étaient motivées par des études de cas venant d'un partenaire industriel.

Mots clefs : Sécurité, vérification automatique, protocoles cryptographiques, affaiblissement de l'hypothèse du chiffrement parfait.

Table des matières

1 Tâche 1 : Sémantique des protocoles et des propriétés

1.1 Le langage Prouvé

The purpose of the PROUVÉ protocol specification language is to give means to describe both protocols and the context in which they are used. It should allow to specify a signature of message constructors together with equational axioms defining the semantics of message constructors.

The sub-language used to define protocols is oriented on existing imperative programming languages. Other projects, like EVA, choose to use a more abstract, supposedly user-friendly language using the so-called Alice-Bob notation. This approach raises the need for a compiler from the user language to some kind of programming language that serves as input language for the verification tools. It is useful to understand how a correct protocol run is supposed to look like but does not describe in an unambiguous way the actions of the protocol participants. We felt that this approach hides from the shortcomings of the Alice-Bob notation, and choose to have the user specify the protocol as seen by the protocol agents, rather then from the point of view of an outside observer of a correct protocol execution. As a consequence, this sub-language describes *roles* which are the programs executed by (legitimate) protocol participants.

We used constructs from imperative programming languages as building blocks for the sub-language describing roles. However, our language is deliberately not Turing-complete in that we excluded constructs that are either not useful for describing protocols or too complex to handle by our verification tools. The basic instructions are sending and receiving messages and pattern matching. Instructions are composed by serial composition and conditional branching. All variables have to be explicitly declared (either as local variables, or as parameters). Variables can be write-once (that is, logical variables) or mutable variables.

Roles are composed in so-called *Scenarios*. The sub-language for scenarios contains constructs for parallel and sequential composition, and non-deterministic choice of values. It has assignment statements, and allows to instantiate roles that have been previously defined.

The accompanying assertion language allows to express safety properties of execution traces of protocol scenarios.

The design of the PROUVÉ specification language has been inspired by the languages muCAPSL [75] and CASRUL [58].

1.2 Syntax of Protocol Specification Files

A protocol specification consists of five main sections some of which are optional.

1. The *signature* section is optional in a protocol specification. Its purpose is to extend the default signature by user-defined function symbols. The

default signature contains the types `int`, `bool`, `nonce`, `algo` (the enumeration type of asymmetric encryption algorithm), `symalgo` (the same for symmetric encryption), `pubkey`, `privkey`, `symkey`, `principal`, as well as a universal type `message`. In addition to these basic types there are polymorphic type constructors for lists and association lists. There is a number of predefined function symbols, for instance for message encryption and decryption.

2. The *axioms* is optional in a protocol specification. Its purpose is to give a semantics to the user-defined symbols through equational axioms.

3. The *roles* section is mandatory, its purpose is to define the different roles which are executed by the protocol participants.

   A *role definition* consists of the keyword `role` followed by the name of the role, a non-terminated declaration list surrounded by parentheses which serves to specify the list of parameters, and an instruction block. An *instruction block* consists of an optional variable declaration (which, if present, consists of the keyword `declare` followed by a declaration list) and an instruction list surrounded by `begin` ... `end`.

   An *instruction list* is a possibly empty list of instructions that are terminated by a semi-colon. An *instruction* is a *raw instruction* which may be preceded by an identifier which, in this case, is separated from the following raw instruction by a colon.

   A *raw instruction* is either an instruction block, a *send* instruction, a *receive* instruction, a `fail` instruction, an *assignment instruction*, a *new* instruction, a *makekey* instruction, a *match* instruction, or a composite instruction. A composite instruction is either a conditional instruction `if` ... `then` ... `else` ... `fi` where the *else* part is optional, a *choice* instruction which consists of a non-empty sequence of instructions lists which are separated by the symbol | and surrounded by the keywords `choice` and `end`, or a case instruction. A *case instruction* consists, besides the keywords, of an expression, a non-empty list of cases which are terminated by semi-colons, and a final instruction list. A *case* is a pair consisting of a pattern sequence and an instruction list, joined by the symbol `->`.

4. The section on *global variables* is optional, its purpose is to declare the variables which are used in the scenario. These variables may also take the form of multi-dimensional arrays.

5. The *scenario* section is mandatory, its purpose is to define how the role instances are instantiated, and how they interact through global variables.

   A *scenario* consists of a scene instruction surrounded by the keywords `scenario` and `end`. A *scene instruction* is a raw scene instruction, possibly prefixed by an identifier. A *raw scene instruction* is one of a role instantiation, a call to `new` or `makekey`, an assignment (where the left-hand side may be any l-expression, contrary to assignment in roles where the left-hand side must be an identifier), a conditional expression (with an optional `else` part), an infinite loop `forever` the body of which is a scenario list, a `parallel` instruction, the body of which is a non-empty list of scenario lists which are separated by the character |, an existential quantification consisting of a value declaration and a scenario, a universal declaration which has an analogous form as an existential quantification but may carry in addition a relational constraint, and a block which consists of a local declaration part and a scenario list.

A *scenario list* is a possibly empty list of scenarios which are terminated by semi-colons. A *simple value declaration* is a pair of a non-empty list of identifiers and a type expression, and a *value declaration* is a non-empty list of simple value declarations which are separated by commas. A *relational constraint* is a non-empty sequence of intervals which are separated by commas. An *interval* is either an omega-number or a pair of omega numbers. An *omega number* is either a positive integer or the symbol $*$.

As an example we give here the specification of the Needham-Schroeder protocol with Lowe's patch:

```
# The Needham-Schroeder Protocole with Lowe patch

# From the Spore repository :
#
# A,B,S :        Principal
# Na,Nb :        Nonce
# KPa,KPb,KPs,KSa,KSb,KSs :      Key
# KPa,KSa :      is a key pair
# KPb,KSb :      is a key pair
# KPs,KSs :      is a key pair
#
# 1.    A      →      S      :      A,B
# 2.    S      →      A      :      {KPb, B}KSs
# 3.    A      →      B      :      {Na, A}KPb
# 4.    B      →      S      :      B,A
# 5.    S      →      B      :      {KPa, A}KSs
# 6.    B      →      A      :      {Na, Nb, B}KPa
# 7.    A      →      B      :      {Nb}KPb

signature
        alice, bob, intruder: principal;
        init_connection_with: principal → message;
        pubkey_of: principal → pubkey;
end

# we assume that Alice initiates a session on receiving a request
# of the form init_connection_with(name)
role Alice (my_name: principal; my_key: privkey;
            server_signkey: pubkey)
```

```
declare
        my_nonce, bob_nonce: nonce;
        bob_pubkey: pubkey;
        bob_name: principal;
        begin
                recv(init_connection_with(bob_name));
                send([my_name,bob_name]);
                recv(sign(asym,inv(server_signkey),[bob_pubkey,bob_name]));
                new(my_nonce);
                send(crypt(asym,bob_pubkey,[my_nonce,my_name]));
                recv(crypt(asym,inv(my_key),[my_nonce,bob_nonce,bob_name]));
                send(crypt(asym,bob_pubkey,bob_nonce));
        end

role Bob (my_name: principal; my_key: privkey;
   server_signkey: pubkey)
        declare
        my_nonce, alice_nonce : nonce;
        alice_name: principal;
        alice_pubkey: pubkey;
        begin
                recv(crypt(asym,inv(my_key),[alice_nonce,alice_name]));
                send([my_name,alice_name]);
                recv(sign(asym,inv(server_signkey),[alice_pubkey,alice_name]));
                new(my_nonce);
                send(crypt(asym,alice_pubkey,[alice_nonce,my_nonce,my_name]));
                recv(crypt(asym,inv(my_key),my_nonce));
        end

role Server (my_signkey: privkey)
        declare
        requester, query: principal;
        begin
                recv([requester,query]);
                send(sign(asym,my_signkey,[pubkey_of(query),query]));
        end

role Send_initial_request (name: principal)
        begin
                send(init_connection_with(name));
        end

variables
        alice_key, bob_key, server_key: privkey;
end

scenario
        begin
                makekey(asym, server_key);
                makekey(asym, alice_key);
```

```
                    makekey(asym, bob_key);
                    Send_initial_request(bob);
                    parallel
                            Alice (alice,alice_key, inv(server_key))
                            | Bob (bob, bob_key, inv(server_key))
                            | forever Server(server_key) end
                    end
            end
end
```

The complete definition of the context-free syntax of the PROUVÉ language, as well as the pertaining static semantics checks (like typing, declarations of variables, etc.) can be found in [60].

1.3 Semantics of Protocol Specifications

The semantics of the PROUVÉ language is formally defined by a set of rules in the style of structured operational semantics.

The semantics is described in a modular way, in that the semantics for scenarios is based on some semantics for roles that is given by a transition system on role configurations, but without assuming any particular properties of the semantics of roles. Also, the semantics of roles is defined while assuming a model of an intruder and a class of algebras that are the semantics of the equational axioms without making any particular assumption on them.

Other noticeable choices that have been made are the following.

Scopes There are several choices that can be made to reflect and take care of the scope of variables. The choice made in our semantic definition is to keep two levels of scoping. We flatten the scopes introduced at scenarios level but keep the scopes at role level. Variable renaming and α-conversion are needed to maintain coherence of variable bindings.

Lazy evaluation of the `forall` and iteration commands The `forall` operator (written $\forall$ in our abstract syntax) can be unfolded at once or in a lazy manner introducing one new scenario in parallel at a time. The same remark applies to iteration. The lazy interpretation is chosen in our semantics. Concerning the `forall` operator, the unfolded scenario is chosen non-deterministically.

Initial values and variable initialization A particularity of the communication in cryptographic protocols is that the reception of a message may assign values to some variables while the values of the remaining variables are used for synchronization. In fact, variables that have not yet been assigned a value are of the first category while the other variables are of the second. So, one has to keep track in a way or another which variables have already a value assigned to them. There are different ways how to keep this information also depending upon how one wants to handle initial values.

The choice made in our semantics is that variables not yet affected by a receive or an assignment are not in the domain of the functions that associate values to variables.

The formal definition can be found in [60].

1.4 A Logics for Prouvé Protocol Specifications

We have defined a logics for expressing security assertions of PROUVÉ protocol specifications. These are so-called *trace properties*: The logics can express the fact that a certain property is true for all possible execution traces of a protocol. For instance, the secrecy property of the Needham-Schroeder protocol listed above is

```
uses
  "needham-schroeder.prv"

public
  alice, bob, intruder, init_connection_with, pubkey_of

initial
  x_M = [[ inv(pubkey_of(intruder)) ]]

always
  (forall P:Bob . P.alice_name != intruder -> issecret(P.my_nonce))
  &
  (forall P:Alice . P.bob_name != intruder -> issecret(P.my_nonce))
```

In this assertion, the initial condition expresses the fact that the intruder initially knows his own private key. The formula in the `always` section expresses the property that is required to hold for all points of all possible execution traces of the protocol: The intruder cannot obtain the nonce of any principal P playing the role of Bob provided that P is not playing the protocol with the intruder in the role of Alice (and the symmetrical assertion for Alice).

An exact definition of the syntax and semantics of the logics is given in [60]. There its is also shown that our logics is strong enough to express the different versions of authentication described in [70].

1.5 Traductions et simplifications

1.5.1 Traduction de PROUVÉ vers Casrul

Casrul est un sous-système du logiciel AVISPA [1], comprenant le traducteur du langage haut-niveau HLPSL vers le langage bas-niveau IF d'AVISPA et les outils de vérification développés dans l'équipe CASSIS (TA4SP et AtSe). Dans le contexte du projet PROUVÉ, nous avons travaillé sur la conception d'un nouveau traducteur, permettant de fournir aux outils de vérification des protocoles

[1] `http://www.avispa-project.org`

décrits dans le langage de spécification de PROUVÉ, ce dernier permettant de considérer des protocoles plus complexes que le langage d'AVISPA ainsi que des scénarios beaucoup plus évolués.

Les principales étapes de cette réalisation sont décrites ci-dessous.

Types, constantes et variables : La conversion des principaux types n'a pas posé de problème, seules les listes et les tables ne sont pas prises en compte par le traducteur. En ce qui concerne les constantes et les variables, un renommage systématique a été nécessaire.

Rôles : Chaque rôle PROUVÉ est traduit en plusieurs règles de transition, décrivant les changement d'états de l'agent jouant le rôle. Pour les cas les plus simples, chaque instruction du rôle est traduite en une règle. Ceci s'applique pour les instructions d'envoi et de réception de messages, l'affectation, la création d'un nonce et l'échec (`fail`).

Pour les instructions plus évoluées, la traduction s'est avérée plus délicate. Ainsi, l'instruction `choice` est traduite en un ensemble de règles décrivant les différents points d'entrée possibles, le choix de l'un d'entre eux restant non-déterministe. Pour l'instruction `if...then...[else...] fi`, les blocs d'instructions des parties *then* et *else* sont traduits en ensembles de règles, mais la difficulté a résidé dans la traduction de la condition : les conditions dans PROUVÉ étant beaucoup plus évoluées, il a fallu convertir ces expressions booléennes en expressions n'utilisant que les opérateurs disponibles dans le langage IF et les mettre en forme normale disjonctive ; l'expression booléenne obtenue (ainsi que sa négation) sont alors traduites en un ensemble de règles, et au final un branchement est fait vers le bloc d'instructions approprié.

Scénarios : Dans une spécification PROUVÉ, les scénarios décrivent comment les instances de rôles se combinent. La langage permettant des combinaisons en parallèle, en séquence et des boucles infinies, il s'est agit là de l'une des tâches les plus difficiles à réaliser, car la langage IF n'accepte que des instances rôles en parallèle. Par exemple, l'enchaînement en séquence d'instances de rôles a été réalisé grâce à la création automatique par le traducteur de rôles supplémentaires, chargé de décrire les combinaisons de rôles à réaliser, et de bien décrire leur état initial.

Ce traducteur a permis d'utiliser des outils de vérification existants, comme AtSe et TA4SP, sans leur apporter aucune modification, mais en élargissant le spectre de protocoles analysables. Les nombreuses expérimentations ont été très positives. Il faut noter également que ce travail permet aussi d'utiliser sans aucun effort supplémentaire tout autre outil d'analyse connecté à AVISPA, comme OFMC et SATMC, par exemples.

1.5.2 Traduction de PROUVÉ vers Hermes

En mythologie grecque, Hermes est le gardien des secrets et le postier des dieux. De nos jours, HERMÈS est également un outil consacré à la vérification des propriétés de sécurité des protocoles cryptographiques. Il a été développé à Verimag, dans l'équipe "Systèmes Répartis et Complexes". Nous avons connecté HERMÈS au langage de spécifications de protocoles et des propriétés PROUVÉ. Pour cela, nous avons implanté des nouveaux fonctionnalités dans HERMÈS afin de permettre la prise en compte de la plupart des constructions du langage PROUVÉ. L'outil, ainsi que quelques exemples sont disponible en ligne à l'adresse http ://www-verimag.imag.fr/∼async/hermes/prouve/hermes.php .

Nous avons apporté une attention particulière au respect de la sémantique du langage PROUVÉ. Ainsi, nous avons dû resoudre plusieurs difficultés pour traiter entièrement le langage. Les point importants sont décrits ci-dessous.

Types de donnés Les listes, les tables d'associations et les entiers sont des types de données qui n'ont pas un correspondant parmi les termes manipules par notre outil. En principe, cette limitation peut sembler très restrictive dans la description des protocoles. Cependant, à retenir que dans la plus part des cas nous sommes arrivés a décrire les protocoles cryptographiques en contournant ces types de données.

Rôles : Chaque rôle PROUVÉ est traduit en plusieurs régles de transition. A une pair (liste de receive, send) correspond une règle de réécriture (une transition). L'ordre d'exécution des instructions d'un rôle est bien préserve par la traduction vers des règles de réécriture (ce qui n'etait pas le cas dans l'ancienne version de HERMÈS).

Pour illustrer la traduction, considérons le rôle d'un des participants dans le protocole Needham-Schroeder.

```
role Alice (A: principal; B: principal; SKa: privkey)
declare
Na, v_Nb: message;
begin
new(Na);
send(crypt(asym,PK(B),[Na,A]));
recv(crypt(asym,PK(A),[Na,v_Nb]));
send(crypt(asym,PK(B),v_Nb)) ;
end
```

L'appel Alice(alice,bob,inv(PK(alice))) dans un scénario quelconque sera traduit vers les deux règles de réécriture suivantes.

```
       null                          {(Na,v_Nb)}_PK(alice)
------------------- (Alice_1)       ---------------------- (Alice_3)
{(Na,alice)}_PK(bob)                  {v_Nb}_PK(bob)
```

Dans ces règles, Na est une valeur fraîche, et v_Nb est une variable.

Conditions Dans la description de rôles, mais aussi dans la description des propriétés il est possible des spécifier des conditions dans tout leur complexité (conjonction, disjonction, négation). Dans HERMÈS nous traitons complètement les testes d'égalité et aussi dans un cas restreint la négation d'égalité (quand on peut énumérer le complément).

Sénarios : Dans une spécification PROUVÉ, les scénarios décrivent comment les instances de rôles se combinent. Le langage permettant des combinaisons en parallèle, en séquence et des itérations et des mises en parallèles infinies, les techniques de preuve de HERMÈS ont dû être étendu pour prendre en compte ce pouvoir d'éxpression. HERMÈS traduit un scénario par une éxpression régulière sur les transitions du protocole.

Les assertions qui peuvent être traitées par HERMÈS sont des secrets, avec des conditions qui sont des conjonctions d'égalités et de non égalités restreintes. De plus, pour exprimer ces propriété nous avons besoin de savoir lesquels parmi les participants sont honnôtes et lesquels sont malhonnôtes. Cette information est dérivée des connaissances initiales de l'intrus.

1.5.3 Traduction de PROUVÉ vers Actas

ACTAS is an integrated system for manipulating associative and commutative tree automata (AC-tree automata for short). It has various functions such as for Boolean operations of AC-tree automata, computing rewrite descendants, and solving emptiness and membership problems. In order to deal with high-complexity problems in reasonable time, over- and under-approximation algorithms are provided. This functionality enables automated verification of safety property in infinite state models. This is useful in the domain of, e.g. network security, in particular, for security problems of cryptographic protocols using equational properties of cryptographic primitives.

In runtime of model construction, a tool support for analysis of state space expansion is provided. The intermediate status of the computation is displayed in numerical data table, and the line graphs are generated. Besides, a graphical user interface of the system provides a user-friendly environment for convenient use.

The ACTAS tool is developed by Hitoshi Ohsaki and Toshinori Takai at the National Institute of Advanced Industrial Science and Technology (AIST), Kansai branch, Japan [78]. It is available at `http://staff.aist.go.jp/hitoshi.ohsaki/actas/index.html`.

In a common work with Hitoshi Ohsaki we have identified the so-called *stateless fragment* of PROUVÉ, and we have given a translation of PROUVÉ protocol specifications of that fragment to the ACTAS verification tool. The most important restrictions of the stateless fragment with respect to the full PROUVÉ language are:

1. Roles: in the stateless fragment we allow only two kinds of instructions:

- a send instruction of the form `send(s)` where `s` is a term, possibly containing parameters of the role. Its semantic is to send the value of this term on the public channel.
- a receive-send block, consisting of
  - an optional declaration of local variables to this block, which must in our case all by logical (write-once) variables;
  - a receive-instruction of the form `recv(r)`, where `r` may contain parameters of the role or variables declared to this block;
  - a send-instruction `s` as before, except that `s` may now also contain variables local to that block which have been instantiated by the preceding receive-instruction.

  The semantics of this block is that on reception of a term matched by r with matching substitution σ the term $s\sigma$ is send on the public channel.

2. Global variables: Skolemized variables are not allowed in the stateless fragment.

3. Scenarios: In the stateless fragment, this section consists of an (possibly empty) sequence of `new` instructions (a `new`(x) instruction assigns a fresh value to the variable x), followed by a parallel execution of several role invocations. A role invocation consists of the name of a role and a list of formal parameters, a formal parameter being a term possibly containing variables.

Assertions are restricted to

1. In case of the stateless fragment, the initial condition is a just a finite list of ground terms which describe the initial knowledge of an intruder.

2. In case of the stateless fragment, the safety formula is of the form `issecret(t)` where t is a ground term, meaning that the intruder is not able to deduce the term t.

For example, Figure 3 shows the translation of the Diffie-Hellman protocol given in Figure 2 and its pertaining security assertion given in Figure 1.

More details of the translation can be found in [77].

```
uses
  "diffie-helman.prv"

public
  kap

initial
  x_M = [[ g ]]

always
  issecret(secret)
```

Figure 1: Security assertion of the Diffie-Hellman protocol of Figure 2.

```
signature
  kap: (symkey, int) → symkey;
  mult: (int, int) → int;
  g: symkey;
  secret: message;
  alice_expo, bob_expo: int;
end

axioms
  declare
    x, y, z : int;
  begin
    kap(kap(g,x),y) = kap(g,mult(x,y));
    mult(x,mult(y,z)) = mult(mult(x,y),z);
    mult(x,y) =mult(y,x);
  end

role Alice(my_expo: int)
  begin
    send(kap(g,my_expo));
    declare
      bob_message: symkey;
    begin
      recv(bob_message);
      send(symcrypt(sym,kap(bob_message,my_expo),secret));
    end;
  end

role Bob(my_expo: int)
  begin
    declare
      alice_message: symkey;
    begin
      recv(alice_message);
      send(kap(g,my_expo));
    end;
  end

scenario
    parallel
      Alice(alice_expo) | Bob(bob_expo)
    end

end
```

Figure 2: The Diffie-Hellman protocol in PROUVÉ.

```
[Comment]
  Translation of assertion diffie-helman.pra
  Translation options:
[Signature]
  const: 0, true, false, sym, asym, rsa, des, g, secret, alice_expo, bob_expo
  var: alice_message, bob_message, my_expo, x, y, z
  AC: mult
[R-rule: TRS1]
  # Actions of Alice
  S_Alice(my_expo) -> kap(g,my_expo)
  RS_Alice(my_expo,bob_message) -> symcrypt(sym,kap(bob_message,my_expo),secret)
  # Actions of Bob
  RS_Bob(my_expo,alice_message) -> kap(g,my_expo)
  # Dolev-Yao simplification roles
  inverse(inverse(x)) -> x
  decrypt(x,y,crypt(x,inverse(y),z)) -> z
  decrypt(x,inverse(y),crypt(x,y,z)) -> z
  symdecrypt(x,y,symcrypt(x,y,z)) -> z
  # Rewriting rules corresponding to the equational axioms
  kap(kap(g,x),y) -> kap(g,mult(x,y))
  kap(g,mult(x,y)) -> kap(kap(g,x),y)
[T-rule(q_message): TAintruder]
  # initial intruder knowledge
  g -> q_symkey
  # function symbols known to the intruder
  inverse(q_pubkey) -> q_privkey
  inverse(q_privkey) -> q_pubkey
  inverse(q_symkey) -> q_symkey
  decrypt(q_algo,q_privkey,q_message) -> q_message
  crypt(q_algo,q_pubkey,q_message) -> q_message
  symdecrypt(q_symalgo,q_symkey,q_message) -> q_message
  symcrypt(q_symalgo,q_symkey,q_message) -> q_message
  s(q_int) -> q_int
  0 -> q_int
  true -> q_bool
  false -> q_bool
  sym -> q_symalgo
  des -> q_symalgo
  asym -> q_algo
  rsa -> q_algo
  kap(q_symkey,q_int) -> q_symkey
  # the type lattice
  q_int -> q_message
  q_symalgo -> q_message
  q_symkey -> q_message
  # triggering actions
  alice_expo -> q_aux_1
  S_Alice(q_aux_1) -> q_symkey
  alice_expo -> q_aux_2
  RS_Alice(q_aux_2,q_symkey) -> q_message
  bob_expo -> q_aux_3
  RS_Bob(q_aux_3,q_symkey) -> q_symkey
[T-rule(q_secret): TAsecret]
  secret -> q_secret
```


Figure 3: Translation into ACTAS.

2 Tâche 2 : Présentation des résultats

Dans le cadre du projet RNTL PROUVÉ, Cril Technology était principalement responsable de la tâche 2 « Présentation des résultats ».

La première partie du projet a été consacrée à la réalisation de l'état de l'art des outils de vérification des protocoles cryptographiques. Nous en avons répertorié et mis en œuvre cinq principaux qui ont des approches différentes : ils s'intéressent tous à prouver des propriétés de secret mais certains sont consacrés à la recherche d'attaques alors que d'autres se concentrent sur la recherche de preuves. La liste des outils identifiés et testés est CASRUL, HERMES, ISPY (H1).

Nous avons également travaillé à la compréhension des algorithmes mis en œuvres dans ces différents outils (techniques de réécriture), pour anticiper sur le développement d'un simulateur permettant de vérifier le bien fondé des attaques générées (en offrant à l'utilisateur des outils lui permettant de détecter les fausses attaques générées par sur-approximation au niveau des algorithmes). L'outil TOM permettant un prototypage rapide d'algorithmes exprimés sous formes de règles de réécriture déjà utilisé pour simuler certains protocoles cryptographiques a été évalué et semble un bon candidat pour la mise en œuvre du simulateur en permettant la compilation des règles dans un langage impératif type C ou Java.

La seconde étape de cette tâche consistait à définir un langage permettant de représenter de manière simple et précise les preuves et les attaques sur les protocoles cryptographiques. Le langage MSC qui est normalisé semble convenir parfaitement à cette tâche. Nous avons donc décrit l'utilisation de ce langage et précisé les adaptations nécessaires dans le cadre du projet PROUVÉ.

Voici les modifications nécessaires à apporter aux formalismes MSC et au package LaTeX associé au MSC pour les adapter à la description des attaques et des preuves des protocoles cryptographiques :

- Le temps n'est pas précis : comme pour un diagramme MSC, seul compte l'ordre d'exécution des messages et des actions. Les timers ne seront donc pas utilisés. Cette hypothèse pourra néanmoins être affaiblie si des protocoles requièrent une précision de ce côté là.
- Un message envoyé est consommé immédiatement : toutes les flèches, représentant les messages, seront donc horizontales et séparées d'un niveau. Il est possible que des messages s'effectuent au même niveau mais dans ce cas les canaux seront différents. (Il n'y a pas besoin d'une notion de niveau).
- La représentation est simple : à première vue, peu de notions avancées présentent un intérêt. Seules les actions, les conditions et, éventuellement, la décomposition seront utilisées. Les co-régions ne seront, par exemple, sûrement pas utiles.

Il faut aussi adapter le langage aux particularités des protocoles cryptographiques. On doit donc transposer certains noms pour que la représentation soit plus claire. Dans le formalisme qui va être défini, les MSC représenteront les attaques et les protocoles cryptographiques, les HMSC représenteront les preuves.

Il est nécessaire d'étoffer le formalisme pour permettre de décrire aux mieux des spécificités qui ne sont pas encore prises en compte. Par exemple, pour les MSC, le canal de communication n'est pas décrit. Il est juste représenté par une flèche qui va de l'émetteur vers le destinataire. De plus, rien ne permet de différencier les canaux : lorsqu'on regarde un diagramme MSC, il est facile de supposer que les informations transitant entre deux instances passent par le même canal ; ceci n'est pas forcément la réalité.

Dans le cadre de protocoles cryptographiques, il est utile de pouvoir décrire ce canal plus précisément et, en particulier, pouvoir en spécifier les caractéristiques. Lors de la transmission d'un message, le canal de communication peut être sécurisé ou non. Dans le cas d'un canal sécurisé, l'intrus peut écouter la transmission, mais en aucun cas intercepter et/ou modifier les messages. S'il n'est pas sécurisé l'intrus pourra écouter, intercepter et même modifier les communications sur le médium.

Un exemple de formalisme XML pour un diagramme MSC est donné sur la figure 4.

La description du fichier `prouve.dtd` est donnée figure 5. Ce fichier permet de préciser certaines caractéristiques communes qui permettront de valider la syntaxe des fichiers XML.

Une fois ce formalisme accepté, nous avons pu établir les fondements de l'outil de simulation d'attaque, que nous allons maintenant présenter.

2.1 Le simulateur

Ce simulateur est développé en Java, sous Eclipse, en collaboration avec le langage de réécriture TOM.

2.1.1 Principe de fonctionnement du simulateur

Son fonctionnement peut être décrit ainsi : à partir d'une spécification en langage PROUVÉ d'un protocole de cryptographie (celui fournit aux outils de preuves automatiques) et de la trace d'exécution résultant de l'analyse de l'outil de preuve automatique utilisé, le simulateur doit exécuter la trace selon le protocole afin de vérifier si l'attaque obtenue est effectivement réalisable (voir aussi figure 6).

Pratiquement, l'implémentation se fait selon l'idée suivante. Pour une trace donnée en entrée, le simulateur crée une base de données comprenant les instances en jeu ainsi que les connaissances dont elles disposent à chaque étape du protocole. Ainsi, à chaque étape, des vérifications peuvent être effectuées. Elles ont pour but de s'assurer, en s'appuyant sur les connaissances dont dispose chaque instance, que la trace est possible.

Le simulateur visite un à un chacun des nœuds du fichier XML.

Notons, à titre de détail technique, que le fichier XML est préalablement transformé pour supprimer tous les blancs de début et de fin de zone texte. On a constaté que les traces fournies n'étaient pas uniformes sur ce point. Par exemple, `pub_key_` et `_pub_key` sont ramenés tous deux à `pub_key` (`_` symbolise

```
<?xml version='1.0' encoding='ISO-8859-1' standalone="no"?>
<!|fichier prouvé.xml -->
<!|exemple de formalisme XML pour un diagramme MSC -->

<!DOCTYPE msc SYSTEM "msc.dtd">

<msc>
   <instance>
      <name>Instance A</name>
      <knowledge>private keys, public keys</knowledge>
   </instance>
   <instance>
      <name>Instance B</name>
      <knowledge>private keys, public keys</knowledge>
   </instance>
   <instance>
      <name/>
     <knowledge/>
   </instance>
   <message>
      <in>tag_in</in>
      <out>tag_out</out>
      <data>data to transfert</data>
   </message>
   <action>
      <name>decode</name>
      <result>key</result>
   </action>
</msc>
```

FIG. 4 – Description du fichier `prouve.xml`.

le blanc au début et à la fin de la portion de texte), ce qui uniformise le contenu du fichier XML et permet son traitement.

2.1.2 Étape 1 : Initialisation

A chaque visite d'un nœud instance,

```
<instance>
   <name/>
   <knowledge/>
</instance>
```

on crée l'instance dont le nom est contenu dans la balise `<name/>` et on initialise sa base de données avec, d'une part, le nom et la clef publique de

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Fichier Prouve.dtd-->
<!-- Ce fichier décrit le formalisme utilisé pour les fichier xml représentant
     des preuves ou des attaques -->

<!-- définition des éléments -->

<!ELEMENT msc (instance+, message+, action*, knowledge+)>
<!ATTLIST msc name  CDATA #REQUIRED>
<!ATTLIST msc type ( preuve | attaque | algo) "attaque">
<!ATTLIST msc outil  ( ISpi | Casrul | Hermes ) #REQUIRED >

<!--Une instance est composée d'un 'nom', d'un 'role' -->
<!ELEMENT instance (role)>
<!ATTLIST instance name CDATA #REQUIRED >
<!ATTLIST instance param_effectif  CDATA #IMPLIED >

<!ELEMENT role (#PCDATA)>

<!--Un message est composé d'un point d'entrée 'in' d'un point de sortie 'out'
    d'un support optionnel et d'une 'donnée'-->
<!ELEMENT message (donnee)>
<!ATTLIST message in CDATA #REQUIRED >

<!ATTLIST message out CDATA #REQUIRED >

<!ATTLIST message support ( normal | coded | armoured) "normal">
<!--Une action est composée d'un nom, de la méthode utilisée, du résultat
    et de l'instance à laquelle elle est rattachée-->
<!ELEMENT action (methode?, donnee)>

<!ATTLIST action name CDATA #REQUIRED >
<!ATTLIST action instance CDATA #REQUIRED >

<!--Une méthode est constituée d'au moins 1 calcul et de son nom-->
<!ELEMENT methode (calcul+)>

<!ATTLIST methode description CDATA #REQUIRED >

<!ELEMENT calcul (fonction, donnee+)>
<!ELEMENT fonction (#PCDATA)>

<!-- Une connaissance est constituée d'une liste de 'donnée' et
     d'un attribut 'instance' à laquelle il est rattaché -->
<!ELEMENT knowledge (donnee+)>

<!ATTLIST knowledge instance CDATA #REQUIRED >

<!--Les données sont composées d'un nom, d'un type et d'une valeur-->
<!ELEMENT donnee (#PCDATA)>

<!ATTLIST donnee id CDATA #IMPLIED >

<!ATTLIST donnee type ( message | Integer | pub_key | priv_key | Id |
                        agent | Bool | Algo  ) "message" >
```

FIG. 5 – Description du fichier `prouve.dtd`.

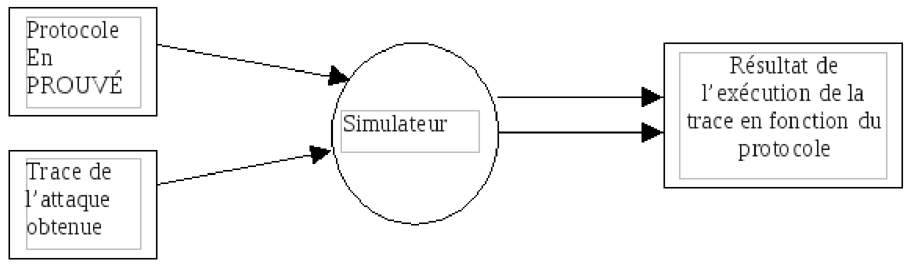


FIG. 6 – Fonctionnement du simulateur.

l'instance, ce qui est implicite dans le fichier XML, et de l'autre le contenu de la balise `<knowledge/>`.

2.1.3 Étape 2 : Visite successive des nœuds du protocole

Le protocole est une suite d'actions et de messages. Pour chacun d'entre eux, trois procédures sont effectuées : une procédure d'affichage du nœud dans lequel on se trouve, une procédure de vérification, et une procédure de mise à jour puis d'affichage de la base de connaissances de chaque instance.

L'affichage du nœud dans lequel on se trouve n'appelle pas plus de commentaires.

Plusieurs types de vérifications sont effectuées.

Vérification de type 1 : Qu'il s'agisse d'une action ou d'un message, on vérifie d'abord que les instances impliquées existent effectivement.

Vérification de type 2 : Au sein d'un nœud action, une méthode peut être définie. Elle comprend un ou plusieurs calculs. C'est la description de ce qui est nécessaire à la réalisation de l'action. Il s'agit donc, pour que celle-ci puisse être effectuée, de vérifier que les données nécessaires sont présentes dans la base de connaissances de l'instance concernée.

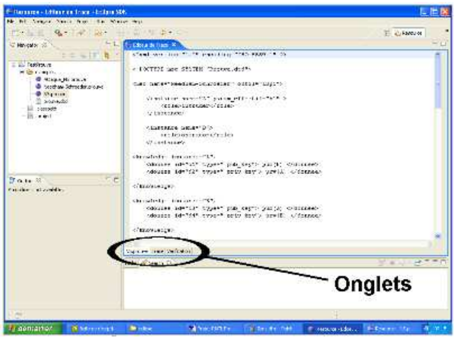
Vérification de type 3 : Pour qu'un message `{X,Y,...}key` puisse être envoyé par une instance, il faut que celle-ci dispose d'une part du contenu `X,Y,...` qu'elle souhaite envoyer, et de la clef `key` pour crypter le message.

Vérification de type 4 : Enfin, lorsqu'une instance reçoit un message, on vérifie qu'elle possède la clef qui lui permette de le déchiffrer.

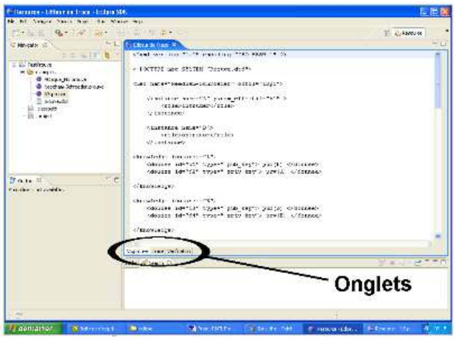
Le simulateur affiche les erreurs rencontrées. Par exemple, « Attention : l'action ne peut être effectuée : la clef de chiffrement manque ».

Une fois ces vérifications effectuées, la base de connaissances est mise à jour (avec le contenu d'un message reçu par exemple) et est affichée.

Note technique à propos des clefs : Une clef est identifiée par son attribut type, ainsi que l'instance qui la concerne, et non par son texte. Ainsi, `prv_key(B)` et `prive_key(B)` sont considérées comme identiques, du moment que l'attribut 'type' est le même (en l'occurrence `priv_key`).



(a) onglet 1 Affichage du fichier XML



(b) onglet 2 Trace

FIG. 7 – Interface

2.2 Prouve Plug-in - Manuel utilisateur

2.2.1 Installation

Prouve Plug-in est un plug-in Eclipse. Pour l'installer, il faut mettre manuellement le fichier `prouve_1.0.0.jar` dans le répertoire `/eclipse/plugins` où « /eclipse » est le répertoire dans lequel le logiciel Eclipse SDK est installé. Il suffit ensuite de démarrer Eclipse.

2.2.2 Vue d'ensemble

Prouve Plug-in apporte un nouveau multi-éditeur à l'environnement Eclipse. Un éditeur permet de lire le contenu d'un fichier. Un multi-éditeur comprend plusieurs onglets comportant chacun un éditeur. On accède à un éditeur en double-cliquant sur le fichier dont l'extension est « .prouve », ou en effectuant l'action « clique droit sur le fichier → ouvrir avec → Editeur Prouvé ».

Le multi-éditeur Prouve Plug-in comprend trois onglets. Le premier onglet est un simple éditeur de texte qui affiche le contenu du fichier XML (voir figure 7 gauche). Le deuxième onglet est un éditeur graphique affichant la trace (voir figure 7 droite).

Le troisième onglet affiche la vérification de la trace (voir figure 8).

2.2.3 Problèmes éventuels

Si la page « Vérification » est blanche, la raison la plus plausible est qu'on trouve la ligne suivante au début du fichier `.prouve`

```
<!DOCTYPE msc SYSTEM "prouve.dtd">
```

qui fait référence au fichier `prouve.dtd`. Si celui-ci n'est pas présent dans le même répertoire que le fichier `.prouve`, le parseur XML ne pourra pas fonctionner (voir figure 9).

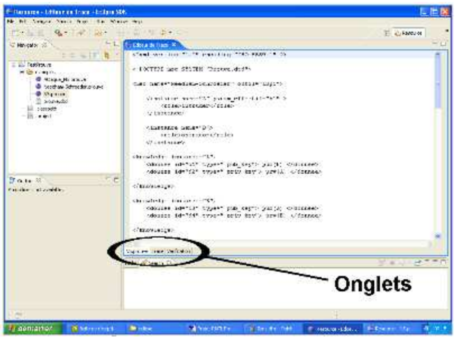


FIG. 8 – onglet 3 Vérification

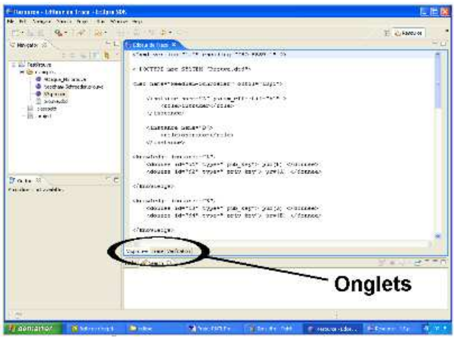


FIG. 9 – Le fichier `prouve.dtd` est nécessaire si le fichier `.prouve` lui fait référence

3 Tâche 3 : Études de cas

Afin de définir les besoins et les priorités à mettre sur les outils consacrés à la vérification de protocoles cryptographiques développés au sein du projet Prouvé, il a été proposé de travailler en situation réelle, sur des protocoles plutôt durs, en effectuant le cycle suivant : modélisation, formalisation puis validation dans un ou plusieurs outils. On se proposait de réaliser ce travail sur deux grandes études de cas : un protocole de porte-monnaie électronique [56] développé récemment par une équipe de France Télécom et un protocole d'enchères électronique.

Ces protocoles avaient été choisis en raison de leur caractéristiques spécifiques. En effet, contrairement aux protocoles que l'on peut trouver dans [31], ces deux études de cas présentent de nombreuses caractéristiques justifiant leur étude au sein du projet Prouvé. Ils sont intéressants aussi bien du point de vue de la modélisation des propriétés algébriques que de la modélisation des propriétés de sécurité. Ces deux points constituent deux objectifs majeurs du projet Prouvé. D'autre part, ces protocoles ont un besoin crucial d'être vérifiés : la moindre faille pouvant permettre la réalisation d'une fraude à grande échelle.

Le protocole de porte-monnaie électronique à clef publique reflète fidèlement les ambitions du projet et sort sans surprise du spectre de tous les outils de vérification existants lors du lancement du projet. La deuxième étude de cas est plus prospective. Il s'agit finalement d'un protocole de vote électronique [84], développé par Jacques Traoré (France Télécom). Elle a été retenue (à la place du protocole d'enchères électronique) en raison de l'importance grandissante et des avancées récentes faites sur ce sujet. En dehors des études de cas mentionnées ci-dessus, nous nous étions également engagés à maintenir une base d'exemples de protocoles cryptographiques.

3.1 Maintien d'une base de test

SPORE is a publicly accessible Web page `http://www.lsv.ens-cachan.fr/spore/`. Its purpose is to provide a public repository of cryptographic protocols, with for each protocol the description of its various versions, the security properties that it is claimed to satisfy, those that it genuinely satisfies and under which assumptions, and the known attacks against the protocol.

The page SPORE aims at being used as a source of case studies for the designers of formal methods and tools for automated cryptographic protocol verification, continuing the endeavor of John Clark and Jeremy Jacob whose survey on protocol verification, published in 1997, has been widely distributed. The whole repository is accessible on line, so as to cater for some interactivity with users and to promote its reusability by tool designers.

The messages of a protocol in SPORE can be specified in one of the two forms :

- following the common syntax (see below)
- in a free syntax, in particular when some particularities of the protocol do not fit in the proposed syntax.

In both cases, the specification of the protocol messages can be preceded by some type declarations.

We propose a *common syntax* for the specification of protocol messages which is close to the one of the BAN paper [21], which has been adopted in many later theoretical works about cryptographic protocols. For instance, most of the the authentication protocols in the famous Clark and Jacob survey [31] follow this syntax (except fot the notation of the encryption operator). However, as mentioned above, not all the specifications of the protocols repository shall necessarily follow this syntax. Of course, such a very general syntax cannot avoid imprecisions and ambiguities concerning the actions taken by the principals following the protocol. Some `actions` can be added between two protocol messages to describe some actions of the receiver of the first message. The syntax for these actions is free.

Since many semantics exist for cryptographic protocols given in the above syntax, we have chosen not to impose a specific one for this repository. The semantical aspects of a protocol can be discussed in the following sections :

- Explanations : This section contains some additional information about the protocol variables, the initial conditions, the treatment of received messages and the construction of new messages by principals, the control flow of the protocol execution, and in general anything that can explain that is not explicit in the protocol specification.
  Some equational axioms for functional identifiers can also be given here.
- Requirements : Some properties that the protocol is supposed to ensure are written here in natural language.

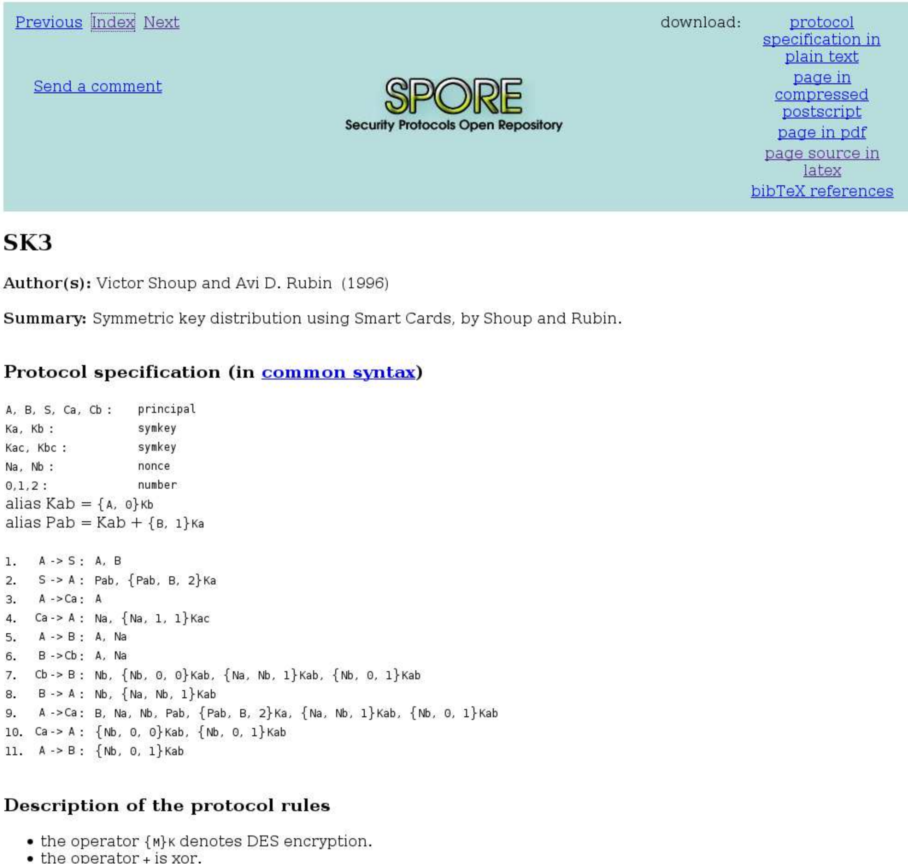
A screenshot of a protocol in the SPORE repository is given in Figure 10.

3.2 Protocole de porte-monnaie électronique

L'étude des protocoles cryptographiques a pris une importance considérable avec le développement du commerce électronique. Le commerce électronique recouvre les paiements par carte à l'aide de terminaux portables où l'utilisateur doit confirmer son identité en entrant son code secret à quatre chiffres, mais aussi la monnaie électronique qui a pour but d'émuler électroniquement la monnaie courante. Cette monnaie est, pour des montants peu élevés, plus intéressante qu'une transaction bancaire par carte qui a un coût élevé pour le commerçant. Toutes ces applications demandent des garanties de sécurité élevées, portant sur des propriétés de secret et d'authenticité, mais aussi de nombreuses autres propriétés, parmi lesquelles, la non-duplication (des factures, dans l'intérêt du client), la non-révocation (des commandes, dans l'intérêt du commerçant), ...

Dans ce contexte nous avons traité deux versions d'un protocole permettant la réalisation d'une transaction à l'aide d'un porte-monnaie électronique matérialisé par une carte à puce [56]. Bien que les deux protocoles aient pour but la même fonctionnalité, ils sont très différents en raison des primitives cryptographiques utilisées.

Le premier protocole étudié est un porte-monnaie électronique à chiffrement symétrique. Ce protocole permet la réalisation d'une transaction entre un



Previous Index Next

download: protocol specification in plain text
page in compressed postscript
page in pdf
page source in latex
bibTeX references

Send a comment

SPORE
Security Protocols Open Repository

SK3

Author(s): Victor Shoup and Avi D. Rubin (1996)

Summary: Symmetric key distribution using Smart Cards, by Shoup and Rubin.

Protocol specification (in common syntax)

```
A, B, S, Ca, Cb :   principal
Ka, Kb :            symkey
Kac, Kbc :          symkey
Na, Nb :            nonce
0,1,2 :             number
alias Kab = {A, 0}Kb
alias Pab = Kab + {B, 1}Ka

1.  A -> S :  A, B
2.  S -> A :  Pab, {Pab, B, 2}Ka
3.  A ->Ca :  A
4.  Ca -> A : Na, {Na, 1, 1}Kac
5.  A -> B :  A, Na
6.  B ->Cb :  A, Na
7.  Cb -> B : Nb, {Nb, 0, 0}Kab, {Na, Nb, 1}Kab, {Nb, 0, 1}Kab
8.  B -> A :  Nb, {Na, Nb, 1}Kab
9.  A ->Ca :  B, Na, Nb, Pab, {Pab, B, 2}Ka, {Na, Nb, 1}Kab, {Nb, 0, 1}Kab
10. Ca -> A : {Nb, 0, 0}Kab, {Nb, 0, 1}Kab
11. A -> B :  {Nb, 0, 1}Kab
```

Description of the protocol rules

- the operator {M}K denotes DES encryption.
- the operator + is xor.

FIG. 10 – A screenshot of a protocol description in SPORE.

porte-monnaie électronique et un serveur : le but est de garantir un bon niveau de sécurité avec de bonnes performances grâce à l'utilisation du chiffrement symétrique (moins coûteux que les mécanismes de chiffrement asymétrique).

Le second protocole étudié est un porte-monnaie électronique à chiffrement asymétrique développé par France Télécom. Ce protocole permet la réalisation d'une transaction entre un porte-monnaie électronique et un serveur : le but est de garantir un bon niveau de sécurité et de gagner en ouverture par rapport à l'approche symétrique. Ce protocole utilise une méthode de chiffrement asymétrique à faible coût fondée sur une exponentielle modulaire. Le point délicat est que le protocole fait intervenir certaines propriétés algébriques de l'opérateur d'exponentiation.

Le but de ce premier cas d'étude était de définir les évolutions à apporter aux outils du projet. C'est pourquoi il a été conduit sur ensemble représentatif d'outils : HERMÈS et CASRUL qui sont des outils du projet et PROVERIF [11].

3.2.1 Formalisation

Afin de mener l'étude de ces protocoles (version symétrique et asymétrique) nous avons procédé à la formalisation du protocole et des propriétés de sécurité [19].

Protocole. Pour les deux versions du protocole, c'est essentiellement la prise en compte du contexte d'exécution des protocoles qui a posé des problèmes dans la plupart des outils. La séquentialité des sessions, issue d'une contrainte physique (à un instant donné, une seule carte est insérée dans le lecteur) n'a pas pu être prise en compte par les différents outils. De ce point de vue, ProVerif est l'outil le mieux placé, il utilise une algèbre de processus pour décrire le contexte, ce formalisme est flexible, simple et naturel à appréhender. L'utilisation de variables qui perdurent au fil des sessions est un aspect qui n'est pas présent dans tous les outils, or le solde du porte-monnaie est une variable de ce type.

Pour la formalisation de la version asymétrique il a fallu traiter spécifiquement l'exponentielle modulaire car à l'époque aucun outil n'était capable d'appréhender directement cet opérateur. Pour cela, nous avons eu recours à un codage utilisant le chiffrement asymétrique. Ce codage reste naturelle et, d'un point de vue théorique l'exponentielle peut effectivement être remplacée par un chiffrement asymétrique sans que les caractéristiques du protocole ne soient modifiées. Par contre dans la pratique cette approche n'a pas été retenue par les concepteurs du protocole pour des raisons évidentes de coûts.

Propriétés. Les propriétés à formaliser étaient la *non création de fausse monnaie* et la *non création de faux litiges.*

La *non création de fausse monnaie* passe par un certain équilibre dans les balances des différents agents, mais l'on peut choisir d'énoncer et de vérifier une propriété plus forte en s'intéressant à l'énoncé suivant : Si le serveur termine une session apparemment avec un certain porte-monnaie P en créditant sa balance d'un montant M alors le porte-monnaie P a bien débité sa balance d'un montant M. Le point intéressant est que ce dernier énoncé correspond à une authentification valuée qui est une propriété cryptographique classique présente dans les outils tels que Casrul ou ProVerif.

La *non création de faux litiges* correspond à une propriété de non répudiation combinée à une propriété de non duplication. Elle doit assurer qu'en cas de litige le serveur pourra prouver de façon indiscutable sa bonne foi devant un juge en exhibant une information spécifique appelée évidence. En particulier quoi que fasse l'intrus le serveur ne doit jamais stocker à son insu une fausse évidence. Faute de primitives adaptées, des propriétés telles que la non répudiation et la non duplication n'ont pas été traitées directement. Pour les formaliser la démarche a été la suivante : dans un premier temps un nouveau rôle représentant le juge a été introduit avec un protocole modélisant le rendu de verdict. Ensuite la non création de faux litiges a été codée via une combinaison d'authentifications valuées.

3.2.2 Résultats et retombées

Le résultat de ce travail de formalisation sur cette première étude de cas a guidé la définition de la syntaxe du langage de spécification développé dans la tâche 1 du projet : *Sémantique des protocoles et des propriétés*. Ces deux protocoles peuvent donc être modélisés fidèlement dans le langage Prouvé. Les résultats obtenus, décrits dans [20], sont les suivants :

- Nous avons vérifié pour les deux protocoles la *non création de fausse monnaie* et la *non création de faux litiges* pour un nombre borné de sessions à l'aide des outils Casrul et ProVerif. Pour cela, nous avons eu recours à un codage permettant de s'abstraire des propriétés algébriques les plus complexes. Ce travail a donné lieu à une publication [47].
- A l'aide de l'outil Hermès nous avons mis en évidence un point intéressant. Si le contexte n'impose pas une exécution séquentielle du protocole, et que serveur et porte-monnaie partagent un même identifiant alors la propriété de *non création de faux litiges* n'est pas satisfaite pour la version symétrique du protocole. Ceci signifie par exemple qu'il serait dangereux de déployer sans précaution ce protocole sur un serveur WEB.

À l'issue de cette étude menée dans les outils existants, nous avons introduit dans le langage Prouvé des constructions permettant de modéliser facilement les notions suivantes : propriétés algébriques, canaux sûrs, scénarios d'exécution, conditionnelles et variables persistantes. Nous pouvons pu ainsi modéliser fidèlement les deux versions de ce protocole.

Des travaux ont également été réalisés dans le cadre du projet pour la prise en compte des propriétés algébriques de l'exponentielle modulaire lors de la phase de vérification. On peut noter la prise en compte de la multiplication sur les exposants pour un nombre borné de sessions dans [26] et [30]. Malheureusement ces résultats ne permettent pas encore de traiter la version asymétrique de notre protocole qui repose sur le segment additif de l'exposant. Une autre approche consiste à réduire la théorie équationnelle de notre problème à une théorie associative et commutative [35]. Ce résultat de réduction à l'avantage d'être particulièrement générique cependant la résolution du problème résultant reste encore un écueil. Cette première étude d'un mécanisme pour la prise en compte des propriétés algébriques a été introduit dans les outils du projet Hermès et Casrul. De plus, un mécanisme pour une prise en compte naturelle de la non répudiation a été introduit dans Casrul. Ce mécanisme est basé sur la notion d'ensemble d'évidences.

3.3 Protocole de vote électronique

Le but de cette seconde étude était de tester les limites du langage et des outils du projet Prouvé. Ainsi le protocole que nous avons choisi d'étudier est volontairement complexe tant au niveau de la modélisation des propriétés de sécurité que de la description du protocole lui-même.

Le protocole étudié est un protocole de vote qui a été mis au point chez France Télécom par J. Traoré [84]. Cette étude de cas, contrairement aux protocoles que l'on peut trouver dans [31] présente de nombreuses caractéristiques justifiant son étude au sein du projet PROUVÉ. En effet, un protocole de vote, pour être utilisable, doit vérifier de nombreuses propriétés de sécurité dont certaines semblent contradictoires. Pour assurer ces propriétés, il a fallu mettre en place de nouveaux mécanismes de base, plus complexes que les primitives cryptographiques classiques que sont le chiffrement (symétrique / asymétrique) et les fonctions à sens unique. Ainsi, un tel protocole est intéressant aussi bien du point de vue de la modélisation des propriétés algébriques que de la modélisation des propriétés de sécurité.

3.3.1 Formalisation

Notre étude de cas est basé sur le mécanisme de signature en aveugle et peut être considéré comme un dérivé du protocole de protocole de Fujioka, Okamoto, Ohta (FOO92) [55]. Ce dernier nécessite l'intervention de l'électeur à plusieurs reprises. Il n'est donc pas *vote and go*. En effet, l'électeur pour ne pas révéler son vote va simplement envoyer un engagement. Il doit donc lors d'une première phase obtenir la signature de cet engagement auprès d'une autorité et envoyer cet engagement. Ensuite, une fois cette première phase terminée, il doit faire parvenir une donnée permettant d'ouvrir son engagement.

Or, pour être utilisable en pratique, il est important que l'électeur n'ait pas à intervenir plusieurs fois au cours de la procédure de vote. Pour parer à ce défaut, Ohkubo *et al.* [76] ont modifié le protocole FOO92. Ils proposent de ne plus utiliser un schéma d'engagement qui aboutit nécessairement à un protocole de vote en deux phases, mais un schéma de chiffrement classique associé à un réseau de mélangeurs. Une implémentation de ce schéma a été réalisée, il s'agit de VOTOPIA [59]. J. Traoré a mis en évidence une faille (concernant la vérifiabilité) sur VOTOPIA et propose l'utilisation d'un schéma de signature en aveugle à anonymat révocable pour contourner le problème. C'est ce dernier protocole qui est notre cas d'étude.

Protocole. Fort de notre première étude de cas, le langage PROUVÉ est relativement complet en particulier en ce qui concerne la description des scénarios. Cette souplesse dans la description du scénario nous a permis de coder le mécanisme de phases du protocole en permettant une synchronisation globale entre les différents processus. Ce type de synchronisation est primordial pour la modélisation de protocoles de vote électronique. En effet, pour garantir l'anonymat des votes, il faut par exemple assurer que la procédure de publication ne commence pas avant que les votants aient fini leur procédure de vote.

La formalisation des propriétés algébriques des primitives cryptographiques n'a également pas posé de problème. La théorie équationnelle inclut les équations décrivant le comportement des clefs publiques, du chiffrement et des signatures. Moins classiquement elle inclut également le comportement de la signature en aveugle à anonymat révocable ainsi que le mécanisme de preuve à divulgation nulle de connaissance nécessaire à la vérification des mélangeurs.

Le principal problème rencontré lors de la modélisation du protocole est lié à la manipulation des listes. Ces dernières sont massivement utilisées dans le protocole pour garantir la vérifiabilité. Le langage PROUVÉ possède un type List mais très peu de fonctionnalités sont offertes. Si l'on souhaite utiliser d'autres fonctionnalités telles que la suppression d'un élément d'une liste ou le parcours d'une liste, il faut alors définir de nouveaux symboles de fonctions et ajouter des équations à la théorie équationnelle. Ceci est possible, mais devient très rapidement extrêmement lourd.

Un autre problème concerne l'absence de boucle while ... do .... Il s'agit d'un choix délibéré est mûrement réfléchi. Une telle instruction est en effet complexe à traiter lors de la vérification. Nous avons donc préférer ne pas l'inclure dans notre modèle.

Propriétés. La formalisation des propriétés d'un protocoles de vote est une tâche particulièrement délicate. Ces dernières sont nombreuses et parfois très subtiles avec en particulier :

Pas de Résultat Partiel - Éligibilité - Double Vote. Personne ne doit être capable d'obtenir des résultats partiels, la connaissance de ces résultats pourrait influencer les électeurs n'ayant pas encore voté. Seules les personnes autorisées à voter le peuvent, et aucun électeur ne doit pouvoir voter deux fois lors d'une même élection. Ces propriétés ont pu être formalisées dans notre langage. L'éligibilité, par exemple, est une propriété qui peut se voir comme une propriété d'atteignabilité. Elle peut se modéliser dans le langage PROUVÉ. Pour cela il suffit de donner à l'intrus un *vote challenge*, et de voir si ce vote peut se retrouver dans le résultat final.

Secret des Votes (Anonymat) - Sans Reçu. Personne ne doit être capable de faire le rapprochement entre un électeur et son vote et aucun électeur ne doit être capable de prouver la manière dont il a voté. Obtenir ou être capable de construire un reçu de son vote, c'est à dire un document prouvant la manière dont on a voté, permettrait l'achat de vote ou la coercition (forcer quelqu'un à voter d'une certaine manière et s'en assurer ensuite). Ces propriétés ont été formalisées et ont fait l'objet d'articles [61, 48]. Cependant ces propriétés ne peuvent pas être formalisées avec le langage PROUVÉ puisque celui-ci ne prend pas en compte la notion d'équivalence observationnelle, indispensable pour modéliser ces notions d'anonymat.

Vérifiabilité (Individuellement / Universellement). Chaque électeur peut vérifier que son vote a été comptabilisé. Toute personne doit pouvoir se convaincre que tous les votes valides ont été comptabilisés sans avoir été modifiés. Ces propriétés sont particulièrement difficile à définir et n'ont pas été étudiées dans le cadre du projet.

A l'origine du projet PROUVÉ le but était de traiter automatiquement des propriétés d'invariance, il s'agissait d'un choix délibéré issu de l'état de l'art et de la multitude de protocoles conçus dans le but de garantir de telles propriétés. Le contenu de cette section montre que pour les protocoles les plus modernes et les plus complexes un tel point de vue n'est plus aussi vrai, bien que les propriétés d'invariance restent présentes dans ces protocoles.

3.3.2 Résultats et retombées

Ce cas d'étude n'a pas pu être traité d'un seul tenant et dans un cadre uniforme. Ceci n'a rien d'étonnant et c'était prévu dès le départ du projet. Contrairement au premier cas d'étude (le porte-monnaie électronique) qui était un problème adapté aux outils du projet ici le but était très différent : évaluer les carences et les évolutions potentielles des techniques développées au sein du projet sur un protocole beaucoup plus complexe.

Globalement les résultats sont très positifs. Ils montrent que même sur un protocole complexe les outils de preuve automatique apportent une aide considérable sur certains aspects comme la preuve de secret du vote ou de l'éligibilité. D'un autre côté, certaines propriétés de sûreté qui sortent du spectre des outils existants ont pu être prouvées manuellement mais formellement. Ce dernier point est important car ce travail manuel a permis de mieux apprécier les méthodes qui devront être mises en oeuvre demain pour mécaniser les preuves qui ont été réalisées. Concrètement les retombées de ce cas d'étude sont les suivantes :

- Le secret faible du vote à l'issue de la première phase du protocole a été vérifié automatiquement pour un nombre non borné de sessions à l'aide de l'outil Hermès du projet.
- La vérification du secret faible ne permet pas d'exclure des attaques par dictionnaire ou une divulgation partielle du secret. Dans un article issu du projet [61] le secret fort et l'éligibilité sont vérifiés pour le protocole de Fujioka, Okamoto, Ohta [55]. Les preuves ont été réalisées automatiquement avec ProVerif [11] qui est capable de prendre en compte l'équivalence observationnelle dans certains cas. Cette article contient également une formalisation et une preuve manuelle mais formelle de l'anonymat. Une formalisation de la propriété de sans reçu a également été réalisée [48].
- Le réseau de mélangeurs est une pièce importante de notre protocole. C'est l'élément qui est à la base de l'anonymat. Les mélangeurs de notre cas d'étude sont des mélangeurs optimistes et à notre connaissance ils n'ont jamais été traité formellement. Un article issu du projet [66] contient un travail préliminaire sur ce thème. Il s'agit de l'étude du schéma de vote de Chaum qui est en fait une version simplifiée de notre réseau de mélangeurs. Ce document est intéressant à plusieurs titres : d'un côté il prouve dans le modèle symbolique l'anonymat de vote pour ce schéma, mais en plus il donne des conditions sur les primitives cryptographiques pour que ce résultat reste valide dans le modèle calculatoire.

3.4 Conclusion

Ces études de cas, en particulier celle du protocole de porte-monnaie, nous ont permis de mettre au point les fonctionnalités du langage PROUVÉ. Ces études ont été réalisées conjointement par les différents partenaires. Plus précisément, un premier travail consistant à comprendre la spécification des protocoles à été réalisé par Francis Klay (France Télécom) et Stéphanie Delaune (en thèse CIFRE chez France Télécom et au LSV) en interaction avec les concepteurs des protocoles (M. Girault et J. Traoré). Plusieurs études sur les protocoles de porte-monnaie ont été menées dans les outils HERMÈS, (Verimag) et CASRUL, (Loria). Ces dernières ont été réalisées en collaboration avec le partenaire concepteur de l'outil (Verimag pour HERMÈS, Loria pour CASRUL).

En ce qui concerne le protocole de vote électronique, les études sur le protocole de J. Traoré ont été réalisées par France Télécom et le LSV. Au delà de cette étude de cas, la thématique du vote électronique a suscité un intérêt chez les différents partenaires. Un projet spécifique, le projet AVOTÉ, dédié à la vérification de protocole de vote électronique, a démarré en janvier 2008 entre les partenaires France Télécom, LSV, Verimag et LORIA.

4 Tâche 4 : Réalisation et mise au point des outils

L'objectif de cette tâche est de développer une boîte à outils qui offre différentes méthodes automatiques de vérification et de mise au point des protocoles cryptographiques.

Dans la boîte à outils nous devions intégrer deux outils de vérification automatique des protocoles cryptographiques : CASRUL [7] développé par l'INRIA Lorraine et HERMÈS [18, 17] développé par Verimag. Ces outils prennent en entrée des représentations des protocoles cryptographiques dans un format commun, le langage PROUVÉ(décrit dans la section Sémantique des protocoles et des proprétés) et envisagent une présentation commune des résultats qui soit accessible à un utilisateur non averti. Comme ces outils utilisent des approches théoriques différentes pour la vérification des protocoles cryptographiques, ils sont complémentaires tant au niveau des protocoles qu'on peut vérifier, qu'au niveau des propriétés et des scénarios qu'ils peuvent analyser. Ceci permet à un utilisateur de disposer d'une famille d'outils qui couvrent la plupart des protocoles cryptographiques et des propriétés de sécurités, et permet de faire la vérification dans le contexte le plus approprié (nombre fini/infini de sessions, itérations, etc.). De plus, l'utilisateur peut comparer les résultats de vérifications obtenus par des techniques différentes sur la même description des protocoles et des propriétés.

Dans les sous-sections suivantes nous allons présenter le principe de chaque outil, les difficultés rencontrées et les décisions prises pour la connexion avec le langage PROUVÉ, ainsi que les prototypes finales obtenus.

4.1 Hermes

4.1.1 Principe de l'outil

L'outil HERMÈS est développé au Laboratoire Verimag (disponible à l'adresse http ://www-verimag.imag.fr/∼async/hermes/prouve/hermes.php). Il permet la vérification de protocoles cryptographiques pour un nombre borné ou non borné des sessions.

HERMÈS calcule des conditions suffisantes pour garantir que les messages échangés au cours d'exécution d'une ou plusieurs sessions du protocole ne peuvent être exploités pour déduire des secrets.

Le contexte d'exécution et le nombre de sessions sont décrites par le scénario. Un scénario PROUVÉ est traduit en Hermes dans une expression régulière sur les règles de réécritures, expression construite en utilisant les opérateurs de choix non-déterministe, de séquence, de parallélisme, d'itération.

L'outil est basé sur la notion de messages capables de garder un secret, appelés *messages protecteurs*. Il s'agit des messages ou tous les secrets sont a l'intérieur des sous-messages de la forme $\{X\}_K$, c'est-à-dire cryptés au moyen d'une clef dont l'inverse n'est pas connu de l'intrus. Les messages protecteurs se déduisent à partir des clefs déclarées secrètes dans les hypothèses du protocole.

Étant donné un protocole, un ensemble de secrets et un ensemble d'hypothèses sur les clefs secrètes, HERMÈS calcule, d'une part, un ensemble de messages protecteurs pour les secrets échangés au cours du protocole ; et d'autre part, il ajoute aux secrets les messages qui permettent de construire une attaque. Il s'agit en particulier des messages dans lesquels les secrets ne sont pas protégés.

Les deux ensembles définissent les conditions dans lesquelles le protocole peut être utilisée sans risque : la propriété de secret est garantie si, dans les messages initialement connus de l'intrus, les secrets calculés sont protégés par les messages protecteurs calculés. Ces ensembles peuvent être calculés de manière exacte pour les scénarios finies, et approximé pour les scénarios infinies.

Le cas borné correspond a des scénarios finies (nombre de sessions fini). L'expression régulière générée ne comporte pas d'itérations, ce qui permet une exploration complète et exacte de l'arbre d'exécution des transitions. L'outil termine dans tous les cas et la réponse est exacte : si le protocole est correct pour le scénario analyse, l'outil le confirme, sinon, l'outil fournit une trace d'attaque.

On considère comme exemple le protocole Needham-Schroeder (voir section A.1) dans un scénario ou le participant honnête Alice a une session ouverte avec un autre participant honnête Bob plus une sessions ouverte avec l'intrus. La traduction en règles de réécriture est présentée dans la section A.2. La propriété qu'on cherche à vérifier exige que les valeurs fraîches Na et Nb générées dans les sessions exécutés entre des participants honnêtes restent secrétes.

```
always
forall Q:Bob ; P:Alice .
( Q.v_A != intruder & Q.B != intruder & P.A != intruder & P.B != intruder )
  -> ( issecret(Q.Nb) & issecret(P.Na) )
```

L'outil HERMÈS génère les ensembles suivantes (un ensemble non-vide de DEDUCIBLE_SECRETS signifie la présence d'une attaque), et fournit aussi des traces d'attaque (pour un exemple de trace d'attaque voir la section A.3).

```
CLAIM SECRETS :
Nb_1;
 Na_3;
 Nb_2;
 inv(PK(bob));
 inv(PK(alice))

SECRETS = {(Na_4,alice)}_PK(bob);
Nb_1;
Na_3;
Nb_2;
inv(PK(bob));
inv(PK(alice));
BADS    = {(Na_4,Sec(v_Nb_4))}_PK(alice)

DEDUCIBLE_SECRETS = {(Na_4,alice)}_PK(bob)
```

Le cas non-borné correspond à un scénario avec un nombre non-borné de sessions en parallèle, ou a des itérations infinies. À l'issue du calcul dont la terminaison est assurée par un opérateur de convergence forcée (widening), HERMÈS retourne un ensemble de secrets qui contient au moins les secrets de départ et le complément de l'ensemble de messages protecteurs. En principe, il peut y avoir des cas ou l'outil ne termine pas, mais sur tous les exemples testés, on n'a jamais rencontré cette situation. L'analyse est correcte : si l'outil termine et réponds OK alors le scénario satisfait la propriété testé, dans le cas contraire une trace d'attaque est proposé, mais cette trace peut correspondre a une fausse attaque.

On considère comme exemple le protocole Needham-Schroeder-Lowe (voir section A.4) dans un scénario ou les participants honnêtes Alice et Bob executent un nombre non-borné de sessions en parallèle. La traduction en règles de réécriture est présenté dans la section A.5. La propriété qu'on cherche à vérifier exige que les valeurs fraîches Na et Nb générées dans les sessions exécutés entre des participants honnêtes restent secrétes.

```
always
forall Q:Bob . ( Q.B != intruder & Q.v_A != intruder ) -> issecret(Q.Nb)
```

L'outil HERMÈS génère les ensembles suivantes.

```
CLAIM SECRETS :
Nb_3;
 inv(PK(H))

SECRETS = Nb_3;
inv(PK(H));
BADS    = {(Na_8,((Na_7_1,(Sup((Sec(_389),I)),H)),H))}_PK(H);
  {(Na_8,((Sec(_342),I),H))}_PK(H);
  {(Na_8,((Na_7_1,(Sup((Na_6_1,(Sec(_76),I))),H)),H))}_PK(H);
  {(Na_8,((Na_6_1,(Sec(_12),I)),H))}_PK(H);
  {(Na_6_1,(Sec(v_Nb_6_1),I))}_PK(H);
  {(Sec(v_Na_2_1),I)}_PK(H);
  {(Na_7_1,(Sup((Na_6_1,(Sec(_19),I))),H))}_PK(H);
  {(Na_7_1,(Sup((Sec(_349),I)),H))}_PK(H)

DEDUCIBLE_SECRETS =
```

L'ensemble DEDUCIBLE_SECRETS étant vide, célà signifie que le protocole Needham-Schroeder-Lowe satisfait la propriété testé pour un nombre non-borné de sessions.

4.1.2 Exemples

Pour la mise au point du langage PROUVÉ et des outils consacrés à la vérification de protocoles cryptographiques, on s'est proposé d'utiliser comme études de cas deux exemples assez complexes : un protocole de porte-monnaie électronique [56] développé récemment par une équipe de France Télécom et un protocole de vote électronique.

Le premier protocole étudié est un porte-monnaie électronique à chiffrement symétrique. Ce protocole permet la réalisation d'une transaction entre un porte-monnaie électronique et un serveur. A l'aide de l'outil HERMÈS nous avons mis en évidence une faiblesse. Si le contexte permet des exécutions en parallèle du protocole, et que le serveur et le porte-monnaie partagent un même identifiant, alors la propriété de *non création de faux litiges* n'est pas satisfaite. Ceci signifie par exemple qu'il serait dangereux de déployer sans précaution ce protocole sur un serveur WEB.

Un autre protocole étudié est un protocole de vote qui a été mis au point chez France Télécom par J. Traoré [84]. L'outil HERMÈS a permis de prouver de manière automatique la propriété de secret faible du vote à l'issue de la première phase du protocole pour un nombre non borné de sessions.

L'outil a aussi permis de vérifier plusieurs protocoles décrits dans la base SPORE (page web `http://www.lsv.ens-cachan.fr/spore/`), dans différents contextes d'exécution (nombre de sessions borné ou non-borné).

4.2 Casrul

Casrul est un sous-système du logiciel AVISPA [2], comprenant le traducteur du langage haut niveau HLPSL vers le langage bas-niveau IF d'AVISPA et les outils de vérification développés dans l'équipe CASSIS (TA4SP et AtSe). Dans le contexte du projet PROUVÉ, Casrul s'est enrichi d'un nouveau traducteur, permettant de fournir aux outils de vérification des protocoles décrits dans le langage de spécification de PROUVÉ, ce dernier permettant de considérer des protocoles plus complexes que ceux exprimables dans le langage d'AVISPA ainsi que des scénarios plus évolués.

4.2.1 Outils de vérification

Casrul est donc composé de deux outils de vérification : AtSe et TA4SP.

AtSe : Cet outil est l'implantation directe de l'approche basée sur une logique contrainte décrite dans [25, 86]. Cet analyseur de protocoles utilise le modèle de Dolev-Yao pour l'intrus, et peut chercher des attaques à différents niveaux : en mode typé, ou en mode non typé, exploitant par exemple l'associativité de la concaténation de messages.
Lorsqu'une attaque est trouvée, elle est décrite très clairement par l'échange des messages entre les agents et l'intrus.

TA4SP : L'originalité de cet outil est de vérifier des propriétés de secret pour un nombre non borné de sessions d'un protocole. La méthode utilisée [13] consiste à résoudre un problème d'accessibilité de termes. Le protocole et certaines capacités de l'intrus sont spécifiés par un ensemble de règles de réécriture.

[2] `http://www.avispa-project.org`

Les connaissances initiales et d'autres capacités de l'intrus, tout comme les instances de sessions, sont représentées par le langage d'un automate d'arbre. L'approche consiste alors à calculer l'ensemble des termes atteignables à partir du langage initial, en utilisant les règles de réécriture. L'ensemble ainsi obtenu correspond aux termes connus par l'intrus.

En général, le problème d'accessibilité est indécidable. L'approche classique consiste à calculer une sur-approximation de l'ensemble des termes accessibles : si un terme n'appartient pas à la sur-approximation, alors il n'est pas accessible par réécriture. Ainsi, une propriété de secret est satisfaite si tous les termes secrets ne sont pas dans la sur-approximation. À l'opposé, une propriété est falsifiée si au moins un terme secret appartient à une sous-approximation des connaissances de l'intrus.

L'analyse avec TA4SP se fait entièrement automatiquement, à partir d'une spécification PROUVÉ.

4.2.2 Exemples

Les principales expérimentations faites dans le cadre du projet PROUVÉ ont concerné le protocole de porte-monnaie électronique proposé par France Telecom R&D. Nous avons ainsi utilisè l'analyseur AtSe pour deux variantes de ce protocole : une approche symétrique et une approche assymétrique.

Le porte-monnaie électronique – approche symétrique. Pour cette étude nous avons procédé en deux temps. Nous avons d'abord formalisé classiquement la spécification standard du porte-monnaie à clé symétrique puis nous avons essayé d'évoluer vers une spécification la plus complète et réaliste possible pour déterminer les limites de l'outil. Dans ce dernier travail les ajouts portent essentiellement sur la nature des informations échangées entre les rôles, plus fidèle à la réalité et sur la modélisation d'un processus de non répudiation électronique virtuel car il semble qh'en pratique ce processus ne soit pas réalisé en ligne.

La spécification a été testée avec l'outil CASRUL et aucune attaque n'a été trouvée dans le mode par défaut de vérification typée.
L'introduction de sessions testant le bon fonctionnement de la spécification, comme par exemple en mettant pour porte-monnaie un agent inconnu de la machine, ou pour machine un agent inconnu du tiers de confiance, provoque les attaques attendues. De même, si l'intrus joue le rôle du porte-monnaie, il peut sans difficulté tromper l'authentification de la machine par le tiers de confiance sur les informations stockées.

Avec la vérification de type désactivée nous aurions dû retrouver la pseudo attaque mise en évidence dans l'outil Hermes. Ce travail n'a pas pu être réalisé á cause des restriction de Casrul dans la prise en compte des règles conditionnelles. Nous avons également effectué la vérification en activant la propriété d'associativité de la concaténation de messages. Dans ces conditions une attaque est systématiquement trouvée, ce qui montre qu'une mauvaise implantation du protocole pourrait nuire à son bon fonctionnement.

Protocole	Diagnostic
NSPKL	SECURE
NSPK	INCONCLUSIVE
Needham Schroeder Symmetric Key	SECURE
Denning-Sacco shared key	SECURE
Yahalom	SECURE
TMN	INCONCLUSIVE
Andrew Secure RPC	SECURE
Wide Mouthed Frog	SECURE
Kaochow v1	SECURE

FIG. 11 – Résultats de vérification obtenus avec TA4SP.

Le porte-monnaie électronique – approche asymétrique. L'étude de cette variante du protocole a volontairement été effectuée sur une spécification standard, sans entrer dans les détails comme nous l'avons fait pour l'approche symétrique.

La spécification a été testée dans l'outil et aucune attaque n'a été trouvée avec le mode par défaut qui implique une vérification typée. Ces tests ont aussi considéré plusieurs sessions avec des montants de transaction identiques.
Si l'intrus est un acteur officiel du protocole, que ce soit dans le rôle du porte-monnaie ou celui de la machine, il pourra facilement fausser les propriétés d'authentification sur le montant de la transaction.

Lorsque la vérification de type est désactivée, aucune attaque n'est trouvée lors de sessions entre agents honnêtes. Par contre, si la propriété d'associativité de la concaténation de messages est activée, une attaque est systématiquement trouvée, ce qui montre qu'une mauvaise implantation du protocole peut nuire à son bon fonctionnement.

Autres expérimentations. D'autres expérimentations ont été effectuées avec l'analyseur TA4SP, pour des propriétés de secret de protocoles de la librairie SPORE[3], spécifiés dans la langage PROUVÉ. Quelques résultats sont listés dans la Figure 11.

En ce qui concerne les protocoles `NSPK` et `TMN`, le diagnostic `INCONCLUSIVE` s'explique par le fait que ces protocoles sont tous deux connus pour être défaillants.

Les protocoles analysés n'utilisent pas toute l'expressivité du langage PROUVÉ. Cependant, leur spécification est beaucoup plus précise qu'avec la notation Alice-Bob classique. Elle garantit l'absence d'ambiguité dans l'interprétation de la spécification.

4.3 Conclusion

Dans le cadre du projet PROUVÉ, nous avons développé et amélioré plusieurs outils de vérifications de protocoles cryptographiques. Ces outils ont été

[3]Security Protocols Open Repository, `http://www.lsv.ens-cachan.fr/spore/`

connectés au même langage d'entré, le langage PROUVÉ. Comme ces outils utilisent des approches théoriques différents pour la vérification des protocoles cryptographiques, ils sont complémentaires tant au niveau des protocoles qu'on peut vérifier, qu'au niveau des propriétés et des scénarios qu'il peuvent analyser.

Ceci permet à un utilisateur de disposer d'une famille d'outils qui couvrent la plus part des protocoles cryptographiques et des propriétés de sécurités, et permet de faire la vérification dans le contexte le plus approprié (nombre fini/infini de sessions, itérations, etc.). De plus, l'utilisateur peut comparer les résultats de vérifications obtenus par des techniques différentes sur la même description des protocole et des propriétés.

5 Tâche 5 : Affaiblissement de chiffrement parfait

Les primitives cryptographiques sont le plus souvent représentées par des symboles libres de fonctions, suivant ainsi l'*hypothèse du chiffrement parfait.* Cependant certaines attaques ou meme le déroulement honnete de certains protocoles utilisent les propriétés algébriques d'opérateurs comme le Â« ou exclusif Â», l'exponentiation modulaire ou l'addition par exemple.

Cette section se découpe en deux parties. Dans un premier temps, nous présentons un état de l'art général des résultats obtenus pour la vérification automatique des protocoles cryptographiques, pour certaines propriétés algébriques. Dans un deuxième temps, nous détaillons les résultats obtenus dans le cadre projet RNTL PROUVÉ, par les différents partenaires du projet.

Cryptographic protocols are short programs designed to ensure secure communications on channels that may be controlled by an attacker. Considering the increasing size of networks and their dependence on cryptographic protocols, a high level of assurance is needed in the correctness of such protocols. These protocols are notoriously difficult to design and test, and serious flaws have been found in many protocols. Consequently, there has been a growing interest in applying formal methods for validating cryptographic protocols. These protocols use cryptographic primitives as public and symmetric encryption. These functions are based on mathematical notions (like modular exponentiation or elliptic curves) and on algorithmically hard problems (as extracting the modular logarithm or factorization into prime numbers). A first approach is to verify a protocol with its actual cryptographic primitives and to show that attacking the protocol can be reduced to solving an algorithmically hard problem. Such proofs are done by hand and are often long and difficult. In particular, they seem very hard to automate.

An other approach is to abstract from cryptographic primitives. This may be justified by the observation that many attacks rely only on the logical structure of the protocols and simply consist in replaying some messages at the right steps. That is why formal methods usually consider encryption schemes as black boxes and assume that an adversary cannot learn anything from an encrypted message except if he has the key. This is called the *perfect encryption assumption.* More generally, formal methods assume *perfect cryptography* : the other cryptographic primitives (like pairing or hashing) are also idealized in order to enable automatic verification. Even if these assumptions are not realistic, many real attacks have been discovered using this approach. The most famous flaw is the man-in-the-middle attack on the Needham-Schroeder protocol with public key encryption, found by G. Lowe [67] using an automatic tool.

Many decidability results have been obtained under this perfect cryptography hypothesis : the secrecy preservation is co-NP-complete for a bounded number of sessions [81], and decidable for an unbounded number of sessions under some additional restrictions [69, 53, 33]. Many tools have also been developed to automatically verify cryptographic protocols like [74, 68, 11].

Recent works investigate how to relax the perfect cryptography assumption by refining the abstraction on cryptographic primitives. The aim is to take into account some of the algebraic properties of the cryptographic primitives. Most of the algebraic properties studied so far and presented in this survey are properties that can be modeled using equations. For example a commutative encryption is expressed by the equality $\{\{x\}_y\}_z = \{\{x\}_z\}_y$. Such a representation of the algebraic properties is natural for many cryptographic primitives and very convenient since it enables to reuse classical methods on first order logic for terms modulo equational theories. The interest of studying the algebraic properties of the cryptographic primitives is that some attacks may be missed when abstracting encryption as a perfect black box. For example, Bull's protocol has been proven secure [79] under the perfect encryption assumption. However ; the protocol uses the exclusive or operator, which is associative, commutative, nilpotent, and has a neutral element. An attack on this protocol relying on these properties has been given in [82]. Even without searching for attacks, one may need algebraic properties to simply be able to specify some protocols. For example, the simple Three Pass protocol proposed by R. Shamir [31] requires a commutative encryption like the RSA encryption. All these examples will be developed in the following sections.

The aim of the work-package 5 is to present sufficient conditions to verify cryptographic protocols with algebraic properties. This survey contains two main sections. In Section 5.1, we first present a brief overview of decidability results or incomplete decision procedures that existed at the beginning of the project for some algebraic properties using formal methods. Producing a report of the state-of-the-art of decision procedures for algebraic properties was the first sub-task of the WP5. This overview has been published [38, 37]. Then in Section 5.2, we provide a detailed description of the results obtained by the partners of the RNTL project PROUVÉ.

5.1 Overview of existing results

Many decidability results have been obtained under the perfect cryptography assumption. Our first task in the project was to identify relevant algebraic properties and overview the existing work. We have published a survey in JCS [38]. The overview of the results under the perfect cryptography assumption are summarized in Table 1, the overview of the results for algebraic properties are summarized in Table 2. Most algebraic properties will be defined more precisely in the next section.

<table>
<tr><th rowspan="2">Bounded number of sessions</th><th colspan="2">Unbounded number of sessions</th></tr>
<tr><th>Without nonces</th><th>With nonces</th></tr>
<tr><td rowspan="4">co-NP-complete
[81]</td><td>Bounded message length :
DEXPTIME-complete
[53, 29]</td><td>Bounded message length :
Undecidable
[53, 6]</td></tr>
<tr><td>Tagged protocols :
EXPTIME
[12]</td><td>Strongly typed protocols :
Decidable [69]
Tagged protocols :
Decidable [80]</td></tr>
<tr><td>One copy :
3-EXPTIME
[33]</td><td>Ping-pong protocols :
PTIME
[52]</td></tr>
<tr><td colspan="2">General case :
Undecidable [54, 32]</td></tr>
</table>

TAB. 1 – Summary of Results for the Security Problem under the Perfect Cryptography Assumption.

5.2 Results obtained within the RNTL Project PROUVÉ - passive case

5.2.1 Intruder Deduction for *AC*-like Equational Theories with Homomorphisms/Distributivity

The well-known perfect cryptography assumption is unrealistic for cryptographic primitives with visible algebraic properties. The classical Dolev-Yao model can be extended to deal with the fact that the intruder can exploit these properties. We have shown that the problem is actually in PTIME for the case of an exclusive or (resp. Abelian Groups) operator in combination with the homomorphism axiom [45]. The problem is addressed by solving a system of linear equations over Z/2Z[h] (resp. Z[h]). This work improves the EXPTIME complexity results we obtained in this case at the beginning of the project [63]. The works described above deal with the verification problem in presence of a passive attacker.

We also investigate the case of several variants of AC-like axioms such as exclusive or, Abelian Groups with an encryption operator which distributes over the AC operator. We prove decidability of the intruder deduction problem in both cases. We obtain a PTIME decision procedure in a restricted case, the so-called binary case. Our decision procedures are based on a careful analysis of the proof system modeling the deductive power of the intruder, taking into account the algebraic properties of the equational theories under consideration. The analysis of the deduction rules interacting with the equational theory relies

	Intruder deduction problem	Security problem	
		Bounded number of sessions	Unbounded number of sessions
Commutativity	*PTIME [28]*	*co-NP-complete [28]*	**Ping-pong protocols :** *co-NP-complete [85]*
Exclusive or	*PTIME [27]*	**General case :** *Decidable [34]* **Restricted protocols :** *co-NP-complete [27]*	**One copy :** *3-EXPTIME [33]* **Two-way automata :** *Decidable [87]*
Abelian Groups	*NP [34]*	*Decidable [83]*	**Two-way automata :** *Decidable [87]*
Homomorphism	*PTIME [36]*	**Fixed blocks** *co-NP-complete [24]*	
Prefix	*PTIME [27]*	**Fixed blocks** *co-NP-complete [27]*	
Abelian Groups and Modular Exponentiation	*PTIME [26]*	**General case :** *Decidable [83]* **Restricted protocols :** *co-NP-complete [26]*	*Decision Procedure [57]*
Time-stamps	*PTIME* (1)	*Decidable [15]*	*Decision Procedure [51]*

(1) For the intruder deduction problem, there is no notion of time. Thus deciding whether a message with timestamps can be deduced from a set of messages with timestamps correspond to the intruder deduction problem where timestamps are considered as constants known to the intruder.

Empty boxes mean that, to our knowledge, no result has been obtained so far.

TAB. 2 – Summary of Decidability Results or Decision Procedures for some Equational Theories.

on the manipulation of $\mathbb{Z}$-modules in the general case, and on results from prefix rewriting in the binary case [65]. We have also investigated this problem in presence of the equational theory of a commutative encryption operator which distributes over the exclusive or operator. We have shown that the intruder deduction problem is decidable and we provided a 2-EXPTIME decision procedure. This work has been published at SecReT [62].

We also investigate the problem of solving deducibility constraint for a complex equational theory involving several AC operators. This theory allows us to model the asymetric version of our case study (electronic payment protocol designed recently by France Télécom). We provide a decision procedure for ground deducibility constraints. This result has been published in Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of his 60th Birthday [23].

5.2.2 Deciding Knowledge and Static Equivalence under Equational Theories

Many formal definitions explain the knowledge of an attacker in terms of message deduction, denoted by $\vdash$. Given a set of messages S and another message M, one asks whether M can be computed from S. The messages are represented by expressions, and correspondingly the computations allowed are symbolic manipulations of those expressions. Intuitively these computations can rely on any step that an eavesdropper who has obtained the messages in S can perform on its own in order to derive M. For example, the eavesdropper can encrypt and decrypt using known keys, and it can extract parts of messages.

Despite its usefulness in proofs about protocol behaviors, the concept of message deduction does not always provide a sufficient account of knowledge, and it is worthwhile to consider alternatives. For instance, suppose that we are interested in a protocol that transmits an encrypted boolean value, possibly a different one in each run. We might like to express that this boolean value remains secret by saying that no attacker can learn it by eavesdropping on the protocol. On the other hand, it is unreasonable to say that an attacker cannot deduce the well-known boolean values "true" and "false". Instead, we may say that the attacker cannot distinguish an instance of the protocol with the value "true" from one with the value "false". More generally, we may say that two systems are equivalent when an attacker cannot distinguish them, and we may then express security guarantees as equivalences. The use of equivalences is common in computational approaches to cryptography, and it also figures prominently in several formal methods (e.g., [5, 4]).

Two systems that output messages that an attacker can tell apart are obviously distinguishable. Conversely, in order to establish equivalences between systems, an important subtask is to establish equivalences between the messages that the systems generate (for example, between the encrypted boolean values). These equivalences may be called *static equivalences*, denoted by $\approx$, because they consider only the messages, not the dynamic processes that generate them. Bisimulation proof techniques can reduce process equivalences to static equivalences plus fairly standard bisimulation conditions [4].

We have established several decidability results for static equivalence, for various equational theories. We have shown that $\vdash$ and $\approx_s$ may be undecidable in general ; that the decidability of $\vdash$ reduces to the one of $\approx_s$ but that the converse does not hold : there exists an equational theory such that $\vdash$ is decidable and $\approx_s$ is not.

To obtain decidability results, we have first considered *convergent subterm theories*, defined by a finite set of equations of the form $M = N$ where N is a proper subterm of M or a constant symbol. Such theories capture a wide array of functions such as pairing, projections, various flavors of encryption and decryption, digital signatures, one-way hash functions, etc. We have shown [1] that both $\vdash$ and $\approx$ can be decided in polynomial time for convergent subterm theories. We have then extended these results [2] for considering theories with associative-commutative symbols, requiring only loose, abstract conditions on the equational theory for messages. As an application, we get decidability for many other useful theories, for example with blind digital signatures, homomorphic encryption, XOR, and other associative-commutative functions. A extended version of these two papers as been published in TCS'06 [3].

We have then considered the axioms of Associativity-Commutativity (AC), Unit element (U), Nilpotency (N), Idempotency (I), homomorphism (h), and more especially the combinations of these axioms that constitute monoidal theories. We have proposed [40, 41] a general approach to handle *monoidal* theories that covers several cases already studied, and furthermore includes some new decidability and complexity results on homomorphic operators. Our setting relies on the correspondence between a monoidal theory E and a semiring Se_E which allows us to give an algebraic characterization of the deducibility and indistinguishability problems. As a consequence we recover easily existing decidability results and obtain several new ones.

We have shown [8] that decidability results can be easily combined for any disjoint equational theories : if the deducibility and static equivalence are decidable for two disjoint theories, they are also decidable for their union. As an application, new decidability results can be obtained by combining previously obtained decision results.

5.2.3 Opacity and Guessing Attacks

We study an information flow property called opacity [73]. This notion, close to non-interference, means that an intruder is not able to deduce whether a certain property ψ is verified. The property ψ is defined on initial variables of the protocol. For example, ψ can have the form $v = 1$ where v is a vote that is casted during the protocol execution. The intruder observes a finite number of sessions in a passive way. For that purpose, we introduce a similarity relation over messages : $E \vdash m_1 \sim m_2$ if an intruder is not able to differentiate message m_1 from message m_2 by using knowledge E. Opacity is equivalent to satisfiability of constraints involving equalities and similarities. We have proven decidability of satisfiability for such constraints and so decidability of opacity when considering a passive intruder. Details can be found in [71].

The previous definition of opacity makes some restrictive hypothesis : it is only possible to consider a passive intruder. Hence a first extension for this work would be to add the general case of active intruders (as defined in Dolev-Yao model). In that case, we have proven a simplified version of opacity to be decidable [72] but we still do not have a proof for opacity in general. The main difficulty is that we have to model that the intruder forges its messages in the same way across the two sessions. This can be done using simultaneous deductibility : $E_1, E_2 \vdash m_1, m_2$ which means that m_1 can be deduced from E_1 and m_2 from E_2 by performing the same operations.

5.3 Results obtained within the RNTL Project PROUVÉ - bounded number of sessions

5.3.1 A Decision Procedure for Security Protocols with Explicit Destructors

We propose a decision procedure solving the problem of insecurity for cryptographic protocols containing explicit destructor symbols (like decryption and projection) and equality tests, in the presence of a bounded number of sessions and of a passive or an active intruder. The destructor operators are axiomatized by an arbitrary convergent rewrite system satisfying some syntactic restrictions. This approach, with parameterized semantics, allows us to weaken the security hypotheses for verification, *i.e.* to address a larger class of attacks than for models based on free algebra.

Our decision procedure is defined by an inference system for symbolic constraint solving based on basic narrowing techniques. It is polynomial in time in the case of a passive intruder and non-deterministic polynomial for an active intruder. Details may be found in [44].

5.3.2 A Decision Procedure for Security Protocols with Homomorphism

The first result, we obtained is an undecidability result in the case of the theory of Abelian Groups with the homomorphism axiom [46]. We have also considered the theory of exclusive or in combination with the homomorphism axiom and we have shown that the problem is decidable. One step of our proof relies on several results about unification problems in this particular theory which has been published at UNIF'06 [64]. However, the main step of our proof consists in reducing the constraint system for deducibility into a constraint system for deducibility in one step and using one particular rule of the constraint system. This constraint system, in turn, can be expressed as a system of quadratic equations of a particular form over the ring of polynomials in one indeterminate over the finite field $\mathbb{Z}/2\mathbb{Z}[h]$. We show that satisfiability of these systems of equations is decidable. This work has been published at ICALP'06 [49]. A general approach, based on this previous work, has also been proposed to deal with the class of monoidal equational theories [50].

5.3.3 Timestamps

Some cryptographic protocols rely upon timestamps that recipients use to verify timeliness of received messages or to identify replays of messages. In some applications where resources are critical, timestamps are used as an approximative implementation of nonces. However, one should be aware that timestamps are ordered and guessable which is not the case of nonces at least in the Dolev-Yao model. More generally, timing constraints can also be used to implement time outs that can be used, e.g. as protection against deny of service attacks. Most of the existing verification methods and decidability results for cryptographic protocols consider time-independent protocols.

In [16], we present a model for time-dependent cryptographic protocols and a corresponding decidability result. Although the model we present only deals with bounded protocols, that is a fixed number of sessions are considered, our model clearly identifies the main ingredients to be included in a general model. It is well-known that the verification problem of unbounded cryptographic protocols is undecidable in the untimed case, and hence, it is so for the timed case. To model timed cryptographic protocols, we include in our model clocks, time variables, and timestamps. Clocks are variables that range over the time domain and advance with the same rate as time. Each agent has its own set of clocks that it can reset. Clocks can be

We provide a complete and sound symbolic verification algorithm for timed cryptographic protocols that builds upon the results of [14]. We consider a rich class of reachability properties that allow to specify confidentiality and authentication. In fact, we introduce a logic that allows to describe secrecy, equalities between terms and control points. Then, given a bounded protocol Π and two formulae in this logic Φ and Ψ, the reachability problem we consider is whether there is a run of Π that starts in a configuration that satisfies Φ and reaches a configuration that satisfies Ψ.

5.3.4 Verification techniques parameterized by equational theories

We have provided general conditions on the equational theory and the intruder deduction system under which one obtains a decision procedure for the verification of security protocols for a bounded number of sessions.

This goal has been achieved in the case of the equational theories which do not involve an associative-commutative (AC) operator (e.g. variant of the Dolev-Yao intruder, blind signature theory used in our e-voting case study, ...). This result is fully described in [9] and has been published at ASIAN 2006 [10]. The statement of our main theorem is satisfactory because the conditions on the equational theory (the equational theory has to satisfy the finite variant property, a property we introduced in [35]) and the conditions on the inference system after the computation of the variants are straightforward to check. Then, they come with a proof normalisation result saying that normnal proofs only involve terms that can be computed from the hypotheses, the conclusion and the protocol rules. Hence, this gives us a proof search strategy and also a decidability result in the case of bounded number of sessions.

Now, in the case of equational theories involving an AC operator, the problem is actually quite challenging and has only been partly solved. The first part has been achieved in [35]. Indeed, the framework of the finite variant property can also been applied in presence of an AC operator. However, the second part seems to be difficult and it appears that the problem of solving even pure AC deducibility constraints is difficult. This has been partially solved in [22]. In this paper, we provide a decision procedure to solve AC deducibility constraints under some assumptions.

5.4 Results obtained within the RNTL Project PROUVÉ - unbounded number of sessions

Analyzing protocols for a bounded number of sessions is a powerful methods for detecting attacks. There is however no guarantee when protocols are used a higher number of times. A classical approach for analyzing protocols for an unbounded number of sessions is to used Horn clauses to represent the rules of the protocol and the abilities of the intruder. We are then reduce to the satisfiability problem of set of clauses for fragment of clauses. In this setting, we have obtained decidability results for various equational theories, relevant for security protocols.

5.4.1 CBC encryption and blind signatures

The *prefix property* is related to Cipher Bloc Chaining mode (CBC) which is important since it is a common encryption mode. Manipulating the encryption blocks, the intruder can get the encrypted prefix of any encrypted message. This yields to attacks that could not be mounted in the case of stronger encryption schemes. The same prefix property holds when weaker encryption schemes are used, like the Electronic Code Book (ECB) encryption mode.

Also, adding new cryptographic primitives for building protocols require to enhance the abstract model and it is unclear whether the known decidability results about security properties are preserved in these extensions. For instance the *blind signature* scheme allows an agent (e.g. a voter) to have a message (e.g. a vote) signed blindly by an another entity (e.g. an administrator). Then in some subsequent protocol steps the message can get unblinded. The blind signature scheme properties offer to the environment new attacks opportunities.

We have proposed [39] a resolution strategy for deciding a fragment of first-order logic that allows one to incorporate the prefix property of CBC encryption in our protocol modeling and to decide the existence of attacks exploiting this property. The same fragment applies to abstract properties of blind signature schemes, which are the main alternative to homomorphic encryption in E-voting [61]. The approach follows the line of [33] but requires a refined strategy in order to eliminate the clauses generated by resolution with the prefix properties. The verification algorithm has been applied to Needham-Schroeder Symmetric

Key protocol, which is subject to an attack when implemented using CBC encryption. We show how to fix the protocol and we show the correction of the resulting protocol.

5.4.2 Security of XOR-based Key Management Schemes

We have developed methods for analysing security protocols that use XOR, such as key-management APIs. As a case study, we consider the IBM 4758 CCA API, which is widely used in the ATM (cash machine) network. Earlier versions of the CCA API were shown to have serious flaws, and the fixes introduced by IBM in version 2.41 had not previously been formally analyzed.

We have first defined [43] a new class of protocols, containing in particular all the versions of the CCA API. We then show that secrecy after an unbounded number of sessions is decidable for this class. Implementing the decision procedure requires some improvements, since the procedure is exponential. We describe a change of representation that leads to an implementation able to verify a configuration of the API in a few seconds. As a consequence, we obtain the first security proof of the fixed IBM 4758 CCA API with unbounded sessions.

Key conjuring is the process by which an attacker obtains an unknown, encrypted key by repeatedly calling a cryptographic API function with random values in place of keys. We have proposed [42] a formalism for detecting computationally feasible key conjuring operations, incorporated into a Dolev-Yao style model of the security API. We show that security in the presence of key conjuring operations is decidable for a particular class of APIs, which includes the key management API of IBM's Common Cryptographic Architecture (CCA).

5.5 Tool development

Some important extensions of Doley Yao intruder with algebraic properties have been implemented in CL-Atse protocol analyser [86]. This tool finds attacks or proves security of protocols for a bounded number of sessions. It takes as input a protocol specified as a set of rewriting rules and uses rewriting and constraint solving techniques to model all reachable states of the participants to decide if an attack exists w.r.t. the Dolev-Yao intruder or some extensions. Any state-based security property (like secrecy, authentication, fairness, etc...) can be modelled in the system, and the algebraic properties of operators like xor or exponentiation are taken into account thanks to a complete modular unification algorithm. In particular, CL-Atse integrates an optimised version of the well-known Baader & Schulz unification algorithm, with modules for xor, exponentiation, and associative pairing. To our knowledge, CL-Atse is the only protocol analysis tool that includes complete unification algorithms for xor and exponentiation, with no limitation on terms or intruder operations. More precisely the system implements :

- The xor operator as an associative ($a \oplus (b \oplus c) = (a \oplus b) \oplus c$) and commutative ($a \oplus b = b \oplus a$) operator equipped with a unit element ($a \oplus 0 = a$) and nilpotency ($a \oplus a = 0$);

- The exponentiation : Denoted $Exp(g, a)$, it represents ga in some fixed group of prime order. Also, the product $\times$ on exponents models the multiplication in the corresponding (abelian) multiplicative group. Properties include inverse ($a \times a^{-1} = 1$), commutativity ($a \times b = b \times a$), normalisation ($Exp(Exp(g, M), N) = Exp(g, M \times N)$), ...
- The associative pairing : it represents the basic bit string concatenation. Associativity models the chance that an agent will not parse correctly the concatenation of some messages.

Protocols with exponentiations include e.g. the IKEv2 protocol with MAC authentication, the Kerberos protocol (cross-real or forwardable ticket versions), or the SPEKE or SSH protocols. These were succesfully analysed in reasonable time with CL-Atse (safe result).

A Exemples

A.1 Le protocole Needham-Schroeder

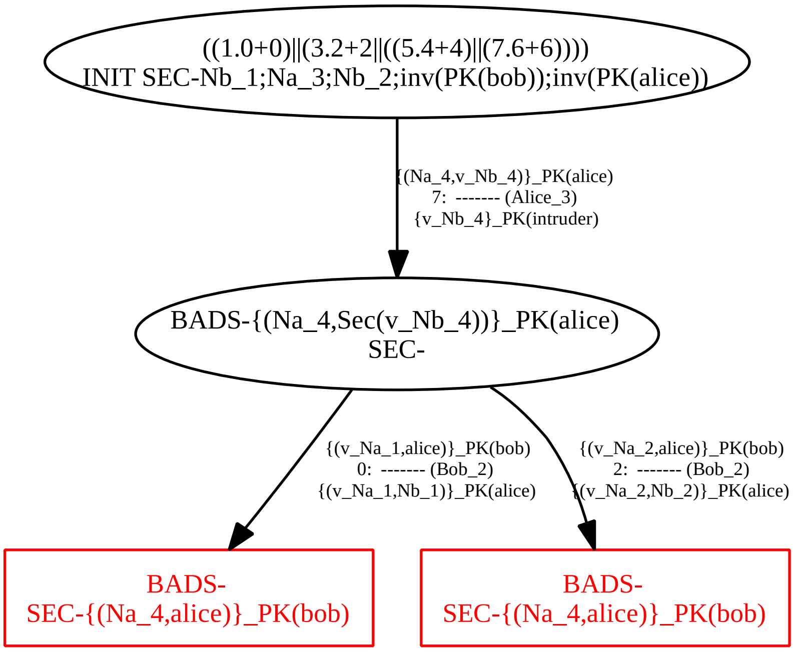
```
# The Needham-Schroeder Protocole
# A,B :    Principal
# Na,Nb :    Nonce
# PKa,PKb,PKs,SKa,SKb,SKs :    Key
# PKa,SKa :    is a key pair
# PKb,SKb :    is a key pair
# PKs,SKs :    is a key pair
# 1.    A  ->  B  :    {Na, A}PKb
# 2.    B  ->  A  :    {Na, Nb}PKa
# 3.    A  ->  B  :    {Nb}PKb

signature
alice, bob, intruder : principal;
PK : principal -> pubkey;
end

role Alice (A: principal; B: principal; SKa: privkey)
declare
Na, v_Nb: message;
begin
new(Na);
send(crypt(asym,PK(B),[Na,A]));
recv(crypt(asym,PK(A),[Na,v_Nb]));
send(crypt(asym,PK(B),v_Nb)) ;
end
role Bob (B: principal; SKb: privkey)
declare
Nb, v_Na : message;
v_A: principal;
begin
recv(crypt(asym,PK(B),[v_Na,v_A]));
new(Nb);
send(crypt(asym,PK(v_A),[v_Na,Nb]));
recv(crypt(asym,PK(B),Nb));
end

scenario
parallel
Bob (bob,inv(PK(bob)))
| Bob (bob,inv(PK(bob)))
| Alice(alice,bob,inv(PK(alice)))
| Alice(alice,intruder,inv(PK(alice)))
end
end
public
  alice, bob, intruder, PK
```

```
initial
  x_M = [[ inv(PK(intruder)) ]]
```

A.2 Les règles générées pour le protocole Needham-Schroeder

```
ROLE : Alice
parameters: A | B | SKa
local_vars: Na | v_Nb
fresh nonces and keys : Na

ROLE : Bob
parameters: B | SKb
local_vars: Nb | v_Na | v_A
fresh nonces and keys : Nb

RULES :

     {(v_Na_1,alice)}_PK(bob)
0  ---------------------------- (Bob_2)
     {(v_Na_1,Nb_1)}_PK(alice)

     ({Nb_1}_PK(bob),{(v_Na_1,alice)}_PK(bob))
1  ---------------------------- (Bob_3)
     null

     {(v_Na_2,alice)}_PK(bob)
2  ---------------------------- (Bob_2)
     {(v_Na_2,Nb_2)}_PK(alice)

     ({Nb_2}_PK(bob),{(v_Na_2,alice)}_PK(bob))
3  ---------------------------- (Bob_3)
     null

     null
4  ---------------------------- (Alice_1)
     {(Na_3,alice)}_PK(bob)

     {(Na_3,v_Nb_3)}_PK(alice)
5  ---------------------------- (Alice_3)
     {v_Nb_3}_PK(bob)

     null
6  ---------------------------- (Alice_1)
     {(Na_4,alice)}_PK(intruder)

     {(Na_4,v_Nb_4)}_PK(alice)
7  ---------------------------- (Alice_3)
     {v_Nb_4}_PK(intruder)

SCENARIO :
((1.0+0)||(3.2+2||((5.4+4)||(7.6+6))))
```

A.3 Un exemple de trace d'attaque fourni par Hermès



A.4 Le protocole Needham-Schroeder-Lowe

```
# The Needham-Schroeder Protocole with Lowe patch
# A,B :    Principal
# Na,Nb :    Nonce
# PKa,PKb,PKs,SKa,SKb,SKs :    Key
# PKa,SKa :    is a key pair
# PKb,SKb :    is a key pair
# PKs,SKs :    is a key pair
# 1.    A  ->  B  :    {Na, A}PKb
# 2.    B  ->  A  :    {Na, Nb,B}PKa
# 3.    A  ->  B  :    {Nb}PKb

signature
alice, bob, intruder : principal;
PK : principal -> pubkey;
end

role Alice (A: principal; B: principal; SKa: privkey)
declare
Na, v_Nb: message;
begin
new(Na);
send(crypt(asym,PK(B),[Na,A]));
recv(crypt(asym,PK(A),[Na,v_Nb,B]));
send(crypt(asym,PK(B),v_Nb)) ;
end

role Bob (B: principal; SKb: privkey)
declare
Nb, v_Na : message;
v_A: principal;
begin
recv(crypt(asym,PK(B),[v_Na, v_A]));
new(Nb);
send(crypt(asym,PK(v_A),[v_Na,Nb,B]));
recv(crypt(asym,PK(B),Nb));
end

scenario
parallel
forall i : int .exists p : principal .Bob ( p, inv(PK(p)) )   |
forall i : int .exists p : principal; q : principal .Alice( q, p, inv(PK(q)) )
end
end

public
  alice, bob, intruder, PK
initial
  x_M = [[ inv(PK(intruder)) ]]
```

A.5 Les règles générées pour le protocole Needham-Schroeder-Lowe

```
ROLE : Alice
parameters: A | B | SKa
local_vars: Na | v_Nb
fresh nonces and keys : Na

ROLE : Bob
parameters: B | SKb
local_vars: Nb | v_Na | v_A
fresh nonces and keys : Nb

      null
18  ---------------------------- (Alice_1)
      {(Na_8,H)}_PK(H)

      {(Na_8,(v_Nb_8,H))}_PK(H)
19  ---------------------------- (Alice_3)
      {v_Nb_8}_PK(H)

      null
16  ---------------------------- (Alice_1)
      {(Na_7_1,H)}_PK(H)

      {(Na_7_1,(v_Nb_7_1,H))}_PK(H)
17  ---------------------------- (Alice_3)
      {v_Nb_7_1}_PK(H)

      null
14  ---------------------------- (Alice_1)
      {(Na_6_1,H)}_PK(I)

      {(Na_6_1,(v_Nb_6_1,I))}_PK(H)
15  ---------------------------- (Alice_3)
      {v_Nb_6_1}_PK(I)

      null
12  ---------------------------- (Alice_1)
      {(Na_5_1,I)}_PK(H)

      {(Na_5_1,(v_Nb_5_1,H))}_PK(I)
13  ---------------------------- (Alice_3)
      {v_Nb_5_1}_PK(H)

      null
10  ---------------------------- (Alice_1)
      {(Na_4_1,I)}_PK(I)

      {(Na_4_1,(v_Nb_4_1,I))}_PK(I)
```

```
11  ---------------------------- (Alice_3)
      {v_Nb_4_1}_PK(I)

      {(v_Na_3,H)}_PK(H)
8  ---------------------------- (Bob_2)
      {(v_Na_3,(Nb_3,H))}_PK(H)

      ({Nb_3}_PK(H),{(v_Na_3,H)}_PK(H))
9  ---------------------------- (Bob_3)
      null

      {(v_Na_2_2,H)}_PK(H)
6  ---------------------------- (Bob_2)
      {(v_Na_2_2,(Nb_2_2,H))}_PK(H)

      ({Nb_2_2}_PK(H),{(v_Na_2_2,H)}_PK(H))
7  ---------------------------- (Bob_3)
      null

      {(v_Na_2_1,I)}_PK(H)
4  ---------------------------- (Bob_2)
      {(v_Na_2_1,(Nb_2_1,H))}_PK(I)

      ({Nb_2_1}_PK(H),{(v_Na_2_1,I)}_PK(H))
5  ---------------------------- (Bob_3)
      null

      {(v_Na_1_2,H)}_PK(I)
2  ---------------------------- (Bob_2)
      {(v_Na_1_2,(Nb_1_2,I))}_PK(H)

      ({Nb_1_2}_PK(I),{(v_Na_1_2,H)}_PK(I))
3  ---------------------------- (Bob_3)
      null

      {(v_Na_1_1,I)}_PK(I)
0  ---------------------------- (Bob_2)
      {(v_Na_1_1,(Nb_1_1,I))}_PK(I)

      ({Nb_1_1}_PK(I),{(v_Na_1_1,I)}_PK(I))
1  ---------------------------- (Bob_3)
      null

SCENARIO :
((9.8+8)||(19.18+18||(0+1+2+3+4+5+6+7+10+11+12+13+14+15+16+17)*))
```

B Thèses soutenues

Les thèses suivantes soutenues par des membres du projet couvrent complètement ou partiellement la thématique du projet PROUVÉ.

Mathieu BAUDET a soutenu le 16 janvier 2007 à l'ENS de Cachan sa thèse de doctorat intitulée *Sécurité des protocoles cryptographiques : aspects logiques et calculatoires*. Il est actuellement ingénieur à la Direction centrale de la sécurité des systèmes d'information.

Vincent BERNAT a soutenu le 1er juin 2006 à l'ENS de Cachan sa thèse de doctorat intitulée *Théories de l'intrus pour la vérification des protocoles cryptographiques*. Il travaille actuellement dans le privé.

Yohan BOICHUT a soutenu sa thèse de doctorat intitulée *Approximations pour la vérification automatique de protocoles de sécurité*, à l'Université de Franche-Comté, Besançon (France), septembre 2006. Il est actuellement maître de conférence à Orléans.

Stéphanie DELAUNE a soutenu le 20 juin 2006 à l'ENS de Cachan sa thèse de doctorat intitulée *Vérification des protocoles cryptographiques et propriétés algébriques*. Elle est actuellement chargée de recherche CNRS au LSV, ENS de Cachan.

Heinrich HOERDEGEN a soutenu en novembre 2007 sa thèse de doctorat intitulée *Vérification des protocoles cryptographiques : Comparaison des modèles symboliques avec une application des résultats - Étude des protocoles récursifs* à l' Université Henri Poincaré, Nancy. Il travaille actuellement dans le privé.

Romain JANVIER a soutenu en septembre 2006 sa thèse de doctorat intitulée *Lien entre modèles symboliques et computationnels pour les protocoles cryptographiques utilisant des hachages* à l' Université Joseph Fourier, Grenoble. Il est actuellement chercheur post-doctorant à l' Université Joseph Fourier, Grenoble.

Pascal LAFOURCADE a soutenu le 25 septembre 2006 à l'ENS de Cachan sa thèse de doctorat intitulée *Vérification de protocoles cryptographiques en présence de théories équationnelles*. Il est actuellement maître de conférence à l'université Joseph Fourier, Grenoble.

Laurent MAZARÉ a soutenu en octobre 2006 sa thèse de doctorat intitulée *Computational Soundness of Symbolic Models for Cryptographic Protocols* à l' Université Joseph Fourier, Grenoble. Il travaille actuellement dans le privé.

Judson Santos SANTIAGO a soutenu en novembre 2006 sa thèse de doctorat intitulée *Spécification et analyse de protocoles complexes dans AVISPA* à l'Université Nancy 2, novembre 2006. Il est actuellement maître de conférence au Brésil (Natal).

Ralf TREINEN a soutenu le 17 novembre 2005 sa thèse de HDR à l'ENS de Cachan intitulée *Résolution symbolique de contraintes*. Il est actuellement professeur à l'université Paris-Diderot.

Eugéne ZALINESCU a soutenu en décembre 2007 sa thèse de doctorat intitulée *Sécurité des protocoles cryptographiques : décidabilité et résultats de transfert*, à l' Université Henri Poincaré, Nancy. Il est actuellement chercheur post-doctorant à Microsoft Research, Saclay.

Yu ZHANG a soutenu le 21 octobre 2005 à l'ENS de Cachan sa thèse de doctorat intitulée *Cryptographic Logical Relations - What is the Contextual Equivalence for Cryptographic Protocols and How to Prove it ?*. Il est actuellement enseignant-chercheur à l'université des Nations Unies à Macao.

Références

[1] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. In *Proc. 31st Int. Coll. Automata, Languages, and Programming (ICALP'2004)*, Turku, Finland, April 2004.

[2] M. Abadi and V. Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 62–76, Aix-en-Provence, France, June 2005. IEEE Comp. Soc. Press.

[3] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2) :2–32, November 2006.

[4] 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, January 2001.

[5] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols : The spi calculus. In *Proc. of the 4th ACM Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997.

[6] R. Amadio and W. Charatonik. On name generation and set-based analysis in the dolev-yao model. In *Proc. of the 13th International Conference on Concurrency Theory (CONCUR'02)*, LNCS, pages 499–514. Springer Verlag, 2002.

[7] 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 K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

[8] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. Combining algorithms for deciding knowledge in security protocols. In Franck Wolter, editor, *Proceedings of the 6th International Symposium on Frontiers of Combining Systems (FroCoS'07)*, volume 4720 of *Lecture Notes in Artificial Intelligence*, pages 103–117, Liverpool, UK, September 2007. Springer.

[9] Vincent Bernat. *Théories de l'intrus pour la vérification des protocoles cryptographiques.* Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2006.

[10] Vincent Bernat and Hubert Comon-Lundh. Normal proofs in intruder theories. In Mitsu Okada and Ichiro Satoh, editors, *Revised Selected Papers of the 11th Asian Computing Science Conference (ASIAN'06)*, volume 4435 of *Lecture Notes in Computer Science*, pages 151–166, Tokyo, Japan, January 2008. Springer.

[11] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW)*, pages 82–96, Cape Breton (Nova Scotia, Canada), 2001. IEEE Comp. Soc. Press.

[12] Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols : Tagging enforces termination. In Andrew Gordon, editor, *Foundations of Software Science and Computation Structures (FoSSaCS'03)*, volume 2620 of *LNCS*, April 2003.

[13] Y. Boichut, N. Kosmatov, and L. Vigneron. Validation of Prouve protocols using the automatic tool TA4SP. In *Proceedings of 3rd Taiwanese-French Conference on Information Technology (TFIT)*, pages 467–480, Nancy, France, March 2006.

[14] L. Bozga, C. Ene, and Y. Lakhnech. On the existence of an effective and complete proof system for bounded security protocols. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume 2987 of *Lecture Notes in Computer Science*, 2004.

[15] L. Bozga, C. Ene, and Y. Lakhnech. A symbolic decision procedure for cryptographic protocols with time stamps. In *Proc. of the 15th International Conference on Concurrency Theory (CONCUR' 04)*, London (England), 2004.

[16] L. Bozga, C. Ene, and Y. Lakhnech. A symbolic decision procedure for cryptographic protocols with time stamps. *Journal of Logic and Algebraic Programming*, To appear, 2005.

[17] L. Bozga, Y. Lakhnech, and M. Périn. Abstract interpretation for secrecy using patterns. In H. Garavel and J. Hatcliff, editors, *Proceedings of TACAS'03 - 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Warsaw, Poland*, volume 2619 of *LNCS*. Springer, April 2003.

[18] L. Bozga, Y. Lakhnech, and M. Périn. Hermes : An automatic tool for verification of secrecy in security protocols. In Jr. W. A. Hunt and F. Somenzi, editors, *Proceedings of CAV'03 - 15th International Conference on Computer Aided Verification, Boulder, Colorado*, volume 2725 of *LNCS*. Springer-Verlag, July 2003.

[19] Liana Bozga, Stéphanie Delaune, Francis Klay, and Ralf Treinen. Spécification du protocole de porte-monnaie électronique. Technical Report 1, projet RNTL PROUVÉ, June 2004. 12 pages.

[20] Liana Bozga, Stéphanie Delaune, Francis Klay, and Laurent Vigneron. Retour d'expérience sur la validation du porte-monnaie électronique. Technical Report 5, projet RNTL PROUVÉ, March 2005. 29 pages.

[21] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, February 1989.

[22] Sergiu Bursuc, Hubert Comon-Lundh, and Stéphanie Delaune. Associative-commutative deducibility constraints. In Wolfgang Thomas and Pascal Weil, editors, *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *Lecture Notes in Computer Science*, pages 634–645, Aachen, Germany, February 2007. Springer.

[23] Sergiu Bursuc, Hubert Comon-Lundh, and Stéphanie Delaune. Deducibility constraints, equational theory and electronic money. In Hubert Comon-Lundh, Claude Kirchner, and Hélène Kirchner, editors, *Rewriting, Computation and Proof — Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of his 60th Birthday*, volume 4600 of *Lecture Notes in Computer Science*, pages 196–212, Cachan, France, June 2007. Springer.

[24] Y. Chevalier. *Résolution de probèmes d'accessibilité pour la compilation et la validation de protocoles cryptographiques.* PhD thesis, Université Henri Poincaré, Nancy, France, 2003.

[25] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In Ed Brinksma and Kim Guldstrand Larsen, editors, *14th International Conference on Computer Aided Verification, CAV'2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 324–337, Copenhagen (Denmark), July 2002. Springer.

[26] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In *FSTTCS*, pages 124–135, 2003.

[27] Yannick Chevalier, Ralf Küsters, Michael Rusinowitch, and Mathieu Turuani. An np decision procedure for protocol insecurity with xor. In *Proc. of 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, 2003.

[28] Yannick Chevalier, Ralf Küsters, Michael Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with commuting public key encryption. In *Proc. of Automated Reasoning for Security Protocol Analysis (ARSPA '04)*, pages 53–63, Cork, Ireland, 2004.

[29] Yannick Chevalier, Ralf Küsters, Michael Rusinowitch, Mathieu Turuani, and Laurent Vigneron. Extending the dolev-yao intruder for analyzing an unbounded number of sessions. In *Proc. of the 17th International Workshop in Computer Science Logic (CSL'03)*, 2003.

[30] Yannick Chevalier and Michaël Rusinowitch. Hierarchical combination of intruder theories. In *RTA*, pages 108–122, 2006.

[31] J. Clark and J. Jacob. A survey of authentication protocol literature. 1997.

[32] H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. In *Research Report LSV-01-13*, december 2001.

[33] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proc. of*

the 14th Int. Conf. on Rewriting Techniques and Applications (RTA'2003), volume 2706 of *LNCS*, pages 148–164. Springer-Verlag, June 2003.

[34] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proc. of 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 271–280, Ottawa (Canada), 2003. IEEE Comp. Soc. Press.

[35] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property : How to get rid of some algebraic properties. In Jürgen Giesl, editor, *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307, Nara, Japan, April 2005. Springer.

[36] Hubert Comon-Lundh and Ralf Treinen. Easy intruder deductions. In Nachum Dershowitz, editor, *Verification : Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 225–242. Springer, February 2003. Invited paper.

[37] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. Technical Report LSV-04-15, Laboratoire Spécification et Vérification, ENS Cachan, France, September 2004. 36 pages.

[38] V. Cortier, S. Delaune, and P. Lafourcade. A Survey of Algebraic Properties Used in Cryptographic Protocols. *Journal of Computer Security*, 14(1/2006), 2006.

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

[40] Véronique Cortier and Stéphanie Delaune. Deciding knowledge in security protocols for monoidal equational theories. In *Proc. of the 14th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'07)*, volume 4790 of *Lecture Notes in Artificial Intelligence*, pages 196–210, Yerevan, Armenia, October 2007. Springer.

[41] Véronique Cortier and Stéphanie Delaune. Deciding knowledge in security protocols for monoidal equational theories. In Pierpaolo Degano, Ralf Küsters, Luca Viganò, and Steve Zdancewic, editors, *Proceedings of the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'07)*, pages 63–80, Wrocław, Poland, July 2007.

[42] Véronique Cortier, Stéphanie Delaune, and Graham Steel. A formal theory of key conjuring. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 79–93, Venice, Italy, July 2007. IEEE Computer Society Press.

[43] Véronique Cortier, Gavin Keighren, and Graham Steel. Automatic analysis of the security of xor-based key management schemes. In *13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 538–552, Braga, Portugal, March 2007. Springer.

[44] S. Delaune and F. Jacquemard. A decision procedure for the verification of security protocols with explicit destructors. In *Proc. 11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 278–287, Washington (Washington, USA), 2004. ACM Press.

[45] Stéphanie Delaune. Easy intruder deduction problems with homomorphisms. *Information Processing Letters*, 97(6) :213–218, March 2006.

[46] Stéphanie Delaune. An undecidability result for AGh. *Theoretical Computer Science*, 368(1-2) :161–167, December 2006.

[47] Stéphanie Delaune and Francis Klay. Vérification automatique appliquée à un protocole de commerce électronique. In *Actes des 6èmes Journées Doctorales Informatique et Réseau (JDIR'04)*, pages 260–269, Lannion, France, November 2004.

[48] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 28–39, Venice, Italy, July 2006. IEEE Computer Society Press.

[49] Stéphanie Delaune, Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Symbolic protocol analysis in presence of a homomorphism operator and *exclusive or*. In Michele Buglesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP'06) — Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 132–143, Venice, Italy, July 2006. Springer.

[50] Stéphanie Delaune, Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Symbolic protocol analysis for monoidal equational theories. *Information and Computation*, 206(2-4) :312–351, February-April 2008.

[51] G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. In *Proc. of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, Barcelone (Spain), 2004.

[52] D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. In R.L. Rivest, A. Sherman, and D. Chaum, editors, *Proc. of CRYPTO 82*, pages 177–186. Plenum Press, 1983.

[53] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. of the Workshop on Formal Methods and Security Protocols*, 1999.

[54] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Technical Report*. IEEE Computer Society Press, 1983.

[55] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology (AUSCRYPT'92)*, volume 718 of *LNCS*, pages 244–251. Springer, 1992.

[56] M. Girault and J.C. Paillès. Contactless EP : A Public-Key Solution with Good Performances. 2001.

[57] M. Roger J. Goubault-Larrecq and K. N. Verma. Abstraction and resolution modulo AC : How to verify diffie-hellman-like protocols automatically. *Journal of Logic and Algebraic Programming*, 2004. To appear.

[58] Florent Jacquemard, Michaël Rusinowitch, and Laurent Vigneron. Compiling and verifying security protocols. In Michel Parigot and Andrei Voronkov, editors, *LPAR*, volume 1955 of *Lecture Notes in Computer Science*, pages 131–160, Reunion Island, France, November 2000. Springer-Verlag.

[59] K. Kim, J. Kim, B. Lee, and G. Ahn. Experimental design of worldwide internet voting system using PKI. In *SSGRR'01*, L' Aquila (Italy), 2001.

[60] Steve Kremer, Yassine Lakhnech, and Ralf Treinen. The PROUVÉ manual : Specifications, semantics, and logics. Technical Report 7, projet RNTL PROUVÉ, December 2005. 49 pages.

[61] Steve Kremer and Mark D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In Mooly Sagiv, editor, *Programming Languages and Systems — Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200, Edinburgh, U.K., April 2005. Springer.

[62] Pascal Lafourcade. Intruder deduction for the equational theory of *exclusive-or* with commutative and distributive encryption. In Maribel Fernández and Claude Kirchner, editors, *Proceedings of the 1st International Workshop on Security and Rewriting Techniques (SecReT'06)*, volume 171 of *Electronic Notes in Theoretical Computer Science*, pages 37–57, Venice, Italy, July 2007. Elsevier Science Publishers.

[63] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In Jürgen Giesl, editor, *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *Lecture Notes in Computer Science*, pages 308–322, Nara, Japan, April 2005. Springer.

[64] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. ACUNh : Unification and disunification using automata theory. In Jordi Levy, editor, *Proceedings of the 20th International Workshop on Unification (UNIF'06)*, pages 6–20, Seattle, Washington, USA, August 2006.

[65] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for the equational theory of Abelian groups with distributive encryption. *Information and Computation*, 205(4) :581–623, April 2007.

[66] Yassine Lakhnech and Laurent Mazaré. Probabilistic opacity for a passive adversary and its application to chaumś voting scheme. Technical Report 4, Verimag, 2005.

[67] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, march 1996.

[68] G. Lowe. Casper : A compiler for the analysis of security protocols. In *Proc. of 10th Computer Security Foundations Workshop (CSFW'97)*. IEEE Computer Society Press, 1997.

[69] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proc. of the 11th Computer Security Foundations Workshop (CSFW'98)*. IEEE Computer Society Press, 1998.

[70] Gavin Lowe. A hierarchy of authentication specification. In *10th Computer Security Foundations Workshop*, pages 31–44, Rockport, MA, USA, June 1997. IEEE Computer Society.

[71] L. Mazaré. Decidability of opacity with non-atomic keys. In *Proc. of the second international Workshop on Formal Aspects in Security and Trust (FAST2004), Toulouse, France*, 2004.

[72] L. Mazaré. Satisfiability of Dolev-Yao constraints. In *Proc. of Automated Reasoning for Security Protocol Analysis(ARSPA'04)*, 2004.

[73] L. Mazaré. Using unification for opacity properties. In *Proc. of the Workshop on Issues in the Theory of Security (WITS'04)*, 2004.

[74] Catherine Meadows. Language generation and verification in the NRl protocol analyzer. In *Proc. of the 9th Computer Seurity Foundation Workshop (CSFW'96)*. IEEE Computer Society Press, 1996.

[75] Jonathan K. Millen and Grit Denker. CAPSL and MuCAPSL. *Journal of Telecommunications and Information Technology*, 4 :16–26, 2002.

[76] M. Ohkubo, F. Miura, M. Abe, A. Fujioka, and T. Okamoto. An improvement on a practical secret voting scheme. In *Proc. 2nd International Workshop on Information Security (ISW'99)*, volume 1729 of *LNCS*, pages 225–234. Springer, 1999.

[77] Hitoshi Ohsaki and ralf Treinen. Translation from PROUVÉ to ACTAS. Technical Report 11, projet RNTL PROUVÉ, May 2007. 12 pages.

[78] Hitoshi Ohsaki and Toshinori Takai. Actas : A system design for associative and commutative tree automata theory. *Electr. Notes Theor. Comput. Sci.*, 124(1) :97–111, 2005.

[79] L. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. of the 10th Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.

[80] R. Ramanujam and S.P.Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proc. of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, Mumbai, 2003.

[81] 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.

[82] P. Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol : a cautionary tale. *Information Processing Letters*, 65(1) :7–10, 1998.

[83] V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *Proc. of the 13th European Symposium On Programming (ESOP'04)*, LNCS, pages 355–369. Springer Verlag, April 2004.

[84] J. Traoré. Are blind signatures suitable for on-line voting ? (extended abstract). In *Proc. of Workshop Frontiers in Electronic Elections (FEE'05)*, Milan, Italy, 2005.

[85] M. Turuani. *Sécurié des protocoles cryptographiques : décidabilité et complexité*. PhD thesis, Université Henri Poincaré, Nancy, France, 2003.

[86] Mathieu Turuani. The CL-Atse Protocol Analyser. In *Term Rewriting and Applications - Proc. of RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286, Seattle, WA, USA, 2006.

[87] K. N. Verma. Two-way equational tree automata for ac-like theories : Decidability and closure properties. In R. Nieuwenhuis, editor, *Proc. of the 14th International Conference on Rewriting Techniques and Applications (RTA'03)*, LNCS, pages 180–196, Valencia, Spain, June 2003. Springer.