Vérification des protocoles cryptographiques avec le langage PDDL et les solveurs SAT

Noureddine Aribi, Yahia Lebbah

Vérification des protocoles cryptographiques avec le langage PDDL et les solveurs SAT

Noureddine Aribi[1] **Yahia Lebbah**[1,2]

[1]Univ. d'Oran Es-Sénia, Dép. Informatique, BP-1524, El-M'Naouer, Oran, Algérie
[2]Lab. I3S, UNSA-CNRS, 930 Route des Colles, BP 145, 06903 Sophia Antipolis, France
{aribi_noureddine,ylebbah}@yahoo.fr

Résumé

Dans ce papier, nous introduisons une approche basée sur la planification pour détecter des attaques logiques sur les protocoles cryptographiques. Nous montrons que les protocoles cryptographiques peuvent être modélisés avec le langage de planification PDDL, et vérifiés avec un système de planification tel que Blackbox.

Notre approche est principalement inspirée des travaux de Compagna sur la planification via SAT des protocoles cryptographiques. Pour vérifier un protocole, Compagna se sert de l'environnement Avispa et ses différents outils de traduction, et finalement des solveurs SAT. Nous proposons une approche plus directe : modélisation en PDDL et vérification avec un planificateur SAT tel que Blackbox.

L'idée principale de notre approche est une formulation PDDL compacte et simple des termes à structures complexes, sans employer des techniques de concrétisation proposées par Compagna. En particulier, nous avons réussi à détecter l'attaque par confusion de type, qui ne peut pas être trouvée avec l'approche de Compagna.

Ce travail contribue à la modélisation directe en langage de planification PDDL et la résolution avec des solveurs SAT.

Abstract

In this paper, we introduce a planning based approach to detect logical attacks against cryptographic protocols. We show that cryptographic protocols can be modelled with PDDL general purpose planning language, and checked with a planning system such as Blackbox.

Our approach is mainly inspired from Compagna's works on a SAT and planning approach for cryptographic protocols. To check a protocol, Compagna makes use of many tools : Avispa's environment to translate a protocol description into a state transition system, which is translated into a planning problem, and converted into SAT problem, and finally solved with a state of the art SAT solvers. We propose a more simplified planning based approach to check security protocols : modelling into PDDL and checking with a planning system mainly Blackbox.

The main idea in our approach is a compact and simple PDDL expression of complex term structures, without using flattening and grounding techniques proposed by Compagna. Furthermore, we have succeeded to detect type flaw attacks, which can not be handled with Compagna's approach.

This work contributes to model directly with PDDL and solve with planning and SAT solvers.

1 Introduction

Les protocoles cryptographiques sont de petits programmes concurrents conçus pour garantir la sécurité des échanges entre les participants dans un réseau peu sûr au moyen de primitives cryptographiques. En dépit de leur simplicité apparente, la conception des protocoles cryptographiques est difficile et sujette aux erreurs. Dans certains protocoles, des attaques ont été trouvées plusieurs années après leur conception. Les failles dans certains protocoles peuvent avoir des conséquences catastrophiques. Il est donc incontournable d'analyser automatiquement des protocoles de sécurité d'une manière rigoureuse et formelle.

Plusieurs approches ont été proposées pour analyser les protocoles cryptographiques. Des méthodes basées sur l'induction et la preuve de théorèmes ont été développées dans [18, 5]. Ces méthodes sont très générales. Elles peuvent traiter des protocoles avec plusieurs dimensions non bornées (nombre d'exécutions parallèles et séquentielles, nombre de principaux, *nonces*, et clefs fraîches), et permettent d'obtenir des preuves de correction. Mais, à la différence du model-checking, elles ne sont pas complètement automatiques

(nécessite une interaction avec l'utilisateur). Et puis, quand un prouveur échoue, il est difficile d'obtenir des contre-exemples, et les attaques possibles sur le protocole. D'autres méthodes sont basées sur la modélisation des protocoles en logique de Horn (*e.g.* programmes Prolog) comme dans [2] et le développement des stratégies appropriées de preuve. Ces méthodes sont automatiques, elles peuvent manipuler des protocoles non bornés et prouver leur correction. Cependant, la terminaison de l'analyse n'est pas garantie en général. En plus, l'abstraction peut également mener aux fausses attaques.

Dans cet article, notre objectif est la proposition d'une démarche simple, exploitant la PPC, pour modéliser et vérifier plus facilement les protocoles cryptographiques[1], tout en gardant des performances comparables par rapport aux approches existantes.

Justement, nous montrons que les protocoles cryptographiques peuvent être modélisés correctement avec le langage de planification PDDL [2] et les attaques peuvent être trouvées en utilisant les planificateurs basés sur SAT tels que Blackbox [3] [15]. Le facteur clé dans notre approche est que nous procédons en deux étapes seulement : *(1)* modélisation avec le langage de planification PDDL, et *(2)* vérification automatique avec des systèmes de planification en particulier BlackBox.

A notre connaissance, Compagna [4] est le premier pionnier dans l'exploitation efficace des langages de planification pour analyser des protocoles cryptographiques. Son approche est basée sur trois étapes : *(1)* modélisation du problème d'insécurité du protocole (PIP) sous forme d'un système état/transition écrit en format intermédiaire IF, qui est utilisé par les divers outils de vérification incorporés dans l'environnement AVISPA ; *(2)* le format intermédiaire ainsi obtenu est converti en un problème de planification spécifié avec le langage SATE ; *(3)* et finalement la génération d'une formule CNF résolue avec un solveur SAT.

Nous améliorons l'approche de Compagna dans deux aspects :

Codage du protocole en PDDL Nous avons exploité le langage PDDL pour modéliser l'intégralité du protocole et ses aspects de sécurité. Nous insistons sur le fait que PDDL facilite la spécification des protocoles même les plus complexes, sans recours à des techniques de concrétisation présentées par Compagna. Nous avons trouvé une manière simple de représenter en PDDL les termes complexes d'une manière très compacte, qui est aussi compacte que celle présentée dans le langage SATE ; les sections 3 et 4.1 les détaillent.

Investigation d'autres attaques possibles En particulier, l'attaque par *rejeu*[4] (en anglais *replay attack*) que nous avons trouvé dans le protocole Otway-Rees et l'attaque par *confusion de type* appliqués sur le protocole de paiement électronique (PME) (voir la section 6). À la différence du model-checker dédié (SATMC [4]) de Compagna, qui nécessite une hypothèse de *typage fort*, et par conséquent il ne peut pas trouver de telles attaques.

Par ailleurs, notre approche bénéficie de la structure compacte qu'offre l'algorithme Graphplan [1], et profite des progrès effectués dans le domaine des techniques SAT pour résoudre efficacement les problèmes de planification. Nous montrons l'intérêt de notre démarche sur deux protocoles significatifs LPD-MSR [5] du projet AVISPA et PME [6] du projet Prouvé. Ces protocoles sont bien conçus et sont utilisés en pratique. Leur vérification est donc un challenge pour toute approche de vérification.

La section 2 définit une terminologie souvent utilisée dans ce papier. Un exemple illustratif de notre approche est détaillé dans la section 3. La section 4 introduit les prémisses nécessaires pour modéliser les protocoles, à savoir la modélisation des termes complexes et du modèle de Dolev-Yao. Les deux sections 5 et 6 sont dédiées à la vérification des protocoles LPD-MSR et PME. La section 7 conclut ce document.

2 Terminologie

Protocole cryptographique Un protocole cryptographique (*c.-à-d.* protocole de sécurité) est un ensemble de règles d'échange de messages permettant à des agents de communiquer de façon sécurisée. Il se décrit de la manière suivante : à l'étape d'étiquette i, A(lice) envoie à B(ob) le message M

$$i. \quad A \rightarrow B : M$$

Agent ou participant Un protocole est défini sur

[1] Les protocoles ne doivent pas être confondus avec les algorithmes cryptographiques. Les algorithmes font appel à des opérateurs numériques très spécifiques, alors que les protocoles fonctionnent essentiellement avec des opérateurs logiques. Les outils dédiées aux applications des algorithmes cryptographiques, tels que Cryptol (www.cryptol.net), ne sont donc pas adéquats à la vérification des protocoles.

[2] Au long de cet article, le mot PDDL se rapportera au formalisme STRIPS.

[3] http ://www.cs.rochester.edu/ kautz/satplan/blackbox/

[4] Les attaques par *rejeu* sont des attaques de type *Man In the Middle* consistant à intercepter des messages et à les rejouer sans aucun changement.

[5] http ://www.avispa-project.org/library/LPD-MSR.html

[6] http ://www.lsv.ens-cachan.fr/prouve/

un certain nombre de participants, que l'on appelle agents. Ceux-ci peuvent être de deux types :
- *agents honnête* ou principal : qui est un participant (une personne ou un programme) légitime du protocole, ayant un comportement bien précis, et prévu à l'avance.
- *agents malhonnête* ou I(ntrus) : celui-ci représente un adversaire essayant de profiter d'une éventuelle faille du protocole pour obtenir une information secrète.

Session Il est souvent intéressant d'étudier plusieurs instances (parallèles ou séquentielles) d'un même protocole cryptographique. Pour cette raison, on appelle session de protocole, un ensemble d'échanges de messages entre plusieurs participants formant un ensemble cohérent et pouvant être répété.

Nonce Les *nonces* sont des nombres aléatoires utilisés une seule fois pour assurer la fraîcheur d'un message (*e.g.* Nx est un nonce généré par l'agent X).

Propriétés de sécurité Une propriété de sécurité est toute propriété qu'un protocole cherche à assurer. Les propriétés les plus fréquentes sont : le *secret* et l'*authentification*.

PDDL PDDL (Planning Domain Definition Language) [10, 11] est un langage de description de domaines et de problèmes de planification. Ce langage est très expressif, extensible et permet de décrire des domaines assez complexes. Il permet de prendre en compte certains aspects temporels et de gestion des ressources.

3 Exemple illustratif

Considérons un exemple très classique : le protocole de Needham Schroeder à clef public (NSPK) [17], qui est basé sur la cryptographie asymétrique, où chaque agent possède une clef privée et une clef publique correspondante.

La figure suivante présente la façon dont devrait se dérouler le protocole selon sa spécification, où Ka représente la clef publique de A, et $\{m\}_k$ représente le message m chiffré avec la clef k, alors que N_A et N_B sont des *nonces* choisis par A et B respectivement.

1. $A \rightarrow B : \{A,\ Na\}_{Kb}$
2. $B \rightarrow A : \{Na,\ Nb\}_{Ka}$
3. $A \rightarrow B : \{Nb\}_{Kb}$

Ce protocole permet une authentification mutuelle entre les deux communicants A et B, afin d'établir un secret partagé entre eux. Ce secret peut être une clef symétrique, utilisée ultérieurement pour chiffrer leurs communications plus efficacement.

Dans le reste de cette section, nous illustrons notre approche en décrivant brièvement les spécifications du protocole NSPK en PDDL. À savoir, comment modéliser les deux sessions concurrentes de ce protocole menant à la fameuse attaque de Lowe [16].

Initialement, nous spécifions le domaine *cryptoprotocol* qui contient la déclaration de tous les *types*, *constantes*, *prédicats* et *actions* nécessaires à la représentation du comportement des deux agents honnêtes et l'intrus (*c.f.* section 4.2).

En outre, nous considérons quelques termes complexes, *e.g.* $\{\langle Na,\ Nb\rangle\}_{Ka}$ (deuxième règle du NSPK) qui peut être écrite dans la logique d'ordre un comme : $encrypt(pair(Na,\ Nb),\ Ka)$. Ce dernier peut être écrit avec une syntaxe plus proche de PDDL, ainsi nous aurons : $pair(Na,\ Nb,\ NaNb)\ \wedge\ encrypt(\ NaNb,\ Ka,\ C)$, où C est le texte chiffré résultant. Maintenant, le passage vers PDDL[7] devient plus facile à réaliser[8] :

```
(and (pair ?Na ?Nb ?Na_Nb) (encrypt ?Na_Nb ?Ka ?C))
```

Chaque agent maintient une liste dynamique de connaissances qui est représentée par le prédicat `know`, *e.g.* si *Alice* connait son propre *nonce* Na et la clef publique Kb de *Bob* dans la session d'identificateur Sid, alors sa liste de connaissances (*c.-à-d.* $[Na,\ Kb]$) peut être spécifiée comme :

```
(and (know ?a ?na ?sid) (know ?a ?kb ?sid) )
```

Nous nous servons de deux prédicats : `witness` et `request` pour modéliser la propriété d'authentification à vérifier, *e.g.* `(witness ?S ?R ?Nb ?Sid)`, signifie que l'agent honnête S veut exécuter le protocole avec l'agent R en employant Nb comme valeur d'authentification dans la session d'identificateur Sid.

Afin de modéliser le comportement de tous les agents honnêtes impliqués dans le protocole NSPK, quatre actions doivent être spécifiées. Par exemple, la première action associée à la première étape, comporte trois parties : *(i)* les paramètres (*e.g.* $?a\ ?kb$), éventuellement typés en utilisant un trait d'union "–" entre le paramètre et le nom du type ; *(ii)* les formules préconditionnelles contenant les prédicats d'index i, et *(iii)* les effets qui sont divisés en *adds* correspondant aux prédicats positifs, et *deletes* correspondant aux prédicats négatifs, comme nous pouvons le voir dans la spécification suivante :

[7] En PDDL les prédicats et les conditions sont exprimés en notation préfixée (syntaxe lispienne).

[8] Les paramètres sont distingués par leur commencement avec un point d'interrogation (" ?").

```
(:action step1
 :parameters (?a ?b - User ?ka ?ka-1 ?kb - PublicKey
              ?sid - Session ?na - Nonce ?a_na_kb - Encrypt
             ?a_na - Pair)
 :precondition
   (and (state zero ?a ?a ?sid) (inv-key ?ka ?ka-1)
        (know ?a ?b ?sid)  (know ?a ?ka ?sid)
        (know ?a ?ka-1 ?sid) (know ?a ?kb ?sid) )
 :effect
   (and (not (state zero ?a ?a ?sid)) (state two ?b ?a ?sid)
        (ft ?a ?sid ?na) (know ?a ?na ?sid) (pair ?a ?na ?a_na)
   (encrypt-pair ?a_na ?kb ?a_na_kb)(msg one ?a ?b ?a_na_kb)
   (witness ?a ?b ?na ?sid) ) )
```

où ft et state sont des prédicats qui modélisent respectivement les *termes frais* (*e.g.* les nonces, les clefs), et l'état courant de l'agent. Les capacités de l'intrus (voir section 4.2) sont également nécessaires afin d'accomplir la spécification du protocole.

Habituellement, une fois que nous avons fini avec la description du domaine, nous passons à la définition du problème, où nous définissons tous les objets nécessaires (*e.g.* `a b i - User`, `fst snd - Session`), les états initiaux (*e.g.* `(ik a)(ik ki)` où ik est un prédicat qui permet de mettre à jour la liste des connaissances de l'intrus avec une donnée passée en argument) et le but à atteindre, *c.-à-d.* : `( :goal (and (witness a i na fst) (request b a na snd) ))`. Avec ce but, nous voulons anticiper un plan qui finit par un *mauvais* état où l'intrus obtient avec succès l'information secrète *na*.

Finalement, nous invoquons le système de planification blackbox afin de résoudre le problème de planification décrit ci-dessus. Ainsi on saisi la ligne de commande suivante :

```
blackbox -o NSPK-Domain.pddl -f NSPK-Problem.pddl
         -solver chaff
```

Le résultat est correct. En effet, blackbox a généré une formule CNF wff comportant 750 variables et 1106 clauses, puis le solveur chaff a trouvé un modèle en 165 *ms*. Tandis que, le système de Compagna a généré une formule CNF wff avec 423 variables et 1276 clauses et il l'a résolue en 140 *ms*. Ces résultats sur NSPK montrent que notre approche, malgré sa simplicité, a des performances qui sont similaires à celles de Compagna.

Ce protocole ne préserve pas la propriété requise comme illustrée à travers la trace d'attaque (*contre-exemple*) suivante.

```
Begin plan
1 (step1 a i ka ka-1 ki fst na a-na-ki a-na)
2 (divert one a i a-na-ki)
3 (decrypt-pair ki ki-1 a-na-ki a-na)
4 (impersonate2-2 a b ka kb kb-1 snd na a-na-kb a-na)
5 (step2 a b ka kb kb-1 snd na nb a-na-kb na-nb-ka na-nb)
6 (divert two b a na-nb-ka)
7 (impersonate3-1 a i ka ka-1 ki fst na nb na-nb-ka na-nb)
8 (step3 a i ka ka-1 ki fst na nb na-nb-ka nb-ki)
9 (divert three a i nb-ki)
10 (decrypt ki ki-1 nb nb-ki)
11 (impersonate4-2 a b ka kb kb-1 snd na nb nb-kb)
12 (step4 a b ka kb kb-1 snd na nb nb-kb)
End plan
```

où divert est une action qui permet à l'intrus de détourner les messages échangés entre les principaux.

En considérant la première règle du protocole NSPK, on aperçoit immédiatement que le message $\{\langle A,\ Na\rangle\}_{Kb}$ peut être composé par l'intrus si l'ensemble de ses connaissances contient au moins un des trois ensembles de messages : $\{\langle A,\ Na\rangle\}_{Kb}\}$, $\{\langle A,\ Na\rangle,\ Kb\}$ ou $\{A,\ Na,\ Kb\}$. Par conséquent, nous créons trois actions (impersonate2-1, impersonate2-2 et impersonate2-3) permettant à l'intrus de jouer le rôle d'un agent honnête A.

À travers cet exemple, nous avons découvert l'attaque de Lowe sur ce protocole en deux étapes seulement : modéliser le problème d'insécurité du protocole en PDDL, et enfin le résoudre avec un solveur SAT.

4 Préliminaires

4.1 Représentation des termes complexes avec le langage de planification PDDL

Le langage PDDL est utilisé par les systèmes de planification standards pour spécifier les problèmes de planification. À première vue, il semble ne pas permettre la modélisation des termes à structures complexes tels que $p(X,\ f(X,\ g(Y)))$ puisque $f(X,\ g(Y))$ n'est pas un symbole individuel mais un terme plus complexe. À travers l'exemple suivant tiré de [4] (page 163), nous verrons comment achever cette tâche d'une manière flexible.

Soit l'opérateur de planification suivant

$$\{p(X,\ f(X,\ g(Y))),\ q(X)\} \overset{foo(X,Y)}{\Rightarrow} \{q(Y);\ p(X,\ f(X,\ g(Y)))\}$$

cet opérateur est réécrit de sorte qu'une nouvelle variable est créée et associée à chaque prédicat imbriqué, *c.-à-d.* :

$$\{g(Y) = Z_1,\ f(X,\ Z_1) = Z_2,\ p(X,\ Z_2),\ q(X)\}$$

$$\overset{foo(X,Y,Z_1,Z_2)}{\Longrightarrow} \{q(Y);\ p(X,\ Z_2)\}$$

Finalement, la formule obtenue est spécifiée en PDDL comme suit :

```
(:action foo
 :prameters (?X ?Y ?Z1 ?Z2)
 :precondition  (and (g ?Y ?Z1)(f ?X ?Z1 ?Z2)(p ?X ?Z2)(q ?X)))
 :effect        (and (q ?Y) (not (p ?X ?Z2)) ))
```

Il est clair que des termes complexes peuvent être correctement spécifiés en notation STRIPS en introduisant de nouvelles variables, sans recourir aux techniques de concrétisation de [4]. Ces techniques de concrétisation procèdent en deux étapes : *(1)* prendre

toutes les combinaisons du produit cartésien des domaines des variables de la formule complexe, *(2)* créer une action pour chaque combinaison. Ainsi dans l'exemple donné, si on suppose que $X, Y \in \{1, 2\}$, alors la concrétisation du terme complexe génère quatre actions :

```
(:action |foo_1_1|
 :prameters ( )
 :precondition  (and (p |1| |f(1,g(1))|)
                     (q |1|))
 :effect        (and (q |1|)
                     (not (p |1| |f(1,g(1))|))))

(:action |foo_1_2|
 :prameters ( )
 :precondition  (and (p |1| |f(1,g(2))|)
                     (q |1|))
 :effect        (and (q |2|)
                     (not (p |1| |f(1,g(2))|))))

(:action |foo_2_1|
 :prameters ( )
 :precondition  (and (p |2| |f(2,g(1))|)
                     (q |2|))
 :effect        (and (q |1|)
                     (not (p |2| |f(2,g(1))|))))

(:action |foo_2_2|
 :prameters ( )
 :precondition  (and (p |2| |f(2,g(2))|)
                     (q |2|))
 :effect        (and (q |2|)
                     (not (p |2| |f(2,g(2))|))))
```

sachant que si `t` est un terme simple alors `|t|` est la constante individuelle appropriée qui représente `t`.

Notez que le nombre exact d'actions ainsi générées est du même ordre que le nombre de combinaisons du produit cartésien des domaines des variables du terme. Ce nombre est donc de nature exponentielle. Alors qu'avec notre approche on garde seulement une seule action.

4.2 Modèle de Dolev et Yao

La plupart des travaux en vérification des protocoles cryptographiques utilise un modèle formel, appelé le modèle de Dolev-Yao [9]. Ce modèle repose sur deux hypothèses importantes : *(i)* les algorithmes de chiffrement sont parfaits (l'intrus doit savoir la clef inverse pour obtenir le texte en clair à partir du texte chiffré) et *(ii)* les *nonces* sont idéalement générés (Il ne doit pas y avoir de confusion, et ne peuvent pas être "devinés").

Nous montrerons ci-dessous comment modéliser en PDDL les capacités de l'intrus de Dolev-Yao à partir des *règles de réécriture (Multi-)ensembles*[9] [4].

La première règle modélise la capacité de l'intrus à détourner des messages échangés entre les agents honnêtes à travers un réseau public, *c.-à-d.* :

$$\mathsf{msg}(I,\ A,\ B,\ M) \xrightarrow{divert(A,B,I,M)} \mathsf{ik}(M)$$

cette règle peut être écrite en PDDL :

```
(:action divert
 :parameters (?I - Number ?A ?B - User ?M)
 :precondition (msg ?I ?A ?B ?M)
 :effect       (ik ?M) )
```

où msg est un prédicat qui modélise les messages échangés entre agents. Cette action sera activée une fois que le $I^{\text{ème}}$ message (M) est créé et est envoyé à son destinataire B. Ce qui aura pour effet l'ajout de ce dernier à la liste des connaissances de l'intrus.

La deuxième règle modélise la capacité à décomposer des paires de messages et à les mémoriser, *c.-à-d.* :

$$\mathsf{ik}(\langle M_1, M_2\rangle) \xrightarrow{decompose(M_1,M_2)} \mathsf{ik}(\langle M_1, M_2\rangle)\ .\ \mathsf{ik}(M_1)\ .\ \mathsf{ik}(M_2)$$

L'action équivalente peut être écrite en PDDL comme suit :

```
(:action decompose
 :parameters (?M1 - (either User Nonce) ?M2 - Nonce ?M - Pair)
 :precondition      (and  (pair ?M1 ?M2 ?M)(ik ?M))
 :effect            (and  (ik ?M1) (ik ?M2)) )
```

Et la troisième action modélise la capacité à déchiffrer les messages chiffrés, *c.-à-d.* :

$$\mathsf{ik}(\{M\}_K)\ .\ \mathsf{ik}(K^{-1}) \xrightarrow{decrypt(K,M)} \mathsf{ik}(\{M\}_K)\ .\ \mathsf{ik}(K^{-1})\ .\ \mathsf{ik}(M)$$

où K^{-1} est la clef inverse de K. Cette règle peut être spécifiée en PDDL comme suit :

```
(:action decrypt
 :parameters (?K ?K-1 - PublicKey ?C - Encrypt ?M)
 :precondition (and (encrypt ?M ?K ?C)(inv-key ?K ?K-1)
                    (ik ?C)(ik ?K-1))
 :effect            (ik ?M) )
```

tels que inv-key désigne un prédicat utilisé pour modéliser la relation *key-paire* (*c.-à-d.* paire de clefs) entre une clef privée et sa clef publique correspondante.

En plus des actions mentionnées ci-dessus, l'intrus peut employer d'autres primitives (*e.g.* hachage, signature) et peut appliquer d'autres attaques (*e.g.* rejeu de vieux messages, et usurpation d'identifications des agents honnêtes).

Axiomes

Une amélioration importante peut être faite en étendant le problème d'insécurité des protocoles afin de permettre la spécification d'axiomes. Un axiome est une formule qui énonce une relation entre les faits d'un système de transition, et reste valide dans chaque état de ce système. La différence principale entre un axiome et une action est dans le fait qu'un axiome persiste dans chaque état du système de transition, alors qu'une action n'est pas forcée à être exécutée même si toutes ses préconditions sont satisfaites. Les axiomes peuvent être appliqués dans deux directions :

[9] L'idée principale consiste à introduire l'opérateur "." qui est associatif et commutatif (AC).

1. pour représenter les relations entre les connaissances de l'intrus. Par exemple, la règle de déchiffrement de l'intrus (vue ci-dessus) peut être modélisée par un axiome comme suit :

$$ik(\{M\}_K) \wedge ik(K^{-1}) \supset ik(M)$$

cet axiome peut être spécifié en PDDL :

```
(:axiom
 :vars (?C - Encrypt ?M - Message ?K ?K-1 - Key)
 :context (and (encrypt ?M ?K ?C)(inv-key ?K ?K-1)
               (ik ?C)(ik ?K-1))
 :implies (ik ?M) )
```

2. pour modéliser les équations algébriques [6] utilisées pour spécifier les propriétés particulières des primitives cryptographiques. Par exemple, le protocole d'échange de clefs de Diffie-Hellman [8] dépend de la propriété de l'exponentielle modulaire suivante : $(g^X)^Y = (g^Y)^X$. Cette propriété peut être modélisée avec l'axiome suivant :

$$ik((g^X)^Y) \supset ik((g^Y)^X)$$

où g est une base (un nombre *premier*), X et Y sont deux nombres entiers secrets, utilisés pour produire une clef de session secrète partagée entre deux participants. Cette clef est notée : $(g^X)^Y$ ou encore $(g^Y)^X$ (on dit que $(g^X)^Y$ et $(g^Y)^X$ sont dans la même classe d'équivalence).

L'axiome décrit ci-dessus peut être réécrit dans une syntaxe proche de PDDL en créant une fonction *exp* qui modélise la fonction de l'exponentielle modulaire, et deux nouvelles variables Z_1 et Z_2 telles que $Z_1 = (g^X)^Y$ et $Z_2 = (g^Y)^X$, ainsi nous obtenons :

$$\{exp(G, X, Y, Z_1),\ exp(G, Y, X, Z_2),\ ik(Z_1)\} \supset ik(Z_2)$$

et enfin l'axiome résultant peut être spécifié en PDDL comme suit :

```
(:axiom
 :vars (?X ?Y ?Z1 ?Z2)
 :context (and (g ?X ?Y ?Z1) (g ?Y ?X ?Z2) (ik ?Z1))
 :implies (ik ?Z2) )
```

La vérification des protocoles cryptographiques n'exige pas la modélisation de ces axiomes, puisqu'ils sont déjà dans les actions. Plus précisément, les axiomes sont utilisés pour réduire l'espace de recherche du problème de planification.

Dans l'étude des protocoles des sections suivantes, nous avons employé Blackbox qui dispose de l'encodage Graphplan et ses optimisations. En contre partie, Blackbox supporte seulement la notation STRIPS qui ne dispose pas d'une syntaxe pour exprimer les axiomes. SATPALN qui supporte PDDL 2.0 contenant une notation pour les axiomes, pourrait également être employé pour tirer profit de la réduction de l'espace de recherche induite par ces axiomes ; en revanche l'encodage Graphplan n'est pas intégré. Nous avons ainsi préféré employer Blackbox/STRIPS pour prouver que le langage *de base* STRIPS est suffisant pour vérifier les protocoles cryptographiques.

5 LPD-MSR

5.1 Spécification informelle

Le protocole LPD (Low-Powered Devices) MSR (Modulo Square Root) est un protocole d'établissement de clefs pour des communications mobiles sécurisées. Un tel protocole est basé sur un cryptosystème à clef publique pour lequel le chiffrement est particulièrement efficace. La technique MSR permet au chiffrement à clef publique d'être implémenté dans une station mobile M à puissance très réduite (microprocesseur – appelé aussi *cryptoprocesseur* –, mémoire et batterie). La description semi-formelle suivante est divisée en deux parties : déclaration des données et la séquence des messages échangés.

$B,\ M$: User
Kb : PublicKey
SCm : Text
X : SymmetricKey (fresh)

1. $B \longrightarrow M : B,\ Kb$
2. $M \longrightarrow B : \{X\}_{Kb}$
3. $M \longrightarrow B : \{M,\ SCm\}_X$

Les objets SCm et X dénotent respectivement le certificat secret du mobile M qui est fourni par une autorité centrale de confiance, et la clef de session.

Propriétés à vérifier

Ce protocole est censé garantir deux propriétés de sécurité à savoir : *(i)* l'*authentification forte* et *(ii)* la *confidentialité* de la clef de session X partagée entre la station de base B et le mobile M.

5.2 Spécification PDDL

Pour modéliser ce protocole en PDDL, nous devons définir quelques prédicats, tels que : `(secret ?R ?S - User ?SK - SymmetricKey)` qui est utilisé pour vérifier la propriété de confidentialité de la clef de session SK partagée entre les utilisateurs R et S. De plus, de nouveaux types sont également créés, comme *SymmetricKey* pour l'ensemble des clefs de session, et *Text* pour l'ensemble des certificats secrets des mobiles.

Par ailleurs, nous définissons trois actions afin de décrire le comportement des agents honnêtes. Par exemple, nous montrerons ci-dessous la deuxième action qui modélise les deux derniers messages échangés, fusionnés dans une seule action, de telle sorte que la clef de session secrète X soit établie entre la station de base B et le mobile M. Cette dernière clef est prévue pour être utilisée durant toute la session[10].

```
(:action step2
 :parameters (?b ?m - User  ?kb ?kb-1 - PublicKey ?scm - Text
  ?sid - Session ?m_scm_skey ?skey_kb - Encrypt ?b_kb ?m_scm
  ?kb_m_scm_skey - Pair ?skey - SymmetricKey)
 :precondition
(and
    (msg one ?b ?m ?b_kb) (state one ?b ?m ?sid)
    (pair ?b ?kb ?b_kb) (know ?m ?b ?sid) (know ?m ?scm ?sid)
    (inv-key ?kb ?kb-1)
)
 :effect
(and
     (not (msg one ?b ?m ?b_kb)) (not (state one ?b ?m ?sid))
     (state three ?b ?m ?sid)
 ; génération d'une clef de session secrète 'skey'
     (secret ?b ?m ?skey)
 ; mise à jour des connaissances du mobile 'm'
     (know ?m ?b ?sid) (know ?m ?kb ?sid) (know ?m ?skey ?sid)
 ; création d'un message crypté par le mobile
     (crypt ?skey ?kb ?skey_kb) (pair ?m ?scm ?m_scm)
     (encrypt-pair ?m_scm ?skey ?m_scm_skey)
     (pair ?skey_kb ?m_scm_skey ?kb_m_scm_skey)
 ; envoie de ce message à la station de base 'b'
     (msg two ?m ?b ?kb_m_scm_skey)
 ; authentification des principaux
     (witness ?m ?b ?skey ?sid) ) )
```

Les règles de l'intrus sont celles basées sur le modèle de Dolev-Yao mentionné ci-dessus (voir la section 4.2). Pour vérifier si le protocole contient une faille de confidentialité ou non, nous pouvons écrire le but suivant : `( :goal (ik seckey) )`.

B. Blanchet [3] a fait un premier pas vers un lien formel entre les deux propriétés d'authentification et de secret, où la preuve du secret suffit à assurer l'authentification. L'attaque de secret peut donc mener à une attaque d'authentification. Enfin, la propriété d'authentification peut être vérifiée en spécifiant le but suivant : `( :goal (and (witness m b seckey snd) (request b i seckey fst)))`

Une fois la définition des domaines faite, nous invoquons blackbox sur le modèle PDDL obtenu en utilisant la même ligne de commande vue dans le protocole NSPK ; ainsi nous obtenons la trace d'attaque illustrée dans le plan ci-dessous.

```
Begin plan
1 (step1 b i kb kb-1 fst b-kb)
2 (divert one b i b-kb)
3 (decompose b kb b-kb)
4 (impersonate2-2 b m ki kb snd b-ki b-kb)
5 (step2 b m ki ki-1 seckey scm snd m-scm-skey skey-ki b-ki
          m-scm ki-m-scm-skey)
6 (divert two m b ki-m-scm-skey)
7 (decompose skey-ki m-scm-skey ki-m-scm-skey)
8 (decrypt ki ki-1 seckey skey-ki)
9 (impersonate3-2 b i ki kb kb-1 seckey fst skey-ki skey-kb
          m-scm-skey kb-m-scm-skey ki-m-scm-skey)
10 (step3 b i kb kb-1 seckey scm fst b-kb kb-m-scm-skey
          skey-kb m-scm-skey)
End plan
```

Cette attaque est appelée "*Man In the Middle attack*". Elle est schématisée dans la figure suivante.

	$B,\ M,\ I$	:	User
	$Kb,\ Ki$	:	PublicKey
	SCm	:	Text
	X	:	SymmetricKey (fresh)
1.1.	$B \longrightarrow I(M)$	:	$B,\ Kb$
2.1.	$I(B) \longrightarrow M$	:	$B,\ Ki$
2.2.	$M \longrightarrow I(B)$	:	$\langle \{X\}_{Ki},\ \{M,\ SCm\}_X \rangle$
1.2.	$I(M) \longrightarrow B$	:	$\langle \{X\}_{Kb},\ \{M,\ SCm\}_X \rangle$

Le terme $I(X)$ signifie que l'intrus I joue le rôle de l'agent légitime X.

Cette attaque suffit pour *(i)* violer le secret de la clef de session *seckey* établie, et *(ii)* pour faire croire à la station de base B qu'elle a dialogué avec le mobile M, alors qu'en réalité elle a dialogué avec l'intrus.

Notez que blackbox a produit une formule CNF comportant 893 variables et 5687 clauses, et par la suite le solveur chaff a trouvé la faille de sécurité en 207 *ms*.

6 Protocole de Porte-Monnaie Électronique (PME)

Jusqu'ici, nous avons présenté deux attaques sur les propriétés de secret et d'authentification. Cependant, les protocoles cryptographiques peuvent assurer d'autres propriétés de sécurité. Par conséquent, d'autres attaques peuvent être appliquées comme nous allons les voir dans cette section.

6.1 Spécification Informelle

Le protocole PME [7] permet de réaliser une transaction entre un porte-monnaie électronique EP et un serveur SAM (Secure Application Module). Le but est de garantir un bon niveau de sécurité, avec un bon niveau de performance grâce à l'utilisation du chiffrement symétrique (très rapide par rapport au chiffrement asymétrique). Ce protocole est illustré dans la figure 1.

Au cours d'un premier échange, le porte-monnaie électronique EP envoie au serveur SAM sa propre identité *Id_EP* et un *challenge frais chall_EP* (msg 1). SAM calcule la valeur de *ep_acqkey*, qui est une clef partagée (construite en utilisant la fonction de hachage F_{ep_acqkey}), et elle sera utilisée pour *chiffrer* les prochains messages. SAM répond au *challenge* de EP

[10] Celle-ci s'appelle la propriété de secret *locale*. Les clefs à long terme sont vues en tant que clefs secrètes *globales*.

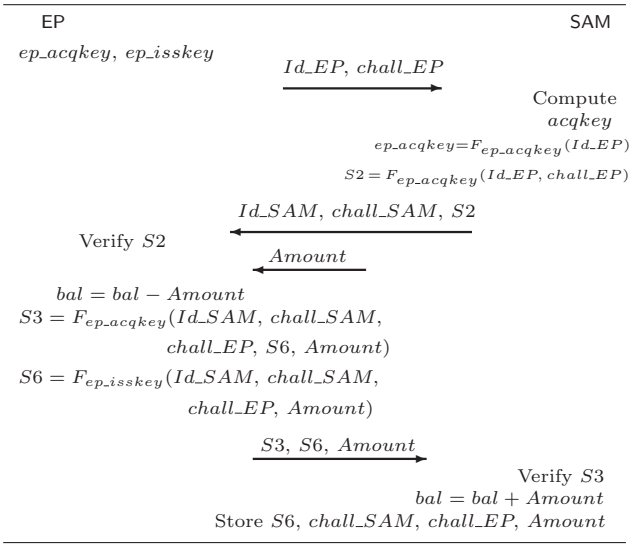


FIG. 1 – Protocole de Porte-Monnaie Électronique – Approche symétrique

et donne son identité Id_SAM accompagnée d'un nouveau *challenge* $chall\_SAM$ (msg 2). EP vérifie la réponse fournie par SAM et reçoit le montant $Amount$ de la transaction (msg 3). A ce moment, EP débite sa balance bal et répond à SAM (msg 4) en construisant un message contenant l'identité de SAM, les valeurs des deux *challenges*, le montant de la transaction ainsi qu'un message noté *S6*. SAM vérifie *S3* en reconstruisant le message et crédite sa balance si la vérification se passe correctement. Enfin, il stocke le message *S6* pour résoudre d'éventuels litiges ultérieurs (F_{ep_isskey} est une fonction de hachage secrète, qui est utilisée pour générer la clef secrète ep_isskey).

6.2 Spécification Formelle

En général, la majeure partie des spécifications est rédigée d'une façon informelle. La première partie du travail consiste à lever les ambiguïtés et produire une spécification formelle du protocole. Ce travail est délicat, car la spécification produite va endosser toute la confiance que l'on peut avoir dans le résultat de la vérification formelle. La modélisation de ce protocole a été faite par [7] en partenariat avec les concepteurs du protocole pour ne pas biaiser leur vision initiale. Un certain nombre de points, parmi lesquels ceux détaillés ci-dessous, ont ainsi pu être éclaircis [7] :

- Les fonctions F_{ep_acqkey} et F_{ep_isskey} ne doivent pas être vues comme des fonctions de chiffrement symétrique : elles ne sont pas inversibles. Autrement dit, il est impossible de retrouver les composantes du message $F_{ep\_isskey}(Id\_SAM, chall\_SAM, chall\_EP, Amount)$, même si nous connaissons la fonction F_{ep_isskey}. Ces deux fonctions, F_{ep_acqkey} et F_{ep_isskey}, doivent être vues comme des fonctions de hachage, et elles seront modélisées en tant que telles par la suite. Pour vérifier les messages tels que $S3$ et $S6$, la seule façon est de les reconstruire et d'effectuer une comparaison. De plus, le message $S6$ est considéré comme un HMAC[11] car il est haché en utilisant la clef privée ep_isskey connue uniquement par son détenteur (*c.-à-d. EP*).
- Le canal sur lequel circule le montant de la transaction est un canal sécurisé, l'intrus peut écouter sur ce canal, mais il ne peut pas intercepter et/ou modifier les messages qui y circulent.
- Ce protocole a été conçu pour le paiement local. L'objectif n'était donc pas d'obtenir un protocole dédié à Internet. Les sessions ont donc lieu séquentiellement, sans s'entrelacer (ni côté EP, ni côté SAM).

Tous les points cités plus haut sont implicites dans la description informelle et sont pourtant indispensables lorsque l'on souhaite formaliser le protocole et vérifier ses propriétés.

Propriété à vérifier

La propriété qu'on veut vérifier dans ce protocole est la propriété de *non création de fausse monnaie* qui peut se traduire en une propriété d'authentification forte (correspondance un à un), encore appelée *injective agreement*, avec accord sur certaines valeurs (ici le montant). Nous allons vérifier qu'à chaque fois que le serveur SAM termine une session avec EP en créditant sa balance d'un certain montant, le porte-monnaie EP a joué une session avec SAM et a débité sa balance du même montant.

Spécification PDDL

Au début, de nouveaux types de données doivent être créés, par exemple : *Money, HashedKey, Hash* et *RandomNum* en plus de ceux qui sont déjà précisés dans le protocole NSPK.

Pour modéliser les fonctions de hachage en PDDL, nous avons besoin du prédicat suivant : `(hash_fissqkey ?EP - User ?HiEP - HashedKey)`, qui modélise une *clef de hachage HiEP*, générée par l'utilisateur *EP*. Cette clef est utilisée pour construire

[11] Un HMAC (A keyed-Hash Message Authentication Code) est une donnée de taille fixe jointe à un message, qui est calculées en utilisant une fonction de hachage en combinaison avec une clef secrète. Un HMAC garantit l'authenticité et l'intégrité d'un message.

des messages hachés avec une clef (*keyed-hash messages*).

Par exemple, le prédicat `(hash-fissqkey ?SAM - User ?Nsam ?Nep - RundomNum ?M - Money ?HiEP - HashedKey ?S6 - Hash)` modélise une fonction de hachage (F_{ep_isskey}) qui accepte deux paramètres en entrée : un message $\langle SAM, Nsam, Nep, M\rangle$ et une clef secrète $HiEP$ (*ep_isskey*) afin de construire le message haché avec une clef, *c.-à-d.* $S6$ (voir la figure 1).

Parmi les prédicats qui sont nécessaires pour spécifier ce protocole, nous pouvons citer : `(amount ?M)`, `(debit ?M)` et `(credit ?M)` qui modélisent le montant à débiter ou à créditer dans la transaction courante.

Le modèle PDDL associé à ce protocole contient cinq actions décrivant le comportement de tous les agents honnêtes. L'intrus est également modélisé avec les actions de Dolev-Yao mentionnées dans la section des préliminaires. La spécification PDDL suivante se concentre sur la troisième action exécutée par le porte-monnaie EP (*c.f.* figure 1 – msg 4). Les étapes restantes peuvent être faites de la même manière.

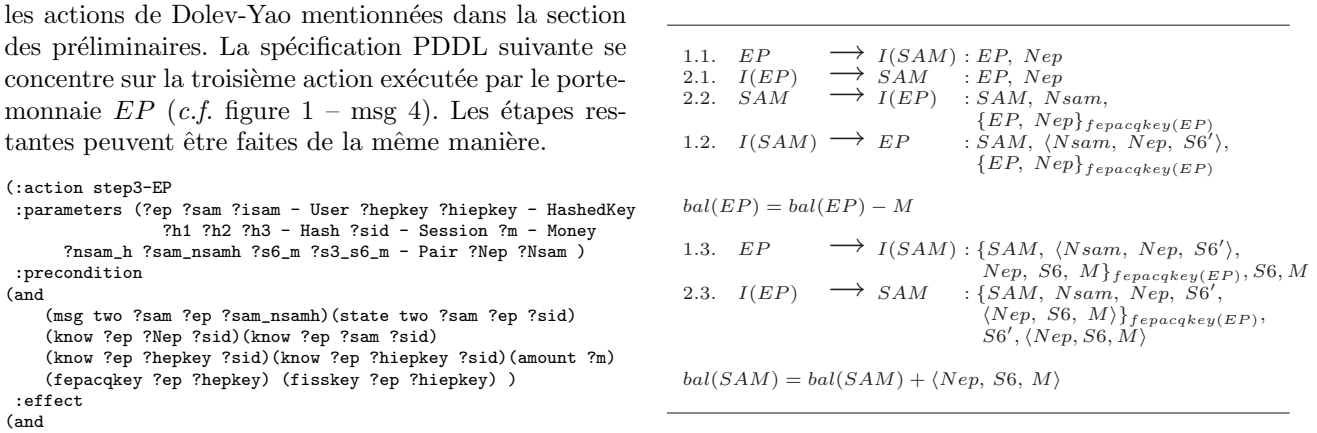
```
(:action step3-EP
 :parameters (?ep ?sam ?isam - User ?hepkey ?hiepkey - HashedKey
              ?h1 ?h2 ?h3 - Hash ?sid - Session ?m - Money
      ?nsam_h ?sam_nsamh ?s6_m ?s3_s6_m - Pair ?Nep ?Nsam )
 :precondition
(and
    (msg two ?sam ?ep ?sam_nsamh)(state two ?sam ?ep ?sid)
    (know ?ep ?Nep ?sid)(know ?ep ?sam ?sid)
    (know ?ep ?hepkey ?sid)(know ?ep ?hiepkey ?sid)(amount ?m)
    (fepacqkey ?ep ?hepkey) (fisskey ?ep ?hiepkey) )
 :effect
(and
    (not (msg two ?sam ?ep ?sam_nsamh))
    (not (state two ?sam ?ep ?sid)) (state four ?sam ?ep ?sid)
    (pair2 ?Nsam ?h1 ?nsam_h)(know ?ep ?Nsam ?sid)
    (debit ?m) (know ?ep ?m ?sid)
    (hash-fisskey ?sam ?Nsam ?Nep ?m ?hiepkey ?h2) ;S3
    (know ?ep ?h2 ?sid)
    (hash-fepacqkey2 ?sam ?Nsam ?Nep ?h2 ?m ?hepkey ?h3) ;S6
    (know ?ep ?h3 ?sid) (msg three ?ep ?sam ?s3_s6_m)
    (witness ?ep ?sam ?m ?sid) ) )
```

Afin d'exécuter cette action, EP devrait être dans le deuxième état, et a reçu le deuxième message de la part de SAM *c.-à-d.* $\langle Id\_SAM, chall\_SAM, S2\rangle$. De plus, EP devrait recevoir le montant de la transaction m. Par la suite, EP peut transiter vers le prochain état et produire un nouveau message représentant la concaténation des deux messages hachés avec une clef (*c.-à-d.* $S3$ et $S6$) et aussi avec le montant m précédemment débité.

Finalement, nous ramenons la propriété de *non création de fausse monnaie* à une propriété d'authentification avec accord sur la valeur du montant m, `( :goal (and (witness ep i m fst) (request sam ep mp snd)))`. L'attaque qui a été détectée est une attaque par *confusion de type* utilisant l'associativité de la paire, décrite dans le plan d'attaque suivant :

```
Begin plan
1 (step1-ep ep i hepkey hiepkey nep fst ep-nep)
2 (divert ep i ep-nep one)
3 (decompose1 ep nep ep-nep)
3 (impersonate-ep-2-1 ep sam hepkey nep snd ep-nep)
4 (step2-sam ep sam hepkey h nep nsam snd ep-nep nsam-h
                                                    sam-nsamh)
5 (divert sam ep sam-nsamh two)
6 (decompose3 sam nsam-h sam-nsamh)
7 (decompose2 nsam h nsam-h)
8 (impersonate-sam-3-1 ep i sam fst h nsam-h sam-nsamh
       nep nsam s6p nep-s6p nsam-nep-s6p nsamp-h sam-nsamph)
9 (step3-ep ep i sam hepkey hiepkey h s6 s3 fst m nsamp-h
            sam-nsamph s6p-mp s3-s6-m nep nsam-nep-s6p)
10 (divert ep i s3-s6-m three)
11 (decompose5 s3 s6p-mp s3-s6-m)
12 (decompose4 s6 m s6p-mp)
13 (impersonate-ep-4-1 ep sam hepkey snd s6 s3 s6p-mp s3-s6-m
          m hm mp nep nsam s6p s6-m s3p s3p-s6p-mp)
14 (step4-sam ep sam hepkey snd s6p s3p s6-m s3p-s6p-mp nep
                                                    nsam mp)
End plan
```

Le déroulement de cette attaque est clairement illustrer à travers la figure 2.



avec $S6 = \{SAM,\ \langle Nsam,\ Nep,\ S6'\rangle,\ Nep,\ M\}_{fepisskey(EP)}$ et $S6'$ un message arbitraire généré par l'intrus.

FIG. 2 – Attaque sur le protocole de Porte-Monnaie Électronique

Du coté porte-monnaie, la transaction a lieu avec un montant M, un montant qui est débité sur la carte, alors que coté serveur, le montant $\langle Nep,\ S6,\ M\rangle$ est crédité. En effet, si aucune vérification n'est faite, ces deux messages sont des suites de bits et peuvent être confondues.

Notez que blackbox a produit une formule CNF comportant 1613 variables et 6124 clauses. Puis le solveur chaff a trouvé un modèle en 484 *ms*.

7 Conclusion

Nous avons présenté une approche systématique pour la vérification des protocoles cryptographiques, en exploitant le langage de planification PDDL, les systèmes de planification, et les solveurs SAT.

Effectivement, nous avons réussi à vérifier des protocoles significatifs, à savoir les deux protocoles LPD-

MSR et EPP. En particulier, notre approche a permis de découvrir plus d'attaques telles que l'attaque par *confusion de types* et l'attaque par *rejeu*, qui ne peuvent pas être abordées avec l'approche de Compagna.

Nous devons expérimenter d'autres systèmes de planification (*e.g.* SATPLAN [14] et FF [12]) supportant la spécification PDDL 2.0 (et donc les axiomes), et voir s'ils sont meilleurs que Blackbox avec son codage à base de Graphplan.

Notre approche fournit de nouvelles possibilités pour vérifier des protocoles cryptographiques. Les protocoles industriels tels que le SSL contiennent divers opérateurs algébriques qui devraient être considérés. Nous pensons que nous devons étudier la coopération avec d'autres approches telles que SATPLAN ou GP-CSP [13].

Références

[1] A. Armando, L. Compagna, and P. Ganty. Sat-based model-checking of security protocols using planning graph analysis. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 875–893. Springer, 2003.

[2] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In IEEE, editor, *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, juin 2001.

[3] B. Blanchet. From secrecy to authenticity in security protocols. In *SAS '02 : Proceedings of the 9th International Symposium on Static Analysis*, pages 342–359, London, UK, 2002. Springer-Verlag.

[4] L. Compagna. *SAT-based Model-Checking of Security Protocols*. PhD thesis, Università degli Studi di Genova and the University of Edinburgh, 2005.

[5] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *14th IEEE Computer Security Foundations Workshop*, pages 97–108. IEEE Computer Society, 2001.

[6] S. Delaune. *Vérification des protocoles cryptographiques et propriétés algébriques*. PhD thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, 2006.

[7] S. Delaune and F. 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.

[8] W. Diffie and E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6) :644–654, 1976.

[9] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2) :198–208, 1983. Also STAN-CS-81-854, May 1981, Stanford University.

[10] M. Fox and D. Long. PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 2003.

[11] J. Hoffmann. The Deterministic Part of IPC-4 : An Overview. *Journal of Artificial Intelligence Research*, 24 :519 – 579, Octobre 2005.

[12] J. Hoffmann and B. Nebel. The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14 :253–302, 2001.

[13] S. Kambhampati. Planning graph as a (dynamic) CSP : Exploiting EBL, DDB and other CSP search techniques in graphplan. *Journal of Artificial Intelligence Research*, 12 :1–34, 2000.

[14] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.

[15] H. Kautz and B. Selman. Unifying SAT-Based and Graph-Based Planning. In Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.

[16] G. Lowe. An Attack on the Needham-Schroeder Public Key Authentication Protocol. *Information Processing Letters*, 56(3) :131–136, Novembre 1995.

[17] M. Needham and D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12) :993–999, 1978.

[18] L. Paulson. Proving properties of security protocols by induction. In *10th Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, 1997.