REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

N° d'ordre : .....

UNIVERSITE DE M'SILA

FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE

Département d'Informatique

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de MASTER

Domaine : Mathématiques et Informatique

Filière : Informatique

Spécialité : Systèmes d'Informations Avancés

Par : ZEROUGA Oussama

SUJET

Vérification automatique des Protocoles de sécurité

Soutenu publiquement le : / /2013 devant le jury composé de :

……	**Université de M'sila**	**Président**
Chikouche Noureddine	**Université de M'sila**	**Rapporteur**
……	**Université de M'sila**	**Examinateur**
……	**Université de M'sila**	**Examinateur**

Promotion : 2012 /2013

REMERCIEMENT

***M**erci au Noble « Allah» Dieu le tout puissant qui m'a donné le courage, l'intelligence, la force et la patience pour réaliser ce travail.*

***J**'exprime ma très sincère reconnaissance à mon encadreur monsieur **Chikouche Noureddine** Professeur à l'université de M'sila, pour m'avoir soutenue et encouragée, je le remercie vivement de sa lecture attentive et de ses remarques et ses conseils.*

***J**e tiens à remercie chaleureusement tous les membres de jury d'avoir accepté, et ainsi de me faire l'honneur d'assister à ma soutenance.*

***M**es remerciements s'adressent également à l'ensemble des enseignants et les responsable de département d'informatique de l'université de M'sila pour leurs efforts afin de nous assurer une meilleure formation.*

***J**e tiens à remercie mes parents pour leur amour et surtout pour m'avoir toujours encouragé et soutenu malgré tous mes défauts, j'espère qu'ils sont fière de moi, je remercie aussi mes sœurs pour leurs gentillesse et leurs encouragements.*

***J**'aimerais également remercie ma famille pour leur support, sans quoi je n'aurais pu trouver le confort et l'équilibre au long de ces années.*

J**e n'oublie pas de remercie mes amis avec lesquels j'ai passé des moments agréable **Zid elkhil said**, **Walid hamoudi**, **Belmokhtar elhosseyn**, **Ben yahia ali.

***E**nfin je remercie tous ceux qui m'ont aidé de près ou de loin à la réalisation de ce mémoire.*

O. Zerouga

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

Avec le développement des réseaux de communication comme internet, les réseaux mobile et cartes à puce…etc. le besoin d'assurer les communications a augmenté. C'est pourquoi des moyen cryptographique sont mis on ouvre afin de garantir des principes essentiels tels que l'authenticité d'entité, l'intégrité ou encore le secret, il est nécessaire donc de créer de nombreux protocoles cryptographique qui sont des règles d'échange entre les point du réseau permettant de sécuriser les donnes envoyés. Ils sont utilisés dans la téléphone mobile, le commerce électronique ou les chaînes télévision.

La conception de ces protocoles est basée sur la cryptographie : les algorithmes de chiffrement, les propriétés de sécurité …etc. Cependant la difficulté de la conception de ces protocoles tient au fait que les messages échange peuvent être écoutés, interceptés et modifies par un tiers. L'usage de méthodes cryptographique, ne suffit pas pour garantir le secret d'une information, tous les jours de nouvelles failles sont découvertes sur des protocoles répute sûrs lors de leur création, l'exemple le plus connu est celui du protocole de *Needhame Schroder Public Key* (NSPK) [6] dont on crie longtemps qu'il était sûr, a lorsqu'on a trouvé de nombreuse attaques appelée « main in the middle » alors il est nécessaire de vérifier ces protocoles avant de les mettre en service.

Le but de la vérification c'est la détection d'existence d'attaque et confirmer la validité du protocole. La plupart des techniques agréées sont basées sur les méthodes formelles.il y a plusieurs outils de vérification automatique de protocole tel que AVISPA (*Automated Validation of Internet Security Protocols and Applications*) [23] qui contient quatre outils d'analyse automatique et qui utilisent un langage de spécification des protocoles de sécurité HLPSL (*High-Level Protocol Specification Language)* [21]. Pour simplifier l'utilisation d'AVISPA, le logiciel SPAN [29] nous fournissait à la fois une interface graphique et la possibilité d'utiliser un langage simpliste que HLPSL.

Objectifs de travail

- Etude de l'état de l'art sur les protocoles de sécurité.
- Avoir une idée générale sur les mécanismes cryptographiques mis en œuvre afin de sécuriser les contactes.
- Présenter le langage de spécification HLPSL et comprendre le fonctionnement du vérifieur AVISPA.
- Spécification et vérification des protocoles de sécurité : Les protocoles des systèmes RIFD et les protocoles des systèmes carte à pus. Les propriétés vérifiées sont : la confidentialité et l'authentification.
- Faire une comparaison entre les différents protocoles d'authentification étudiés en termes de sécurité et la complexité des primitives cryptographiques.

Structure du Mémoire

Le mémoire est composé de quatre chapitres :

- Dans le premier chapitre « ***les protocoles des sécurités*** », on présente des principales concepts cryptographiques les plus importants. Puis on cite les différentes techniques et approches formelles de la vérification des protocoles de sécurité. Finalement, nous donnons brièvement les différents outils de vérification soit ceux consacrés à la recherche d'attaque et ceux consacrés à la preuve des protocoles.

- Dans le deuxième chapitre « ***Le langage de spécification HLPSL et la plateforme AVISPA***», nous allons présenter le langage de spécification des protocoles de sécurité HLPSL. Ensuite nous allons donner un bref aperçu sur la plate-forme AVISPA et nous expliquent les quatre outils de vérification : OFMC, CL-ATSE, SATMC et TA4SP. Enfin, nous allons montrer l'animateur de protocoles de sécurité SPAN.

- Dans Le troisième chapitre «***Contribution : Spécification et Vérification des Protocoles de Sécurité*** », nous allons spécifier et vérifié les protocoles de sécurité avec la plateforme AVISPA. Pour chaque protocole étudié, nous avons présenté la description formelle sous forme Alice-Bob, son spécification avec le langage HLPSL, et on a donné les résultats obtenus de la vérification.

- Dans le quatrième chapitre «***Etude comparative*** », nous allons donner les résultats des vérifications automatiques en précisant les types d'attaque pour chaque trace d'attaque détectée dans les protocoles étudiés. Ensuit nous allons faire une comparaison entre des travaux existants et les résultats d'AVISPA. Puis nous allons présenter la complexité du tag et carte à puce.

Ce mémoire termine par une conclusion générale qui résume notre travail et quelques perspectives.

Chapitre I

Les Protocoles de Sécurité

Sommaire

I.1. Introduction :

Les protocoles de sécurité spécifient les échanges de messages entre les participants dans un réseau. Ils se basent sur des fonctions cryptographiques afin d'assurer des propriétés de sécurité telles que la confidentialité des données ou l'authentification des entités …etc.

Dans ce chapitre, on présente des principales concepts cryptographiques les plus importants, tels que : le chiffrement, fonction de hachage, signature numérique, certificat,…etc. Puis on cite les différentes techniques et approches formelles de la vérification des protocoles de

sécurité. Finalement, nous donnons brièvement les différents outils de vérification soit ceux consacrés à la recherche d'attaque et ceux consacrés à la preuve des protocoles.

I.2. Concepts cryptographiques

La cryptologie est la science du secret, ne peut être vraiment considérée comme une science que depuis peu de temps. Cette science englobe la cryptanalyse qui est la science du décryptage sans connaissance de la clé, et la cryptographie qui est la science qui utilise les concepts mathématiques (tels que : congruence, calculs modulaires, …) pour le chiffrement et le déchiffrement de données. C'est aussi l'étude des techniques mathématiques afin de réaliser les objectifs de la sécurité informatique (confidentialité, intégrité et authenticité).

I.2.1. Propriétés de sécurité :

Dans cette section, on étudie les principales propriétés de sécurité qui sont généralement désirables lors des communications à travers les réseaux [1]:

- **Confidentialité** : Elle assure qu'une personne non autorisée ne puisse prendre connaissance d'une information sensible devant rester secrète.
- **Authentification :** consiste à prouver qu'un participant au protocole est bien celui qu'il prétend être et non un intrus se donnant une fausse identité. L'authentification peut être à sens unique (un seul des participants doit s'authentifier auprès de l'autre) ou mutuelle (les deux participants doivent s'authentifier l'un auprès de l'autre).
- **Non-répudiation :** empêche un parti de nier être la source d'un message. La non-répudiation se divise en deux : la non-répudiation de l'origine par laquelle un participant ne peut nier être à l'origine d'un message et la non-répudiation de la réception par laquelle un participant ne peut nier avoir reçu un certain message.
- **Intégrité :** est un mécanisme pour s'assurer que les données reçues n'ont pas été modifiés durant la transmission.

I.2.2. Le chiffrement : transformation d'un message lisible (en clair) en un message illisible (chiffré). L'opération inverse est le déchiffrement, qui nécessite pour cela la clé de déchiffrement.

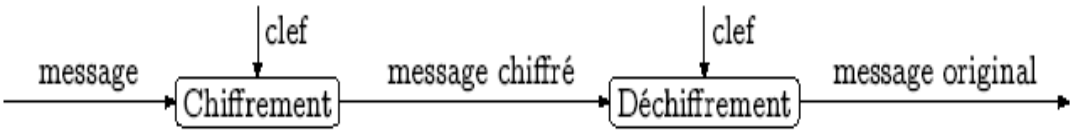

Figure I.1: Chiffrement et déchiffrement [2]

Il existe actuellement deux grands principes de chiffrement : le chiffrement symétrique basé sur l'utilisation d'une clé secrète et le chiffrement asymétrique qui utilise deus type de clés : une privée et l'autre publique.

I.2.2.1. Le chiffrement symétrique

Le chiffrement symétrique (aussi appelé chiffrement à clé privée ou chiffrement à clé secrète) est la plus ancien catégorie de chiffrement et consiste à utiliser la même clé pour le chiffrement et le déchiffrement.

Les principaux cryptosystèmes sont : DES (Data Encryption Standard), AES (Advanced Encryption Standard), IDEA (International Data Encryption Algorithm) et RC5 (Rivest Cipher).

Figure I.2: Chiffrement symétrique [3]

Avantages & Inconvénients:

Les avantages de chiffrement symétrique : la rapidité d'exécution (une seule clé utilisée) et la simplicité d'implémentation (gestion d'une seule clé). Pour les inconvénients, on cite : la complexité de fonctionnement : une obligation d'avoir le nombre de clés privées égal au nombre de destinataires et la sécurisation de la chaîne de transmission de la clé.

I.2.2.2. Le chiffrement asymétrique :

Le chiffrement asymétrique (aussi appelé chiffrement à clé publique**)** utilise une clef de chiffrement différente de celle de déchiffrement. La clef de chiffrement est souvent connue de tous les agents, elle est appelée clef publique et utiliser pour chiffrer un message. Cependant seuls les participants connaissant la clef de déchiffrement, appelée clef privée, peuvent déchiffrer les messages.

Les algorithmes de cryptographie asymétrique les plus utilisés sont : RSA (Rivest Shamir Adleman), ELGAMAL et DSA (Digital Signature Algorithm)

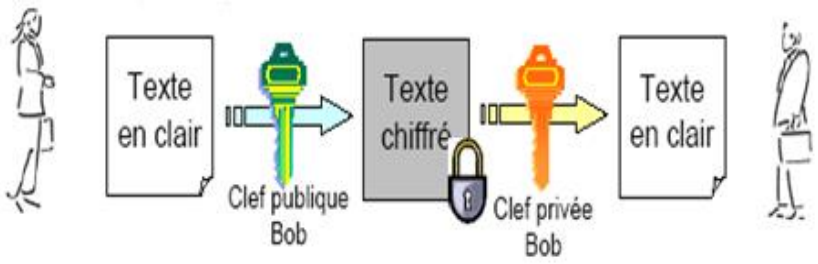

Figure I.3: Chiffrement asymétrique [3]

Avantages & Inconvénients :

Les avantages du chiffrement asymétrique sont : Impossibilité de substitution du destinataire : clé privée connue de lui seul, aucun transfert de clé privée : confidentialité assurée, et un seul couple de clés pour plusieurs expéditeurs. Concernant les inconvénients du chiffrement asymétrique, on cite le temps d'exécution est plus lent que le cryptage symétrique et sa complexité est très grand.

I.2.3. Fonction de hachage :

La fonction de hachage (aussi appelée fonction de hachage à sens unique) est très utilisé en cryptographie, principalement dans le but de réduire la taille des données à traiter par l'algorithme de chiffrement. En effet, la caractéristique principale d'une fonction de hachage est de produire un haché des données (ou l'empreinte), c'est-à-dire un condensé de ces données. Ce condensé est de taille fixe, dont la valeur diffère suivant la fonction utilisée [4].

Les fonctions de hachage les plus utilisées sont : MD5 (Message Digest 5)**,** SHA-1 (Secure Hash Algorithm)**,** SHA-256**.**

I.2.4. Signature numérique:

La signature numérique (parfois appelée digitale ou électronique) est un mécanisme permettant d'authentifier l'auteur d'un document électronique et de garantir son intégrité, par analogie avec la signature manuscrite d'un document papier. Un mécanisme de signature numérique doit présenter les propriétés suivantes:

- Il doit permettre au lecteur d'un document d'identifier la personne ou l'organisme qui a apposé sa signature.
- Il doit garantir que le document n'a pas été altéré entre l'instant où l'auteur la signé et le moment où le lecteur le consulte.

I.2.5. Certificat numérique:

Pour assurer l'intégrité des clés publiques, les clés publiques sont publiées avec un certificat. Un certificat (ou certificat de clés publiques) est une structure de données qui est numériquement signée par une autorité de certification (CA : Certification Authority). Une autorité en qui les utilisateurs peuvent faire confiance. Il contient une série de valeurs, comme le nom du certificat et son utilisation, des informations identifiant le propriétaire et la clé publique, la clé publique elle même, la date d'expiration et le nom de l'organisme de certificats. Le CA utilise sa clé privée pour signer le certificat et assure ainsi une sécurité supplémentaire.

Si le récepteur connaît la clé publique du CA, il peut vérifier que le certificat provient vraiment de l'autorité concernée et est assuré que le certificat contient donc des informations viables et une clé publique valide [5].

I.2.6. Générateur de nombres pseudo aléatoires

Générateur de nombres pseudo aléatoires est un composant logiciel, plus précisément un objet, qui produit une séquence de nombres, statistiquement uniformément distribués, liés à l'aspect aléatoire.

I.3. Protocole de sécurité :

Un protocole de sécurité ou protocole cryptographique est un ensemble de règles d'échange de messages permettant à des agents de communiquer de façon sécurisée. Il utilise des primitives cryptographiques (chiffrement, signature, ...etc) afin de garantir que ces messages sont échangés de façon sûre, même si le réseau lui-même n'est pas sûr.

I.3.1. Notation Alice-Bob

Pour simplifier la description des protocoles cryptographiques et ses attaques; on utilise une notation est largement utilisé, s'appelée les notations de Alice(A)-Bob(B) (voir Table I.1).

I.3.2. Catégories des protocoles

Les protocoles de sécurité sont classés en quatre grandes catégories : les protocoles d'authentification, les protocoles d'échange de clés, les protocoles de commerce électronique et les autres types de protocoles.

Notation	Signification
m	Message
A, B, R, T, C, S	Agent
Na, Nb, Nr, Nt	Nonce (nombre aléatoire «fraichement»)
h	fonction hachage
,	Concaténation
K	Clé symétrique partagé entre deux agents
K_A	Clé publique de A
{m}K	Le message m chiffré avec la clé K
ID	Identificateur partagé entre deux agents
A→B : m	L'envoi par A d'un message m à B
C(A) →B : m	L'envoi d'un message m par C se faisant passer pour A auprès de B

Table I.1. Notations

I.3.2.1. Protocoles d'authentification

Les protocoles d'authentification sont les protocoles de sécurité le plus largement utilisés. Un protocole d'authentification est une séquence d'échanges de message entre les participants, les objectifs de ses protocoles sont de fournir les divers services de sécurité à travers un système distribué. Ces objectifs sont les suivants: établir des clés de session entre les participants, la diffusion d'informations confidentielles, en garantissant l'authentification des participant, le secret, l'intégrité, et ainsi de suite, Elles impliquent l'échange d'informations entre les participants, nécessitant parfois la participation d'un serveur. Les informations qu'ils échangent typiquement sont protégées par des différents mécanismes cryptographiques, les chiffrements symétriques et asymétriques, les fonctions de hachage unilatérales et ainsi de suite. Un nombre considérable de protocoles d'authentification a été spécifié et a été exécuté.

Certains sont des protocoles d'authentification unilatérale ou mutuelle visant à convaincre l'autre individu qu'il communique avec la *bonne personne.* On cite quelque protocole les plus utilisé dans le mode de communication : EAP (*Extensible Authentication Protocol*), Kerberos, NSPK, PGP (*Pretty Good Privacy*),… etc.

On présente en détail le protocole d'authentification *Needham-Schroeder Public Key* (NSPK) [6], ce protocole décrit l'échange de messages entre deux participants, par exemple Alice et Bob, que nous dénoterons respectivement par A et B. Les deux participants veulent échanger en toute sécurité une clef privée N_B. Le protocole utilise un algorithme de chiffrement asymétrique. Nous dénotons par {m} k_A un message m chiffre par la clef publique d'Alice k_A et considérons que tous les participants connaissent toutes les clefs publiques. Le protocole est décrit comme suit :

1: A → B :{A,N_A}k_B

2: B → A :{N_A,N_B}k_A

3: A → B : {N_B}k_B

1. Alice envoie à Bob un message chiffre par la clef publique de Bob, K_B Il contient son identité A et un nombre aléatoire N_A t « fraichement » choisi par Alice, aussi appelé *nonce*.

2. Une fois ce message reçu, Bob choisit un nouveau nonce N_B et répond à Alice. Il envoie le nonce N_A précédemment reçu, ainsi que le nouveau nonce N_B dans un message chiffre par la clef publique d'Alice, k_A. N_B constitue la future clef échangée.

3. Alice confirme à Bob qu'elle a bien reçu le message, en lui renvoyant le nonce N_B chiffre par k_B, la clef publique de Bob.

I.3.2.2. Protocoles d'échange de clé

Ces protocoles permettent d'établir des secrets partagés par plusieurs personnes, comme par exemple une clé, ou un canal de communication privé : IKE (Internet Key Exchange) ,TLS (Transport Layer Security) , AKA (Authentication and Key Agreement) , EKE (Encrypted Key Exchange) . Un principe couramment utilisé pour construire une clé fraîche et symétrique est celui introduit par Diffie-Hellman [7]. Ce protocole est un protocole d'échange de clés entre deux agents A et B. Une variante simple de ce protocole, en trois étapes est présentée ci-dessous.

A -> B : g^{Na}

B -> A : g^{Nb} , A et B calculent la clé K = $(G^{Na})^{Nb} = (G^{Nb})^{Na}$

A -> B : {*Nsec*}_*K*

A l'étape 1, A génère le nonce (un grand nombre aléatoire) Na et calcule l'exponentiation modulaire de G par Na, G^{Na}, où G est un nombre public. A envoie ensuite le message G^{Na} à l'agent B.

A l'étape 2, l'agent B génère lui aussi un nombre Nb et calcule G^{Nb} puis K = $(G^{Na})^{Nb}$. Il envoie le premier à A et conserve le second comme clé partagée avec A. Dés que A reçoit le message G^{Nb} , il est capable de calculer $(G^{Nb})^{Na}$ et il a donc lui aussi la clé partagée. En effet, d'après les propriétés de l'exponentiation modulaire K = $(G^{Na})^{Nb} = (G^{Nb})^{Na}$.

Et finalement, à l'étape 3, le message {Nsecret}K est envoyé par A à B avec Nsecret une donnée qui doit rester secrète entre A et B. Notons que { }K montre bien l'utilisation d'un cryptage symétrique de clé K.

I.3.2.3. Protocoles de commerce électronique

Les protocoles de commerce électronique (appelés protocoles e-commerce). Un protocole e-commerce est combiné avec certains protocoles de sous protocoles, comme un protocole de paiement électronique, un protocole de gestion électronique, et ainsi de suite. En raison de l'importance de protocoles e-commerce, il nécessite presque tous les objectifs de sécurité tels que l'authentification, secret, non-répudiation, intégrité, anonymat. Un exemple de protocole de commerce électronique est SET (*Secure Electronic Transaction*) pour sécuriser les transaction bancaires via l'internet sans modifier les circuits bancaire existants pour l'autorisation tout en pouvant se connecter à distance, et le protocole SSL (*Secure Socket Layer*) permettant des échanges sécuritaires sur l'internet.

I.3.2.4. Protocoles de groupes

En général, un protocole est défini pour un nombre fixe d'agents. Ensuite, plusieurs sessions sont jouées, mais le nombre d'agents jouant les sessions est constant. Il existe une classe de protocoles telle que le protocole est défini pour un nombre quelconque d'agents. Ces protocoles sont qualifiés de protocoles de groupe. Certains protocoles permettent, par exemple, l'établissement d'une clé partagée entre *n* individus : IKA Cliques-I et Cliques-II [8]. Ces deux protocoles mettent en jeu la méthode de Diffie-Hellman que nous avons présentée précédemment pour l'établissement d'une clé commune à tous les agents.

I.3.3. Principales attaques

I.3.3.1. L'attaque « man in the middle » : littéralement « *attaque de l'homme au milieu* » ou « *attaques de l'intercepteur* », parfois notée *MITM*, est un scénario d'attaque dans lequel un intrus écoute une communication entre deux agents et falsifie les échanges afin de se faire passer pour l'une des parties.

I.3.3.2. Les attaques par « rejeu » : les attaques par rejeu (en anglais « *replay attaque* ») sont des attaques de type « man-in-the-middle ». L'attaque par rejeu consiste à rejouer des messages envoyés précédemment dans la même session du protocole ou dans d'autres sessions de ce même protocole. C'est-à-dire les retransmettre tels quel (sans aucun déchiffrement) au serveur destinataire.

L'intrus peut bénéficier des droits de l'utilisateur. Par exemple le client transmet un nom d'utilisateur et un mot de passe chiffrés à un serveur afin de s'authentifier. Si un intrus

intercepte la communication et rejoue la séquence, il obtiendra alors les mêmes droits que l'utilisateur.

I.3.3.3. Usurpation d'identité: dans ce type d'attaque, l'intrus utilise une fausse identité pour abuser un utilisateur ou usurper son identité dans le but d'obtenir des informations sensibles ou modifier des données.

I.4. Modélisation des protocoles de sécurité :

Depuis les années 80, deux approches ont été développées pour modéliser des protocoles de sécurité. L'une de ces approches repose sur un modèle calculatoire et l'autre repose sur une modélisation symbolique des exécutions du protocole, où les primitives cryptographiques sont traitées comme de boites noires**.** Dans notre travail, nous intéressons par les modèles symboliques.

I.4.1. Le modèle calculatoire

Le modèle calculatoire (appelé aussi, probabiliste) prend en compte les probabilités et la complexité algorithmique .Cette approche permet de définir une notion très forte de sécurité, garantissant contre n'import quelles attaques probabilistes et polynomiales. Le point faible consiste en le fait que les preuves sont souvent pénibles et se prêtent mal à une automatisation [9].

I.4.2. Le modèle symbolique

Le modèle symbolique (appelé aussi, formel). Dans ce modèle les fonctions cryptographiques sont considérées comme des boîtes noires, les messages sont des termes sur ces fonctions cryptographiques et l'attaquant est restreint à calculer à l'aide de ces fonctions. Ce modelé basé sur l'hypothèse de la cryptographie parfaite. Ainsi, pour le chiffrement, on suppose qu'on ne peut déchiffrer que si on a la clé. Le point fort d'un modèle symbolique est le fait qu'il permet d'obtenir des preuves simplifiées et souvent automatisées [10].

Les premiers modèles symboliques sont généralement attribues à Needham et Schroeder [6] et à Dolev et Yao. Des différents modèles symboliques ont été proposés, nous mentionnons les principaux modèles dans la section suivante.

I.5. Les principaux modèles symboliques :

On présente dans cette section les techniques utilisées pour la vérification des protocoles cryptographiques dans les modèles symboliques.

I.5.1. Modèle de Dolev-Yao

En 1983, les scientifiques Danny Dolev et Andrew Yao [11] ont proposés un modèle de l'intrus. Il représente les capacités que possède un attaquant pour apprendre de l'information. Dans ce modèle l'intrus est capable de lire, supprimer et modifier tous les messages échangés entre les participants honnêtes dans le réseau, outre l'intrus peut générer de nouveaux messages en se basant seulement sur les messages qu'il possède et les envoyer à tous les agents et jouent également le rôle d'un agent honnête, prendre une part de la session ou sessions multiple pair de l'exécution du protocole. Nous détaillons les hypothèses faites par le modèle de Dolev-Yao dans la section 7.

I.5.2. Model checking

Le Model-checking [12] est une technique automatique de vérification formelle. Il s'agit de construire un modèle du système à vérifier avec un formalisme donnée, généralement représenté par un automate à états fini. Ce modèle est vérifié s'il satisfait un ensemble de propriétés (une spécification) souvent formulées en logique temporelle. En autres, le système est décrit par un modèle sémantique et les propriétés sont décrites par des formules logiques.

I.5.3. Règle de réécriture

Plusieurs modèle représentent les réglé de protocole et le pouvoir de l'intrus par des règles de réécriture sur des termes. La principale limitation de cette modélisation est de ne pas toujours exprimer les nonces de manière fidèle et de permettre la réparation de règles déjà jouées.

I.6. Les approche de vérification

Dans le domaine de vérification de la sécurité des protocoles cryptographiques, les recherches s'articulent sur deux axes : d'une part la recherche d'attaques à l'aide d'outils de vérification, d'autre part la preuve des propriétés de sécurité par abstraction.

I.6.1. Recherche d'attaques

Le premier axe de vérification automatique de protocoles cryptographiques a été des découvertes d'attaques à l'aide d'outils d'analyse de protocoles [13] comme l'attaque trouvée par G. Lowe [14] sur le protocole de Needham-Schroeder à clefs publiques. Dans cette approche le nombre de sessions et des participants est borné.

I.6.2. Preuve d'un protocole :

Le point faible des méthodes basées sur la recherche d'une attaque est que le nombre de sessions et les tailles des messages sont bornés. Ainsi, s'il n'y a pas détection d'une attaque dans un protocole avec une ou deux sessions, il ne peut pas dire que ce protocole est sûr, parce que c'est possible de trouver des attaques en utilisant quatre sessions et ainsi de suite. Donc dans cette approche l'utilisation du nombre de sessions n'est pas bornée.

I.7. Principales hypothèses de la modélisation

I.7.1. Canaux de communication

On peut distinguer deux types de canaux de communication. Les canaux de communication publics et les canaux de communication privés [15] :

- ***Publics.***

  Un canal de communication est dit *public,* si tous les messages émis sur ce canal peuvent être vus par tous les participants, qu'ils soient honnêtes ou pas. Les messages échanges sur ces canaux sont donc connus de tous, en particulier de l'intrus qui peut s'en servir pour construire une attaque.

- ***Privés.***

  Les canaux de communications dits *prives* sont des canaux de communications définis entre certains participants honnêtes, et seuls ces participants peuvent recevoir et envoyer des messages sur ces canaux. Par conséquent, un intrus ne peut donc pas écouter les messages qui circulent sur ce genre de canaux.

I.7.2. Nonces.

Les protocoles cryptographiques utilisent des nombres aléatoires « frais » dans leur spécification. Cela signifie que le générateur de nombres aléatoires garantit une probabilité faible que deux nombres aléatoires soient identiques. Ces nombres aléatoires sont appelés *nonces*. Les nonces sont utilisés pour éviter les attaques de type « attaque par rejeu ».

I.7.3. Les participants :

Chaque participant peut effectuer, simultanément, avec différents participants, plusieurs sessions du protocole. Le serveur, le client, la carte à puce, le programme et le lecteur sont des exemples de participants. Les participants sont de deux types : honnêtes ou intrus.

I.7.3.1. Participants honnêtes :

Participants honnêtes**.** C'est un agent (une personne ou un programme) légitime du protocole, ayant un comportement bien précis, et prévue à l'avance.

I.7.3.2. Intrus.

Un *intrus* ou *attaquant* est un participant qui ne suit pas exactement le déroulement du protocole. Il espionne la communication qui circule sur les canaux publics, joue plusieurs sessions de protocoles avec des participants, en se faisant passer pour un agent honnête et ainsi effectue des actions non prévues par la spécification du protocole, afin de découvrir des informations supposées rester secrètes [15]. Nous distinguons deux types d'intrus :

- *Intrus passif.*

  L'adversaire ou l'intrus se contente d'écouter, d'enregistrer les messages circulant sur le réseau pour les analyser afin de déduire un secret. Ce genre d'intrus menace la confidentialité des données.

- *Intrus actif.*

  L'adversaire ou l'intrus a les capacités de l'intrus passif et peut aussi supprimer, ajouter, intercepter ou changer la transmission sur le canal. Cet intrus modifie les messages passés dans le réseau. Ce genre d'intrus menace n'est pas attaque sur la confidentialité des données seulement, mais aussi l'authentification et l'intégrité des données.

I.7.4. Hypothèse de chiffrement parfait.

Il considère les fonctions mathématiques de chiffrement dans l'hypothèse de chiffrement parfait, comme des « boites noires ». L'hypothèse de chiffrement parfait est : *Un message crypté peut être décrypté uniquement avec la clé inverse.* On peut appliquer cette hypothèse sur le chiffrement symétrique, chiffrement asymétrique, les fonctions de hachage et les nonces. L'inconvénient de l'hypothèse de chiffrement parfait est que les primitives algébriques tels que *ou exclusif* et *exponentiation modulaire* ne sont pas pris en compte dans l'analyse des protocoles.

I.8. Outils pour vérifier les protocoles

On peut distinguer deux familles d'outils : ceux consacrés à la recherche d'attaque et qui ne considèrent qu'un nombre borné de sessions et ceux consacrés à la preuve des protocoles, pour un nombre non borné de sessions. Nous allons présenter certains de ces outils en quelques mots.

I.8.1. Outils consacrés à la recherche d'attaques

Casper/FDR. Cet outil a été développé par G. Lowe [16] pour vérifier les protocoles cryptographiques. Le nombre de sessions et de participants est borné, la taille des messages est également bornée. De plus, les messages sont typés. La vérification du protocole correspond alors au model-checking.

C'est en particulier le model-checking avec FDR du protocole de Needham-Schroeder qui a permis à G. Lowe de trouver son attaque sur ce protocole [14] puis d'en proposer une nouvelle version.

Casrul. Cet outil, développé par *Jacquemard et al.* [17], permet également de chercher des attaques pour un nombre de sessions et de participants borné. C'est l'un des seuls outils qui permet de traiter les messages dans toute leur généralité : les messages n'ont pas besoin d'être typés. Les protocoles sont décrits sous forme de règles de réécriture.

Aviss. L'Aviss [18] est un logiciel assez récent qui traduit les protocoles en un langage commun à trois logiciels de vérification (OFMC, Atse, SATMC). L'utilisation commune de ces trois logiciels permet de retrouver de très nombreuses attaques déjà connues et a également permis de découvrir une attaque nouvelle sur le protocole de Denning-Sacco.

I.8.2. Outils consacrés à la preuve

Cryptic Cet outil [19] permet de prouver des propriétés d'authentification. Chaque protocole est annoté en fonction de la propriété voulue et des types sont associés aux nonces, aux clefs et aux messages. Le but de l'outil est alors de vérifier que le protocole est bien typé, ce qui est une condition suffisante pour garantir la propriété demandée.

***Proverif*.** Est un outil de vérification de protocole de sécurité, développé par Bruno Blanchet [20]. Cet outil est utilisé pour prouver les propriétés de sécurité : secret, et authentification de différents protocoles. Il peut gérer un nombre illimité de sessions du protocole et d'un espace

message sans bornes. Un avantage d'utiliser ProVerif comme outil de certification est qu'il est un des modèles à un attaquant qui est conforme au modèle Dolev Yao automatiquement. Nous n'avons pas besoin de modéliser explicitement l'attaquant.

Timbuk. Cet outil permet de calculer une approximation d'un ensemble de messages reconnaissable par un automate d'arbre, modulo des règles de réécriture. Cet outil a permis de vérifier une propriété autre que le secret.

I.9. Conclusion

Dans ce chapitre, Nous avons présenté les notions de base de chiffrement et des protocoles de sécurité. On a expliqué les différentes approches de vérification de sécurité des protocoles cryptographiques. On a détaillé les méthodes formelles et des outils automatiques de la vérification. Dans le chapitre suivant, nous allons présenter les outils AVISPA, qui sont utilisés pour valider la sécurité des protocoles d'authentification étudiés.

Chapitre II
Le langage de spécification HLPSL
et la plateforme AVISPA

Sommaire

II.1. Introduction

Pour la vérification des protocoles de sécurité, il existe plusieurs outils d'analyses mais les outils AVISPA (*Automated Validation of Internet Security Protocols and Applications*) est la plus connue. Dans ce chapitre nous allons présenter le langage de spécification des protocoles des sécurités HLPSL (*High Level Protocol Specification Language*) et nous allons donner un exemple de spécification de protocole fil-rouge. Ensuite nous allons donner un bref aperçu sur la plateforme AVISPA [23] et nous expliquent les quatre outils de vérification qui ont été intégrés dans la plate-forme : OFMC, CL-ATSE, SATMC et TA4SP. Enfin, nous allons montrer l'animateur de protocoles de sécurité SPAN (*Security Protocol ANimator*) [29].

II.2. Le langage de spécification HLPSL :

Le HLPSL (High Level Protocol Specification Language) [21,22] est un langage expressif de la communication et de la modélisation des protocoles de sécurité pour l'outil AVISPA. Il permet ainsi la représentation d'un protocole de sécurité par un système d'états/transitions sur lequel la vérification des propriétés de sûreté exprimées en logique temporelle linéaire(LTL) sera effectuée.

C'est un langage de spécification modulaire basé sur la notion de rôles (participants) et de rôles composés (sessions, instances). Un rôle simple sert à décrire les actions d'un agent lors de l'exécution du protocole. Un rôle composé permet d'instancier plusieurs rôles simples afin de modéliser l'exécution du protocole entier. En plus de la notion de rôle fournie par HLPSL, ce langage possède les caractéristiques suivantes :

- primitives cryptographiques variés (clés symétriques, clés publiques et privées, fonctions de hachage,…);
- information typée (ou non), avec des types simples ou composés ;
- propriétés algébriques supportées (concaténation, OU exclusif, exponentiation…);
- canaux pour les échanges de messages;
- flux de contrôle assurant les transitions valides;
- propriétés de sécurité à vérifier : confidentialité, authentification, intégrité.

II.3. Structure d'une spécification HLPSL

Une spécification HLPSL est composée de trois parties : Une liste de définition des rôles, une liste des objectifs ou des propriétés de sécurité à vérifier, et enfin, l'instanciation du rôle principal (généralement sans arguments).

Dans notre exemple de spécification de protocole *fil-rouge* par le langage HLPSL. La description de ce protocole par notation Alice-Bob est comme ci-dessous:

```
1. A -> B : Kab, {M}Kab
```

II.3.1. Définition des rôles

Les rôles peuvent être définis comme des processus indépendants qui ont des noms, reçoivent des informations en paramètre et contiennent des déclarations locales. On distingue deux catégories de rôles :

II.3.1.1. Les rôles basiques

Cette catégorie décrite la connaissance initiale et le comportement de chaque entité (agent) intervenant dans le protocole spécifié. Ci-dessous, un exemple simplifié d'une déclaration de rôle pour l'agent ''Alice'' dans le protocole *fil-rouge*. Cet exemple est donné afin de mieux comprendre cette notion de rôle.

```
role alice (A,B : agent,
            Kab : symmetric_key,
            SND,RCV: channel(dy))
played_by Adef=

local State : nat,
          M : text
const id1 : protocol_id
init State := 0
transition
  1. State = 0  /\ RCV(start) =|>
     State':= 1 /\ M' := new()
                /\ SND(Kab.{M'}_Kab)
                /\ secret(M',id1,{A,B})
end role
```

La connaissance initiale liée à chaque rôle (dans notre exemple, concerne alice) est exprimée par une liste de paramètres qui sont : la clé publique Kab, les différents agents connus par Alice, un ensemble de canaux SND, RCV qui seront utilisés pour la communication. Dans la déclaration d'un rôle, on peut trouver une clause optionnelle ''played_by'' pour spécifier quel agent joue le rôle considéré. Dans le rôle Alice l'acteur principal est l'agent A (played_by A).

Nous pouvons trouver aussi une section qui contient une liste de variables locales et leurs types, qui peuvent être initialisées dans la section 'init'. L'extrait de spécification exprime la déclaration de deux variables *locales State* et *M*, précisant que la variable *State* est initialement instanciée à la valeur 0.

```
local State : nat,
      M: text

 init State := 0
```

Une liste de transitions est définie comme dans la toute dernière partie du rôle. Cette dernière va définir le comportement de l'agent à travers de des transitions qu'il peut faire suivant une certaine condition pour changer d'état.

Concernant notre exemple (protocole *fil-rouge*), si l'agent A se trouve à l'état (State=0) et que sur le canal Rcv il peut lire le message ''*start*'' alors la transition est déclenchée et change d'état vers State' = 1. (State' signifie que l'ancienne valeur stockée dans State sera écrasée par la nouvelle valeur qui est "1"). La variable *M*est instanciée par une valeur aléatoire (nonce) grâce à l'instruction *M' :=new().*Cette nouvelle valeur est d'abord chiffrée par Kab, puis concaténée à Kab et les envoyées sur le canal SND.

```
1. State = 0    /\ RCV(start) =|>
   State':= 1   /\ M' := new()
                /\ SND(Kab.{M'}_Kab)
```

II.3.1.2. Les rôles de composition

Chaque spécification doit avoir un rôle spécial qui peut s'appeler rôle de session et qui va représenter la structure des sessions pendant l'exécution du protocole. Dans l'exemple ci-dessous est représentée une session du protocole *fil-rouge* entre deux agents Alice et Bob. L'opérateur / \ indique que ces rôles devraient exécuter en parallèle. Les variables SA et SB sont des canaux d'émission des messages des agents A et B respectivement. Au contraire RA et RB sont des canaux de réception des messages des agents A et B respectivement.

```
role session(A, B: agent, Kab : symmetric_key)def=

local SA, RA, SB, RB: channel (dy)
composition
      alice(A,B,Kab,SA,RA)
   /\ bob  (A,B,Kab,SB,RB)

end role
```

Il existe un autre rôle de composition appelé environnement ou rôle principal. Ce rôle ne possède aucun paramètre et sert à définir l'état initial du système en précisant d'un côté, la connaissance initiale de l'intrus par la clause *intruder knowledge* et d'un autre côté, un nombre fini d'instances du rôle session.

Dans la spécification d'une session de protocole *fil-rouge* Le rôle ci-dessous exprime une session qui se déroule entre deux agents a et b en utilisant la clé kab. Les données a, b et kab sont des constantes associées à un type dans la section *const*. L'intrus connaît initialement les agents a et b.

```
role environment() def=
  const a, b : agent,
        kab  : symmetric_key

  intruder_knowledge = {a, b}

    composition
        session(a,b,kab)

end role
```

II.3.2. Les objectifs de sécurité

Il existe différents signaux ou évènement. L'évènement *secret* permet de déterminer les propriétés de confidentialité alors que les deux autres évènements (*Witness*, *Request*) permettent la définition de deux types de propriété d'authentification : authentification faible et authentification forte. Ces signaux permettant d'exprimer les propriétés de sécurit [21] :

- **Secret (E, id, S) :** déclare que l'information E est un secret partagé par un ensemble S d'agent.
- **Witness (A, B, id, E) :** pour la propriété d'authentification faible de l'agent A auprès de B grâce à la donnée E. Cet objectif sera identifié par la constante id dans la section réservée à la déclaration des propriétés de sécurité.
- **Request (B, A, id, E) :** pour l'authentification forte entre A et B. Elle déclare que B demande une vérification de la valeur E. La fonction de id est la même que pour witness.

La déclaration des objectifs ou des propriétés à vérifier se fait dans une section à part. Chaque propriété est identifiée par une constante qui réfère le prédicat défini (secret, witness, request) pour une transition donnée.

L'exemple (protocole *fil-rouge*) permet de déclarer la nouvelle valeur de M après exécution de la transition comme étant un secret partagé entre A et B. L'instruction M' := new () permet de générer aléatoirement une nouvelle valeur pour M.

```
State=0     ∧ RCV(start) =|>
State′:=1 ∧ M′ := new()
            ∧ SND(Kab.{M′} Kab)
            ∧ secret(M′, id_sec,{A,B})
```

En HLPSL, nous pouvons exprimer les objectifs à vérifier à l'aide des macros suivantes dans une section HLPSL réservée et nommée goal :

```
Secrecy_ofsec_id
Authentication_onauth_id
weakauthentication_onwauth_id
```

Les données *sec_id*, *auth_id* et *wauth_id* sont les identifiants attribués aux signaux. Remarquons que pour exprimer une propriété d'authentification faible ou forte, nous devons spécifier deux signaux *witness*et *request*ayant le même identifiant.

II.3.3. Instanciation d'un rôle

La création d'une instance d'un rôle est comme l'appel d'une procédure, en donnant une valeur pour chaque paramètre déclaré dans le rôle, et bien sûr le nombre d'arguments ainsi que leurs types doit être les mêmes que ceux des paramètres du rôle. Pour le rôle principal, il suffit d'invoquer son nom (usuellement sans paramètre comme suit : *environment()*).

II.4. La plate-forme AVISPA

II.4.1. Description et Architecteur

Le projet AVISPA [23] a développé une plate-forme de vérification automatique de protocoles sécurités, appelée également AVISPA. Elle est accessible sur le réseau à l'adresse www.avispa-project.org Elle met à disposition le langage HLPSL qui est utilisé pour spécifier à la fois les protocoles et les propriétés à vérifier. Dans ce langage, deux symboles sont définies pour le chiffrement symétrique et asymétrique. Il permet aussi la déclaration des fonctions de hachage. Il existe également un opérateur de concaténation. Par contre, il n'y pas d'opérateur pour la signature. Le langage HLPSL permet de décrire l'échange des messages entre les participants pendant l'exécution d'un protocole. La plate-forme permet de vérifier trois propriétés : le secret, l'authentification faible et l'authentification forte. Cette dernière propriété est une combinaison entre l'authentification faible et la protection contre le rejeu. Après avoir spécifié le protocole dans HLPSL, celui-ci est traduit dans le langage IF (Intermediate Format) par le compilateur ou traducteur HLPSL2IF.

Le code IF peut ensuite être utilisé par des outils d'analyse différents qui effectuent la vérification. Quatre outils de vérification ont été intégrés dans la plate-forme AVISPA : OFMC, ATSE, SATMC et TA4SP.L'architecture de la plateforme AVISPA est montrée dans la Fig.II.2.

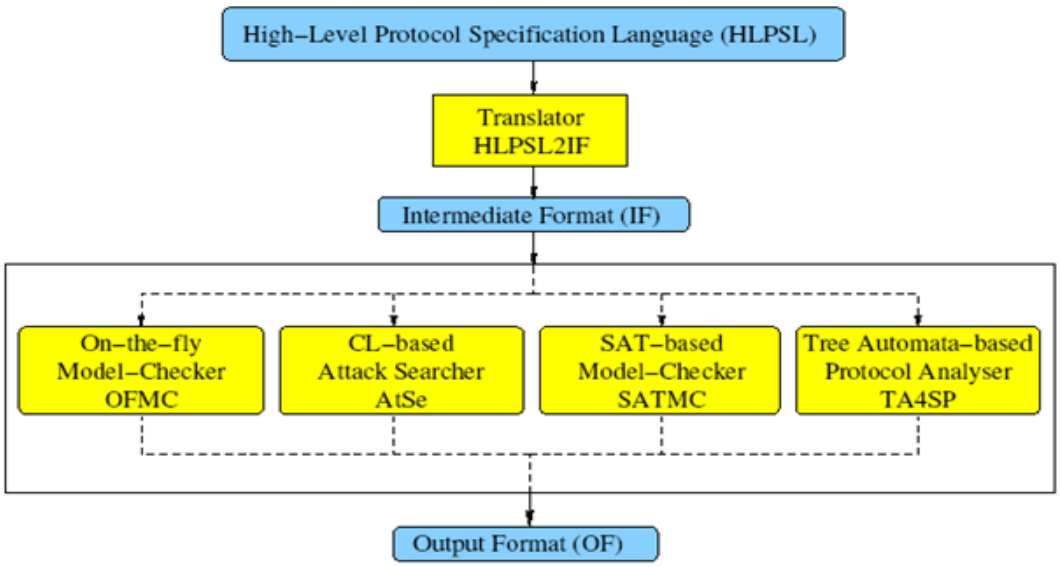

Figure II.1 : l'architecture de la plateforme AVISPA [24]

Une interface graphique (FigureII.3) est également disponible selon deux modes :

- Mode *basique* simplifiant les actions de l'utilisateur, et lançant les quatre outils en parallèle, le résultat de chacun étant ensuite visible ;
- M*ode expert* permettant de choisir l'outil d'analyse et de régler quelques paramètres d'analyse.

Lorsqu'une attaque est trouvée, un diagramme de séquence permet de visualiser les messages échanges qui ont mené à cette attaque.

II.4.2. Le traducteur HLPSL2IF

Le traducteur HLPSL2IF traduit automatiquement une spécification de HLPSL vers une spécification IF. Le traducteur fonctionne comme suit. Premièrement, il analyse la spécification de HLPSL, vérifiant qu'un nombre de conditions sont rempliés (par exemple que toutes les variables utilisées sont déclarées). Ensuite, il aplatit la structure hiérarchique des descriptions de rôles dans HLPSL et les traduit dans les règles d'étape du IF, décrivant les transitions agents honnêtes peuvent exécuter.

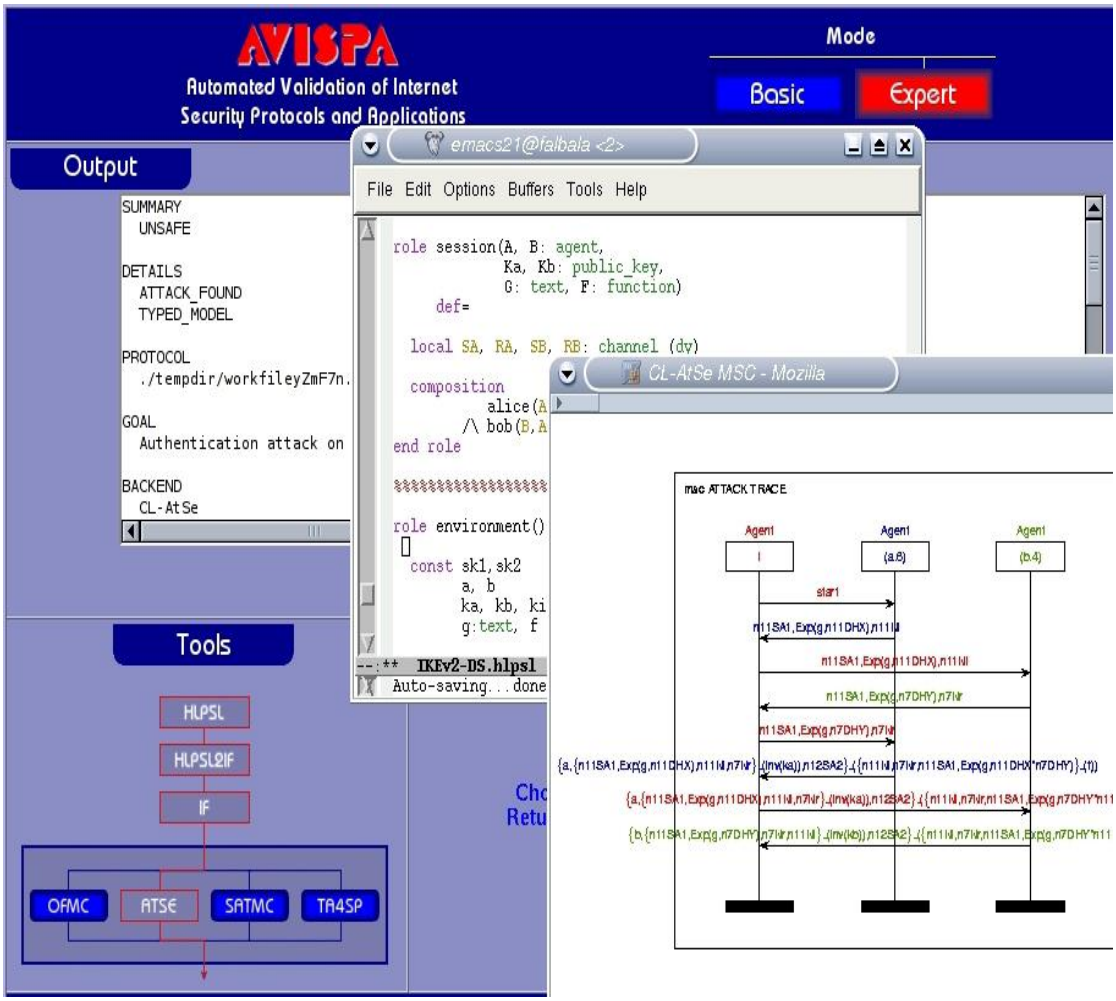

Figure II.2 : Interface graphique du logiciel AVISPA [25]

L'état initial du IF est calculé de l'instanciation donnée dans le fichier HLPSL (déclarant quels agents sont de jouer quels rôles du protocole avec que). Enfin, les objectifs sont calculés comme un codage basé sur l'état des propriétés données dans le fichier HLPSL.

II.4.3. Langage format intermédiaire IF

Le IF (**I**ntermediate **F**ormat) [26] est un langage de bas niveau, simple mais expressif pour spécifier des protocoles de sécurité et de leurs propriétés. Les spécifications IF peuvent être générées automatiquement par le traducteur HLPSL2IF de spécifications écrites dans le langage de haut niveau HLPSL. La première version de IF a été développé pendant le projet AVISS et cette nouvelle version est une refonte complète de la conception de la langage, basée sur l'expérience que les partenaires de projet ont fait avec une variété de problèmes

d'analyse de protocole, afin d'être en mesure d'analyser les protocoles de sécurité Internet et les applications que nous allons considérerons dans le projet AVISPA.

II.4.4. Outils AVISPA

Les outils (ou back-ends) intégrés à AVISPA permettent de vérifier les propriétés de sécurité, selon les besoins. Ils prennent en entrée le format intermédiaire IF.

II.4.4.1. Constraint logic-based Attack Searcher (CL-AtSe)

C**'**est un outil basé sur des techniques de résolution de contraintes. Il permet de faire une traduction d'une spécification d'un protocole de sécurité sous forme de relations de transition au format IF, vers un ensemble de contraintes qui peuvent être utilisées pour trouver des attaques sur le protocole en question [24]. Les deux méthodes de la traduction et la vérification sont totalement automatiques et prises en charge par l'outil CL-AtSe sans intervention d'outils externes. Les possibilités de CL–AtSe ont été étendues lors du projet AVISPA pour supporter de vérification des protocoles intégrant les opérateurs algébriques (**Xor** ou **Exp**).

II.4.4.2. On-the-fly Model-Checker (OFMC)

Pour but d'analyser la sécurité des protocoles cryptographiques. Il effectue une vérification bornée en explorant le système de transition décrit par une spécification IF. OFMC implémente des techniques symboliques correctes et également complètes [27]. Il supporte la spécification des opérateurs à propriétés algébriques tels que le OU exclusif ou encore l'Exponentielle.

II.4.4.3. SAT-based Model-Checker (SATMC)

Cet outil a été développé au laboratoire DIST à Gènes (Italie) [27]. Il construit une formule propositionnelle codant pour un déroulement borné de la relation de transition spécifiée par IF, l'état initial et l'ensemble des états représentant une violation des propriétés de sécurité [28].La formule propositionnelle est ensuite à résoudre à un solveur SAT et tout modèle satisfaisant cette formule est retourné sous forme d'attaque.

II.4.4.4. Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP):

La particularité de cet outil de vérification est qu'à partir d'un état initial il fait soit une sur-approximation (si le protocole est sûr pour un nombre illimité de sessions) ou une sous-approximation (si un protocole est erroné) des connaissances de l'intrus en utilisant des automates d'arbres [24]. Cette méthode permet de savoir si un certain état est accessible ou non et que l'intrus peut savoir certaines connaissance ou non et ainsi de conclure l'absence d'attaque sur le secret pour des scénarios exécutés un nombre indéterminé de fois.

Les trois premiers outils cherchent des attaques, alors que le dernier essaie de démontrer la validation des propriétés, tâche beaucoup plus difficile et ne pouvant être appliquée que pour des propriétés relativement restreintes.

II.5. Utilisation de la plateforme AVISPA

L'interaction avec le type d'outil AVISPA est la suivante :

1. On commence par la spécification du protocole à tester grâce au langage HLPSL, ainsi que les propriétés à vérifier.
2. On lance AVISPA à l'aide d'une invite de commandes toute en précisant l'analyseur (back-end) qu'on va utiliser.
3. Ensuite AVISPA, après analyse, il déclare que soit le protocole est sûr, ou bien le protocole présente des failles.

II.6. Le résultat de vérification

Quand l'analyse d'un protocole est réussie, en trouvant ou pas des attaque possible, la sortie de AVISPA décrit précisément le résultat. Le premier résultat obtenu est de résumé les diagnostics de chaque outil de la plateforme AVISPA. Il existe trois types de messages :

- Le message « *UNSAFE* » signifie que le protocole n'est pas sûr et l'outil présente une trace d'attaque,
- Le message « *SAFE*» signifie que le protocole est sûr, et il n'y a pas détection des attaques,
- Le message « *INCONCLUSIVE* » signifie que l'outil n'a pas aboutit a un résultat.

Dans l'exemple de protocole *fil-rouge*, le résultat de la vérification de ce protocole avec la plateforme AVISPA est :

```
AVISPA Tool Summary
OFMC    : UNSAFE
CL-AtSe : UNSAFE
SATMC   : UNSAFE
TA4SP   : INCONCLUSIVE
Refer to individual tools output for details
```

Apres avoir affiché un diagnostic général, une deuxième phase consiste à appuyer sur le bouton comportant le nom d'outil afin de visualisé la trace d'attaque s'il existe en détail, ainsi que les informations spécifiques de la vérification tels que :

- La première section *SUMMARY* indique si le protocole est sécurisé ou non, ou si l'analyse n'a pas été concluante,
- La deuxième section *DETAILS* décrit sous quelles conditions le protocole est déclaré sécuriser ou non, sous quelles conditions une attaque est trouvée, et finalement pourquoi l'analyse n'a pas été concluante,
- La section *PROTOCOL* rappelle le nom du protocole analysé,
- La section *GOAL* présente le but de l'analyse, comme par exemple la confidentialité de la clé de chiffrement des données,
- La section *BACKEND* désigne le traducteur des spécifications HLPSL, utilisé lors de l'analyse.
- La section *STATISTIC*S le temps de recherche, le temps d'analyse grammaticale, le nombre de nœuds visités et la profondeur.

Le résultat obtenu peut prendre trois formes : la forme texte, la forme MSC (Message Sequence Chart), ou enregistrer dans un fichier postscript. Et voilà un exemple de résultat de vérification du protocole KN avec l'outil OFMC sous forme texte :

```
% OFMC
% Version of 2006/02/13
SUMMARY
UNSAFE
DETAILS
  ATTACK_FOUND
PROTOCOL
  /home/avispa/web-interface-
computation/./tempdir/workfileThH3Ff.if
GOAL
authentication_on_aut_server
BACKEND
  OFMC
COMMENTS
STATISTICS
parseTime: 0.00s
searchTime: 0.03s
visitedNodes: 3 nodes
  depth: 2 plies
ATTACK TRACE
i -> (c,3): start
(c,3) -> i: idi.h(idi.xs) XOR Nc(1)
i -> (s1,3):idi.h(idi.xs) XOR Nc(1)
(s1,3) -> i:h(h(idi.xs) XOR
Nc(1)).Nc(1).h(idi.xs) XOR Ns(2)
i -> (c,3): h(h(idi.xs) XOR
Nc(1)).Nc(1).h(idi.xs) XOR x250
(c,3) -> i: h(h(idi.xs) XOR x250).x250
```

La figure II. 4 illustre le résultat de la vérification du protocole précédent sous forme MSC qui contient la trace d'attaque:

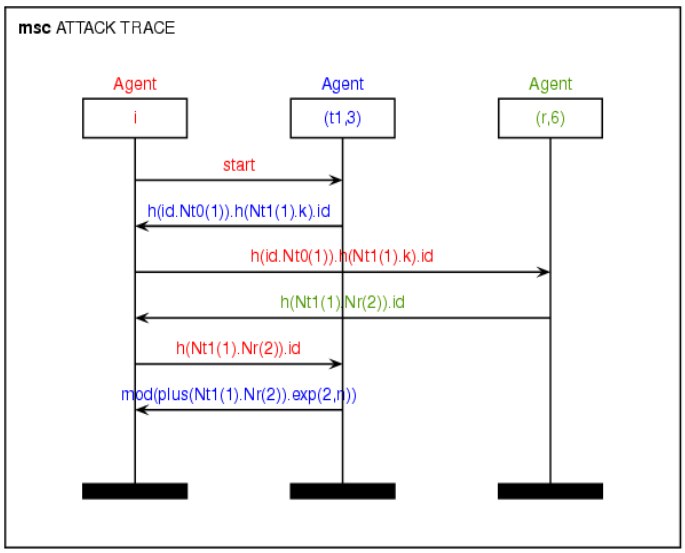

Figure II.3 : Trace d'attaque sur le protocole KN (OFMC)

II.7. Outil graphique SPAN

SPAN (Security Protocol ANimator for AVISPA) [29] est un animateur de protocole de sécurité. SPAN permet d'animer les spécifications HLPSL, c'est-à-dire produire interactivement des MSC (Message Sequence Charts) [30] qui peuvent être vus comme une trace "Alice & Bob" d'une spécification HLPSL. Cet outil permet de simuler le protocole, l'intrus et l'attaque.

Dans la Figure II.4, le cadre de droite (1) ainsi que dans la fenêtre à gauche (2), SPAN affiche les messages déjà envoyés. Dans le cadre de gauche (3), on voit les transitions possibles(les envois de messages) qu'on peut déclencher d'un double-clique. S'il n'y a plus de transitions, c'est qu'on est arrivé à la fin du protocole ou qu'il y a une erreur dans la spécification HLPSL.

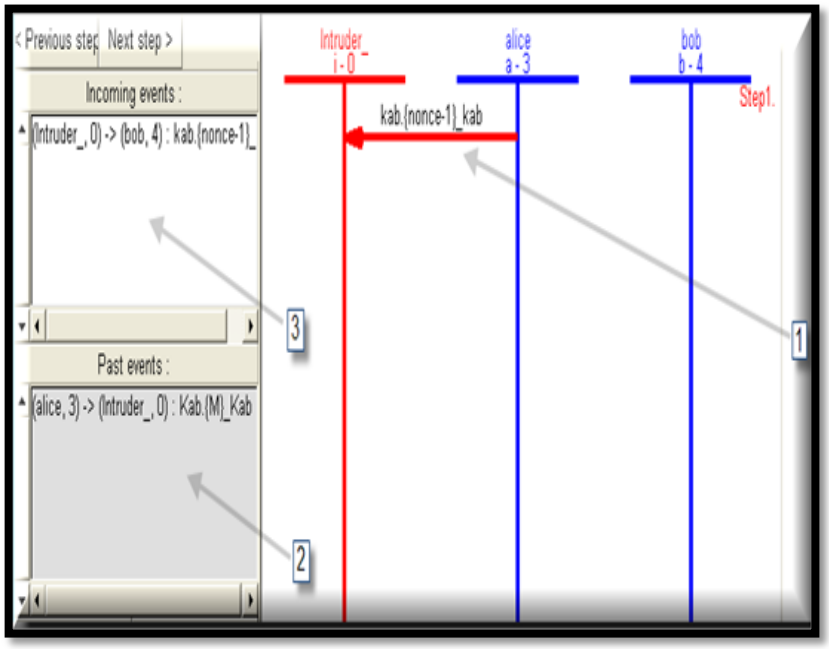

Figure II.4 : L'animation de la spécification HLPSL de *fil-rouge*

SPAN a plusieurs utilités dont celle de vérifier que la spécification HLPSL correspond bien au protocole qu'on avait en tête avant même de chercher des attaques. La plus grande utilité de SPAN est la possibilité de reconstruire des attaques fournies par les outils de vérification. SPAN permet à l'utilisateur de choisir à chaque étape si les messages sont reçus par l'intrus ou non et affiche dans un cadre les connaissances de l'intrus.

II.8. Synthèse

Les motivations choisies pour la plateforme AVISPA pour vérifier les protocoles de sécurité sont multiples, elles sont en majorité:

- La nature du langage de spécification HLPSL, tel qu'un langage devient facile à manipuler, haut niveau, modulaire et expressif pour les protocoles et les propriétés de sécurité vérifiées.
- La plateforme AVISPA n'est pas un seul outil de vérification automatique, mais elle contient quatre back-end et chaque back-end possède des techniques particulières pour vérifier le protocole. Pendant la vérification, la considération des nombres des sessions est facteur important, les outils AVISPA sont partagés en deux catégories : *(1)* OFMC, CL-Aste et SATMC (nombre fini de sessions), *(2)* TA4SP (nombre non borné de sessions).
- La forme de la plateforme AVISPA est en plusieurs modes (Interface Web, mode Mac, Linux), ceci permet d'interaction avec les utilisateurs.
- Les outils AVISPA ne sont pas restés sans améliorations après la publication en juillet 2005, mais ils ont connu des mises à jour et des améliorations sur leur efficacité.

Chapitre III

Contribution : Spécification et Vérification des Protocoles de Sécurité

Sommaire

III.1. Introduction

Tout au long de ce chapitre, nous allons étudier des protocoles d'authentification qui utilisent des primitives cryptographiques, et qui peuvent être spécifiés en HLPSL pour qu'ils deviennent vérifiables à l'aide des outils AVISPA. Cette vérification particulière touche les transmissions sur les canaux publics, car ce dernier peut subir des attaques par un intrus.

III.2. Les propriétés à vérifier

On doit vérifier les propriétés de sécurité suivantes :

Confidentialité : on peut l'appeler aussi *secret*.

- Pour le système RFID, la vérification que la clé secrète (*K*), et/ou l'identificateur du tag (*ID*), ne soit jamais transmise en clair sur l'interface radiofréquence qui peut être espionnée.
- Pour le système carte à puce, la vérification que la clé secrète (Xs ou PWi), et/ou l'identificateur de la carte (*ID*), ne soit jamais transmise en clair sur les canaux publics qui peut être espionnée.

Authentification : il existe deux entités d'authentification qu'il faut vérifier et qui sont :

- **L'authentification du tag ou carte :** Un lecteur (ou le serveur) doit être en mesure de vérifier un tag (ou carte) correct pour authentifier et identifier un tag (ou carte) en toute sécurité.
- **L'authentification du lecteur ou serveur:** Un tag (ou carte) doit être en mesure de confirmer qu'il communique avec le lecteur (ou serveur) correct.

III.3. Les scénarios de la vérification :

On propose trois scénarios de vérification des protocoles de sécurité étudiés. La spécification de ces scénarios en HLPSL est représentée dans le rôle `environement`.

Scénario 1: ce scénario dépend du traitement d'une session du protocole qui concerne un tag (carte) légitime possède une clé partagée et le même lecteur (serveur) légitime. Ci-dessous la spécification de scénario du protocole OTYT [32].

```
role environment() def=
const t, r : agent,
       k :symmetric_key ,
       h :hash_func ,
       auth_tag:protocol_id
       intruder_knowledge = {t,r,h}
    composition

% spécification avec le premier scénario:
    Session(t,r,k,h)
end role
```

Scénario 2: ce scénario dépend du traitement de deux sessions du protocole qui sont en parallèle (on la note par le symbole `/\`) concernant un tag légitimes possède la même clé partagée et le même lecteur légitime. Ce scenario permet de détecter les attaques du type "*Attaque par rejeu*" s'il existe. Ci-dessous la spécification de scénario de même protocole.

```
role environment() def=
const t, r : agent,
       k :symmetric_key ,
       h :hash_func ,
       auth_tag :protocol_id
       intruder_knowledge = {t,r,h}
    composition

% spécification avec le premier scénario:
    session(t,r,k,h) /\ session(t,r,k,h)
end role
```

Scénario 3: ce scénario dépend du traitement de deux sessions du protocole qui sont en parallèle concernant deux tags légitimes différents possédant la même clé, même identificateur partagée et le même lecteur légitime. Ci-dessous la spécification de scénario du même protocole.

```
role environment() def=
const t1,t2, r : agent,
        k: symmetric_key ,
        h: hash_func ,
        auth_tag: protocol_id
        intruder_knowledge = {t1,t2,r,h}
    composition

% spécification avec le premier scénario:
     session(t1,r,k,h) /\ session(t2,r,k,h)
end role
```

III.4. Les protocoles des systèmes RFID :

L´identification par radiofréquence (RFID) est une technologie qui utilise des ondes radio à transmettre et à identifier de manière unique les objets. Les systèmes d'identification par radiofréquence (RFID) sont composés de trois entités : le tag (ou l'étiquette) se constitue d'une puce et d'une antenne, le lecteur qui communique avec le tag par des ondes radiofréquences et le serveur (ou base de données, back-end) qui utilise les informations obtenues à partir du lecteur.

Nous effectuons une étude de cas sur les protocoles RFID pour présenter les attaques possibles contre le système RFID.

III.4.1. Protocole KN

III.4.1.1. Description du protocole :

Ce protocole est proposé par Kang et Nyang [31]. Dans ce protocole, le tag génère une valeur r_0 hasard d'un petit domaine et une valeur aléatoire r_1 de longueur n. le tag calcule et envoie deux fonctions de hachage h (ID, r_0) et h (r_1, k), et ID $\oplus$ r_1 au lecteur. Après la vérification de l'authentification du tag par la comparaison les fonctions émis et les fonctions calculées, le lecteur génère une valeur aléatoire r_2 de longueur n et envoie ID $\oplus r_2$ et h (r_1, r_2) au tag. Le tag vérifie ceux-ci et envoie $r_1 + r_2 \bmod 2^n$ vers le lecteur. Le tag et le lecteur actualisent l'ID par xor avec $r_1 \oplus r_2$. Ce protocole est décrit dans la figure III.1.

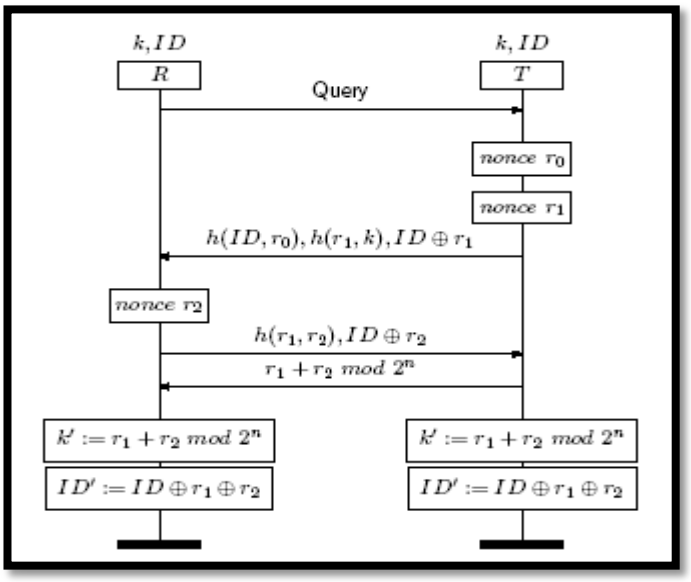

Figure III.1: Le protocole KN [31]

La description de ce protocole par la notation Alice-Bob est comme ci-dessous:

1 : T → R : H (ID, Nt0), H (Nt1, K), ID ⊕ Nt1
2 : R → T : H (Nt1, Nr), ID ⊕ Nr
3 : T → R : Nt1+ Nr Mod 2^n

Pour la spécification en HLPSL, les opérateurs comme le plus (+) et le mod ne sont pas supportées en HLPSL, on modélise ces opérateurs comme des fonctions de hachage, et l'intrus peut intercepter et calculer ses fonctions. La spécification HLPSL complète de ce protocole est donnée dans l'annexe A.

III.4.1.2. Les résultats de la vérification :

Scénario 01et 02 :

Le résultat de la vérification du protocole KN avec le premier scénario et le deuxième par la plateforme AVISPA ne détecte pas des attaques, cela signifie que le protocole est Sûr. Ce résultat est montré dans la figure III.2.

```
AVISPA Tool Summary
OFMC    : SAFE
CL-AtSe : SAFE
SATMC   : INCONCLUSIVE
TA4SP   : INCONCLUSIVE
Refer to individual tools output for details
```

Figure III.2: le résultat de vérification du protocole KN

Scénario 03 :

Après l'exécution, les outils AVISPA détectent une faille et affichent le message «*UNSAFE*». L'outil génère automatiquement et affiche la trace d'attaque sur l'authentification du lecteur. La figure III.3 montre la trace d'attaque découverte par OFMC. Dans ce résultat d'attaque, i représente l'intrus, (t1, 3) le tag et (r, 6) le lecteur. La signification des informations affichées tel que : Nt0(1) et Nt1(1) sont des instances du nonce Nt0, Nt1 respectivement et Nr(2) est instance du nonce Nr.Dans cette attaque, on peut noter les remarques suivantes:

- Le tag génère deux nonces Nt0(1) et Nt1(1) et calcule les deux hachages h (id.Nt0(1)), h(Nt1(1).k) et id XOR Nt1(1) puis l'envoi à l'intrus. L'intrus renvoie le message reçu au lecteur.
- Le lecteur produit un nombre aléatoire Nr(2) et l'envoi h (Nt1(1).Nr(2)).id XOR Nr(2) à l'intrus. L'intrus renvoie le message reçu au tag.
- Le tag calculer l'opération mod (plus (Nt1(1).Nr(2)).exp (2, n)) et l'envoi à l'intrus.

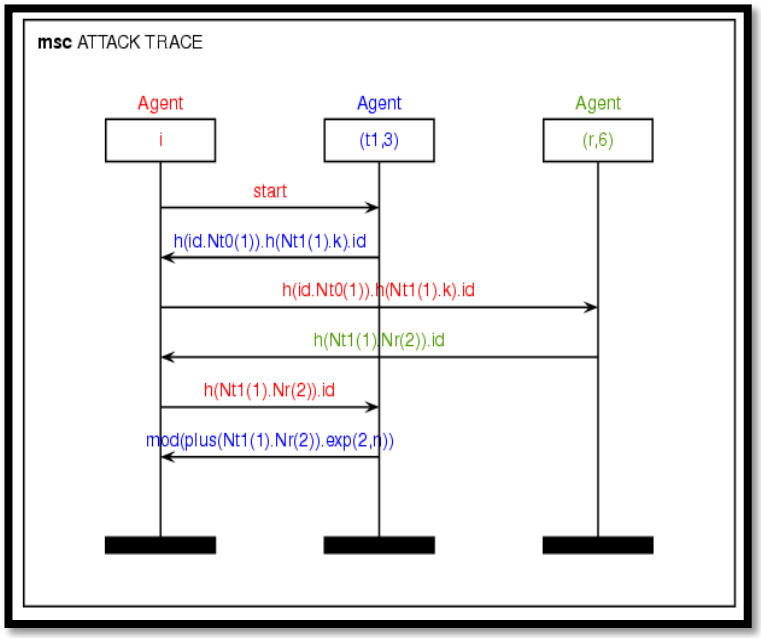

Figure III.3 : Trace d'attaque sur le protocole KN (OFMC)

L'intrus fait donc un relais sur les deux premiers messages, mais le dernier $(N_{t1} + N_r \, Mod \, 2^n)$ est inutile parce que l'objectif du dernier message est l'authentification du tag. Donc, l'intrus a réussi son attaque à la fin du deuxième message $(H\,(N_{t1}, N_r), ID \oplus N_r)$.Le but de l'intrus est de faire une usurpation de l'identité du lecteur.

III.4.2. Protocole OTYT

III.4.2.1. Description du protocole :

La référence [32] représente le protocole OTYT. Ce protocole basé sur une fonction de hachage et d'une clé symétrique. Dans La Figure III.4 l'échelle de temps va de haut en bas. C'est à dire, le plus haut message est envoyé d'abord et le plus bas message est envoyé dernier.Le lecteur R et le tag T partagent le même secret k. Le lancement par le lecteur qui génère un nombre aléatoire r_1 frais et l'envoie au tag. Le tag calcule la fonction de hachageh ($k \oplus r_1$) et la transmet au lecteur. Le lecteur génère un nonce k_1. Il calcule ensuite l'opération de bits exclusif-ou ($\oplus$) de K et k_1, ensuit envoie vers le tag. Le lecteur et le tag effectuent la mise à jour de la clé k. La spécification de protocole OTYT complet se lit par le langage HLPSL est donnée dans l'annexe B.

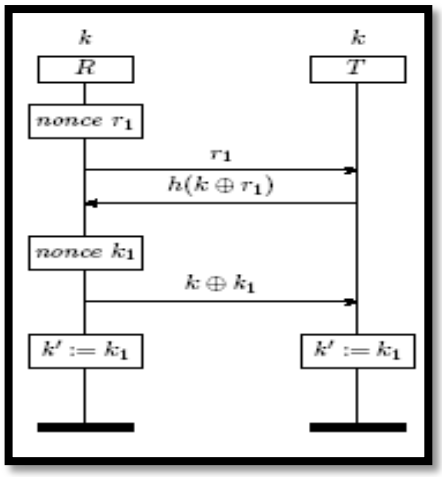

Figure III.4: Le protocole OTYT [32]

La spécification du protocole sous la forme Alice-Bob comme suit :

```
1 : R → T : Nr
2 : T → R : H (k ⊕ Nr)
3 : R → T : k ⊕ Nt
```

III.4.2.2. Les résultats de la vérification :

Scénario 01 :

Après la vérification de protocole OTYT avec le premier scénario par les outils AVISPA. La Figure III.5 illustre le résultat de ce protocole. Ce résultat signifie en clair qu'il n y a pas d'attaque. On peut ainsi déduire que le diagnostic des outils AVISPA et SPAN pour ce protocole est sûr.

```
AVISPA Tool Summary
OFMC    : SAFE
CL-AtSe : SAFE
SATMC   : INCONCLUSIVE
TA4SP   : INCONCLUSIVE
Refer to individual tools output for details
```

Figure III.5: le résultat de vérification du protocole OTYT

Scénario 02

Pour le deuxième scénario, Les outils AVISPA détectent une attaque du type "*attaque par rejeu*". La figure III.6 montre la trace d'attaque avec l'outil CL-Atse. Dans cette trace d'attaque, l'intrus exécute deux sessions en parallèle. La première session implique deux participants (r, 7) qui joue le rôle de lecteur et (t1, 6) qui joue le rôle d'un tag. Le lecteur génère un nombre aléatoire n13 (nr) et l'envoie à l'intrus. L'intrus génère un autre nonce Nr (9) puis l'envoi au tag. Le tag calcule la fonction h (xor (Nr(9), k)) et transmet à l'intrus. La deuxième session implique de participants (r, 4) qui joue le rôle de lecteur et (t1, 3) qui joue le rôle d'un tag. Le lecteur génère un nonce n13(Nr) et l'intrus capte et enregistre ce nonce au cours de la communication et l'envoi au tag. Le tag calcule la fonction h (xor (k, n5(Nr))) et la transmet à l'intrus. L'intrus envoie la fonction reçue au lecteur. Le lecteur produit un nouveau nonce n6 (k1) et calcule l'opération xor (k, n6(K1)) puis la transmet à l'intrus. L'intrus envoi le message reçu au tag en deux sessions (t1, 3) et (t1, 6).

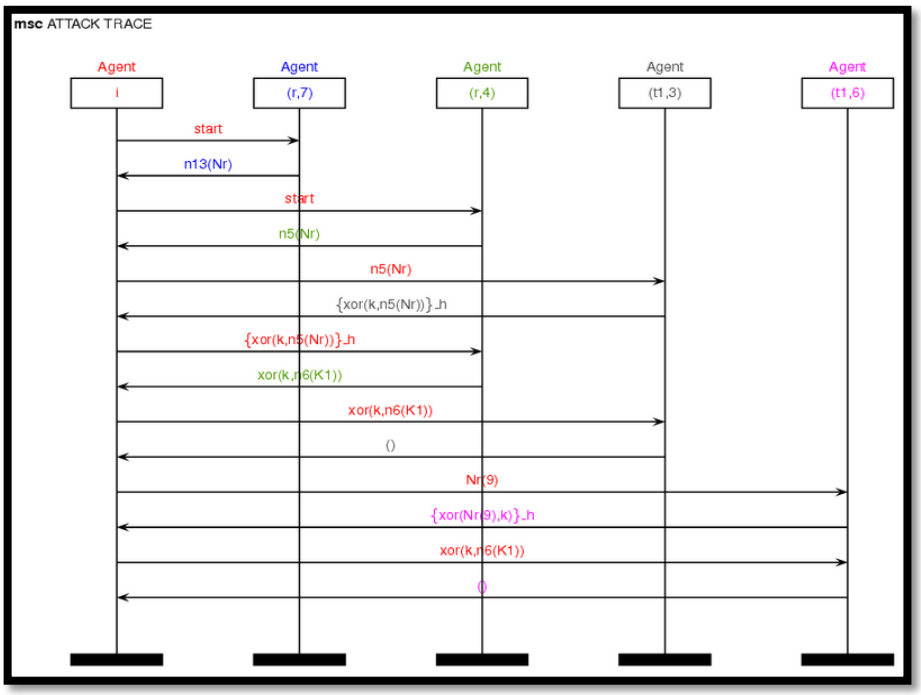

Figure III.6: Trace d'attaque sur le protocole OTYT (CL-Atse)

Scénario 03 :

Les outils AVISPA détectent une trace d'attaque sur l'authentification du tag. La figure III.7 montre la trace d'attaque du protocole OTYT avec l'outil CL-Atse. Dans ce résultat d'attaque, i l'intrus, (r,4) et (r,7) représentent le lecteur, et (t1,3) le tag. La signification des informations affichées tel que :n13(Nr) et n5(Nr) des instance du nonce Nr. n14(k1) l'instance du nonce Nt. la fonction « {}_h » signifie une fonction du hachage.On note que le principe de cette trace est le même principe de la trace d'attaque du protocole KN pour les mêmes scénarios, mais cette fois l'intrus initialise deux sessions en parallèle `(r,3)` et `(r,6)`.

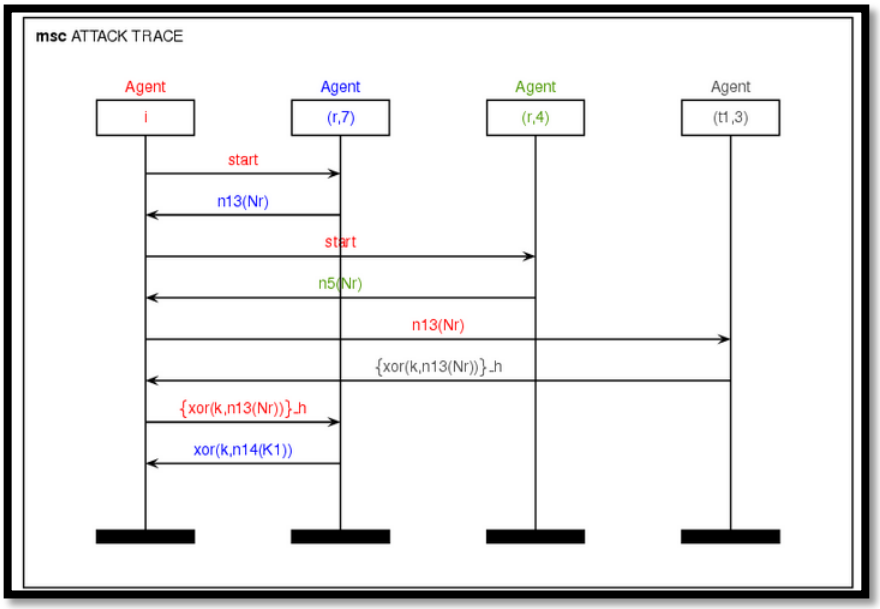

Figure III.7: Trace d'attaque sur le protocole OTYT (CL-Atse)

III.4.3. Protocole CD

III.4.3.1. Description du protocole :

Le protocole CD est un protocole d'authentification mutuelle proposé par Chen et Deng [33].Ce protocole est lancé par le lecteur, tel que le lecteur génère un nonce r_R et calcule la fonction CRC ($N_i \oplus r_R$) et l'envoie avec M_{req} au tag. Une fois reçus le message, le tag génère un nonce r_T, calcule $X= K_i \oplus EPC_t \oplus r_T$, $Y= CRC (r_T \oplus N_i \oplus X)$ et l'envoie r_T, X, Y au lecteur. Le lecteur envoi le message M_{resp} au tag. La figure III.8 décrit ce protocole.

La notation Alice-Bob de ce protocole est la suivante :

$1 : R \rightarrow T : M_{req}, Nr, CRC (Ni \oplus Nr)$
$2 : T \rightarrow R : Nt, K_i \oplus EPC_t \oplus Nt, CRC (Nt \oplus Ni \oplus K_i \oplus EPC_t \oplus Nt)$
$3 : R \rightarrow T : M_{resp}$

La spécification HLPSL complète de ce protocole est donnée dans l'annexe C.

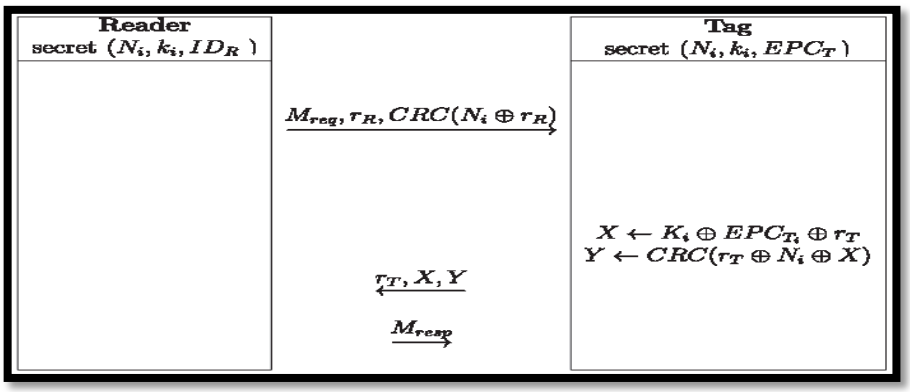

Figure III.8: Le protocole CD [33]

III.4.3.2. Les résultats de la vérification :

Scénario 01 et 03 :

Les deux outils CL-Atse et OFMC détectent une trace d'attaque sur l'authentification du lecteur. La figure III.9 montre la trace d'attaque du protocole CD avec l'outil CL-Atse. La signification des informations affichées tel que : `n5(Nr)` est l'instance du nonce Nr. `n9(Nt)` est l'instance du nonce Nt. La remarque qui peut être tirée à partir de la trace est :

- l'existence de deux phases de communication. La première est entre les agents i et r, et quant à la deuxième phase, elle située entre les agents i et t2, d'où l'exclusion définitive de la communication avec le lecteur légitime.

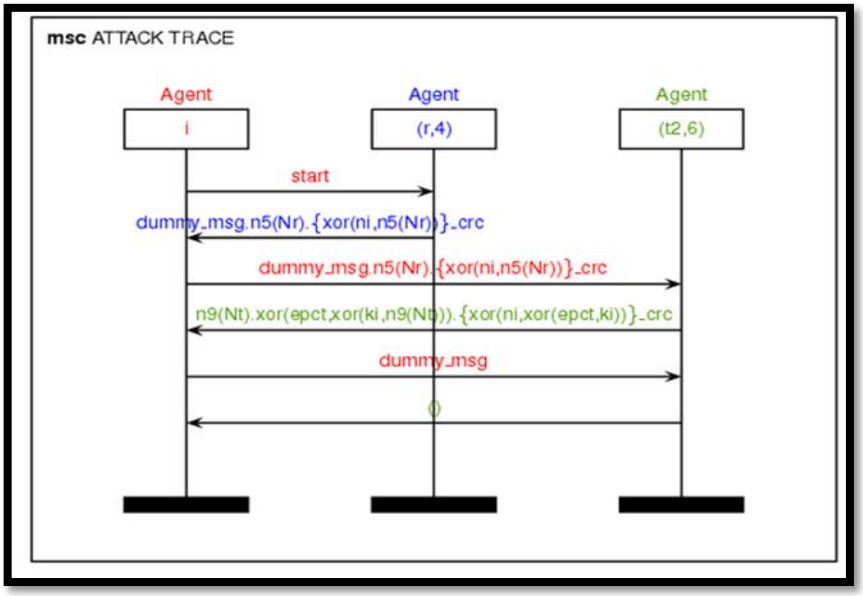

Figure III.9 : Trace d'attaque sur le protocole CD (CL-Atse)

Scénario 02:

La Figure III.10 montre le résultat d'AVISPA par l'outil CL-Atse sur la spécification de protocole CD. Cette trace d'attaque est du type "*attaque par rejeu*". Le principe de cette trace est le même pour le protocole YPL dans le même scénario, mais cette fois l'intrus initialise un seul session `(r,4)`.

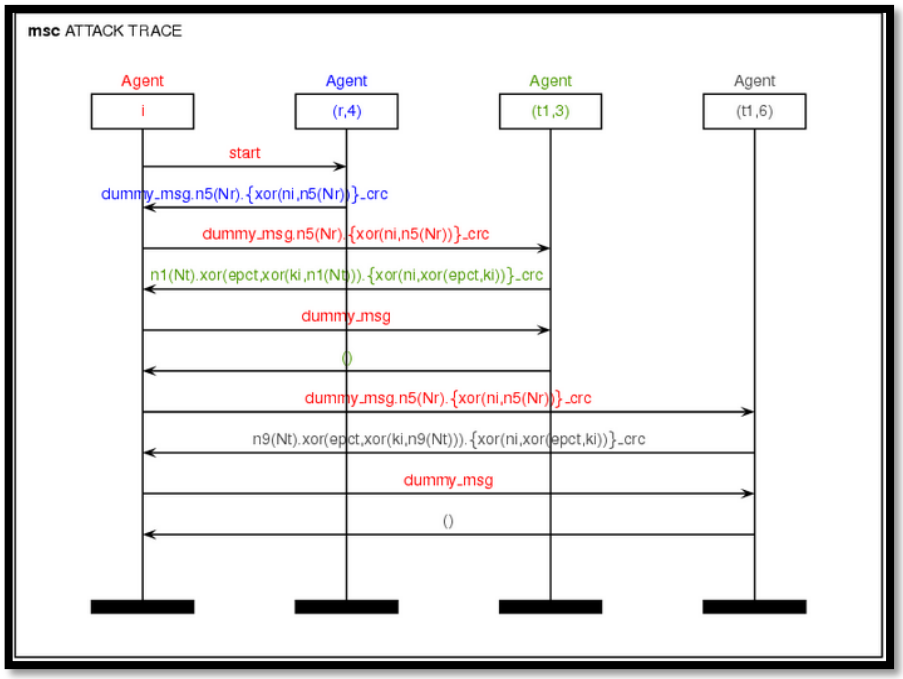

Figure III.10 : Trace d'attaque sur le protocole CD (CL-Atse)

III.4.4. Protocole YLP

III.4.4.1. Description du protocole :

La référence [34] représente le protocole YPL. Ce protocole basé sur l'authentification mutuelle entre le tag et le lecteur, la figure III.11 fournit un aperçu global de ce protocole. Le lecteur R et le tag T partagent des secrets k, k_1, k_2. Le lecteur produit un nonce r_1 et l'envoie ce nonce au tag, le tag calcule la fonction de hachage h ($k_1 \oplus r_1 \oplus k$) et le transmet au lecteur. Le lecteur calcule la fonction de hachage de k_2 et l'envoie au tag. Le tag vérifie cette fonction de hachage reçue par calcule une nouvelle h (k_2) et se comparer avec h (k_2) reçue. Si sont égaux, l'authentification mutuelle est enfin réussie, ensuite le tag et le lecteur effectuent les mises à jour des secrets partagés k_1 et k_2 de façon : h ($k_1 \oplus r_1 \oplus k$) et h (k_2) respectivement. La description du protocole sous la forme Alice-Bob comme suit :

1 : R → T : Nr
2 : T → R : H ($k_1 \oplus$ Nr $\oplus$ k)
3 : R → T : H (k_2)

La spécification par le langage HLPSL de ce protocole complet est donnée dans l'annexe D.

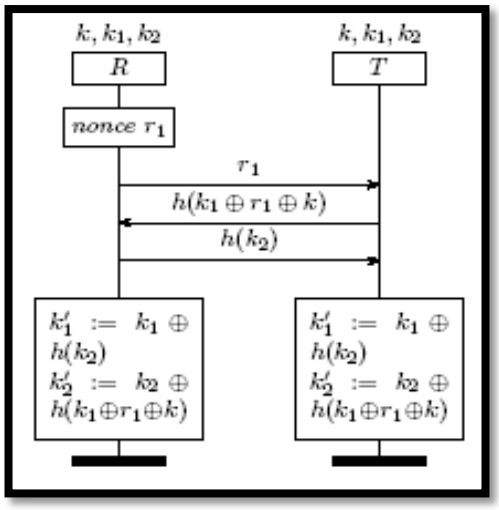

Figure III.11: Le protocole YLP [34]

III.4.4.2. Les résultats de la vérification :

Scénario 01 :

La vérification sur la spécification du protocole YLP avec le premier scénario n'a montré aucune attaque c'est-à-dire le protocole est *sûr*. Le résulta de cette vérification est décrit dans la Figure III.12.

```
AVISPA Tool Summary
OFMC    : SAFE
CL-AtSe : SAFE
SATMC   : INCONCLUSIVE
TA4SP   : INCONCLUSIVE
Refer to individual tools output for details
```

Figure III.12: le résultat de vérification du protocole YPL

Scénario 02

Pour le deuxième scénario, le diagnostic conclu après la vérification est que la détection une trace d'attaque du type "*attaque par rejeu*". La figure III.13 montre la trace d'attaque avec l'outil CL-Atse. Le principe de cette trace est le même pour le protocole OTYT dans le même scénario

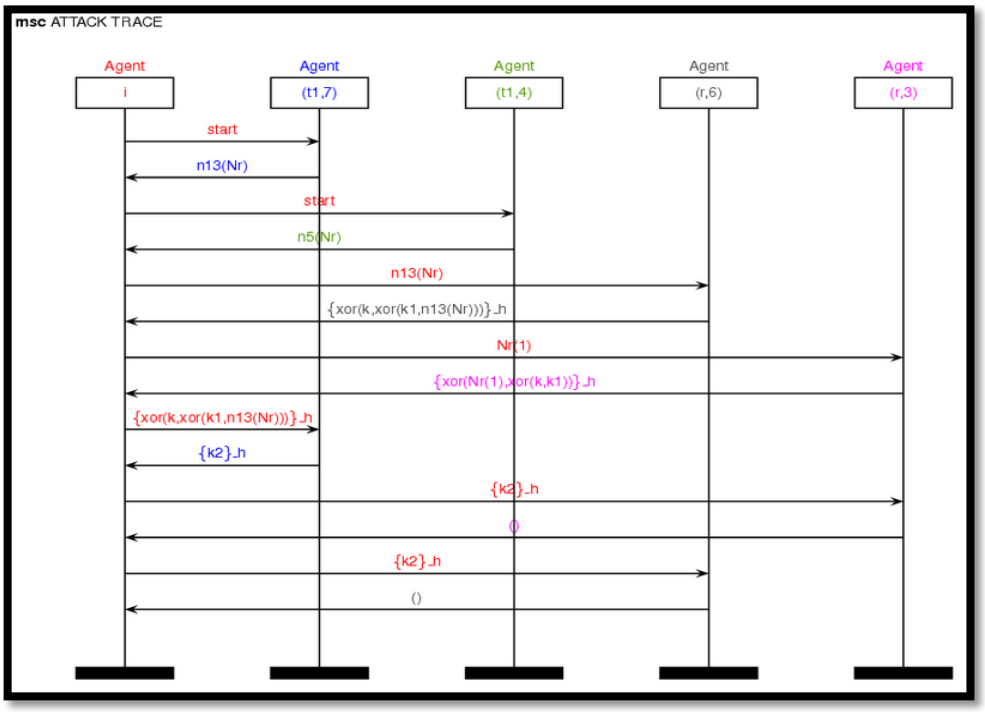

Figure III.13: Trace d'attaque sur le protocole YPL (CL-Atse)

Scénario 03 :

Les outils OFMC et CL-ATSE détectent une trace d'attaque. Cette trace d'attaque est la violation de l'authentification du tag. Le principe de cette trace est le même principe de la trace d'attaque du protocole OTYT avec les mêmes scénarios. La figure III.14 illustre cette trace.

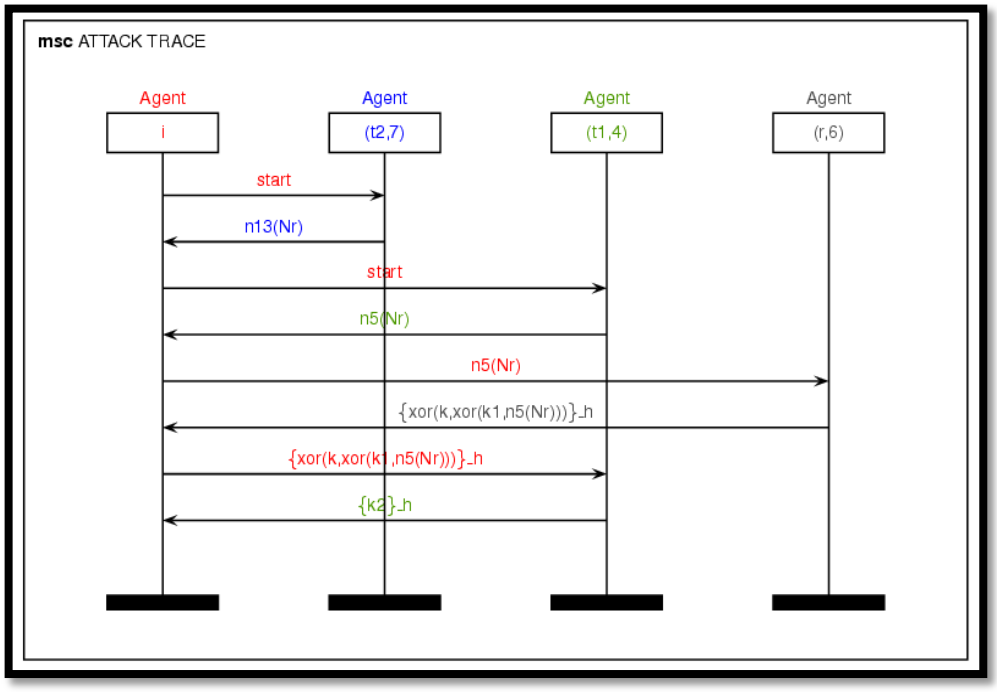

Figure III.14: Trace d'attaque sur le protocole YPL (CL-Atse)

III.4.5. Protocole KCL

III.4.5.1. Description du protocole :

Ce protocole est proposé par Kim et *al.* [35] représenté dans la figure III.15**.** Il est conçu pour authentifier un tag T au un lecteur R. Chaque tag possède un identificateur ID et une clé K, tous les deux connus au lecteur. Le lecteur lance le protocole en générant une valeur aléatoire fraîche r_1. Dès réception de la requête r_1, le tag génère un nonce r_2 et calcule ID $\oplus$ r_2, h (r_2, k) $\oplus$ r_2 et l'envoyé pour l'authentification du tag.

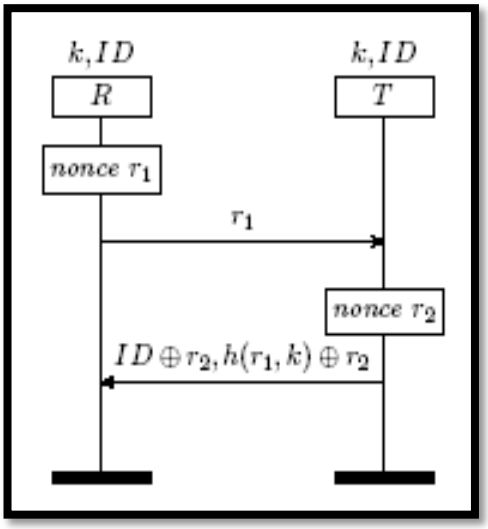

Figure III.15 : Le protocole KCL [35]

La description de ce protocole par la notation Alice-Bob est comme ci-dessous:

```
1 : R → T : Nr
2 : T → R : ID ⊕ Nt, H (Nr, k) ⊕ Nt
```

La spécification HLPSL de ce protocole est donnée dans l'annexe E.

III.4.5.2. Les résultats de la vérification :

Scénario 01

Concernant le premier scénario, les outils AVISPA affiche le message: «*UNSAFE*». La figure III.16 montre la trace d'attaque découverte par CL-Atse. Dans cet attaque l'intrus est de type passif, tel que l'adversaire ne pas faire des modifications sur les messages transmets mais de les retransmettre seulement.

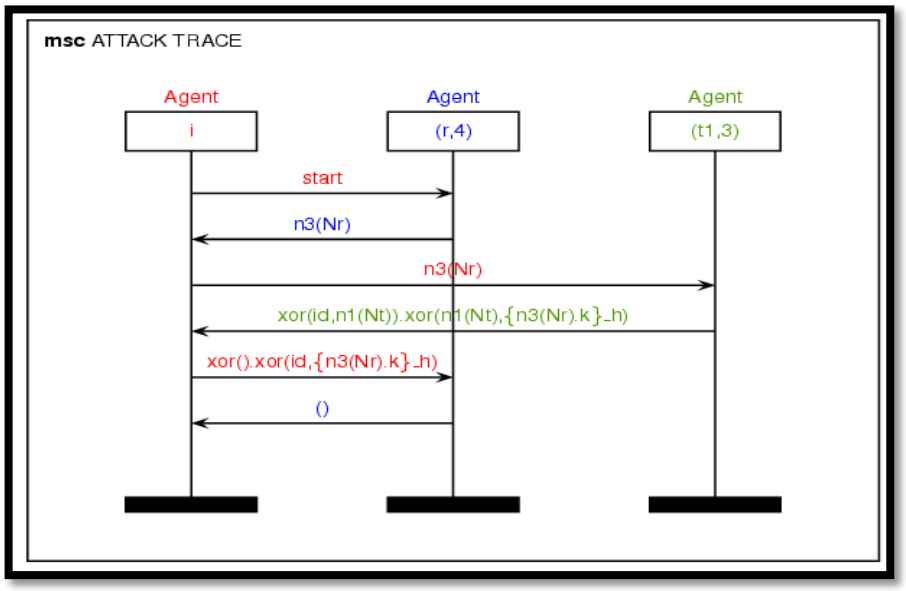

Figure III.16 : Trace d'attaque sur le protocole KCL (CL-Atse)

Scénario 02 et 03

Les résultats de vérification du protocole KCL dans le deuxième et le troisième scénario digest dans AVISPA sont exposés dans La Figure III.17. Dans ce cas, l'outil CL-Atse détecte une trace attaque sur l'authentification du tag. On note que le principe de cette trace est le même principe de la trace d'attaque avec le premier scénario, la seule différence est le nombre de sessions de lecteur. Avec ces scénarios, le nombre de session du lecteur est deux.

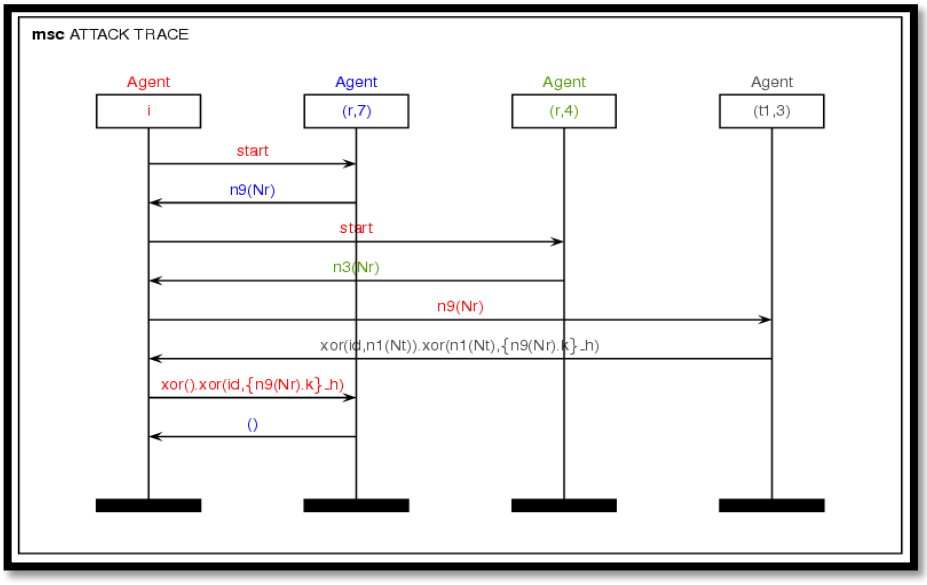

Figure III.17 : Trace d'attaque sur le protocole KCL (CL-Atse)

III.4.6. Protocole SR

III.4.6.1. Description du protocole :

C'est un protocole d'authentification dans un environnement RFID-Mobile. Il est proposé par Sandhya et Rangaswamy [36]. Dans ce protocole (voir la Figure III.18), Le lecteur génère et enregistre un nombre aléatoire r et envoie une requête au tag. Après avoir reçu le message de requête, le tag calcule H ($ID_t \oplus K_i$) $\oplus$ r et le transmet au lecteur. Le lecteur génère H(ID_R) et le même nombre aléatoire r qui a été envoyée au tag et l'envoie avec le message H ($ID_t \oplus K_i$) $\oplus$ r vers le serveur. Le serveur vérifie l'authenticité du lecteur en correspondre le code de hachage reçue du lecteur H(ID_R) avec le code de hachage stockée. S'il trouve une correspondance, alors le lecteur est légitime. Il utilise les informations sur le nombre aléatoire r et effectue l'opération XOR de celui-ci avec le code de hachage reçu H ($ID_t \oplus K_i$) $\oplus$ r pour obtenir la valeur de H ($ID_t \oplus K_i$). Ensuite, il vérifie l'authenticité de tag en faisant correspondre la valeur de code de hachage stockée avec le code de hachage H ($ID_t \oplus K_i$) obtenu. Si elles sont égales, le tag passe l'authentification, sinon, le tag n'est pas authentifié. Le serveur exécute l'opération XOR du nombre aléatoire g et r avec la fonction de hachage H (K_i). Ce message ainsi que les informations détaillées de tag D est transmis sous forme crypté au lecteur en utilisant la clé s de lecteur. Le lecteur décrypte et obtient les données D du tag et transmet l'information restante au tag. Le tag vérifie l'authenticité du lecteur en utilisant le nombre aléatoire r. Il effectue l'opération XOR de r et H(K_i) pour obtenir les détails de nombre aléatoire g.

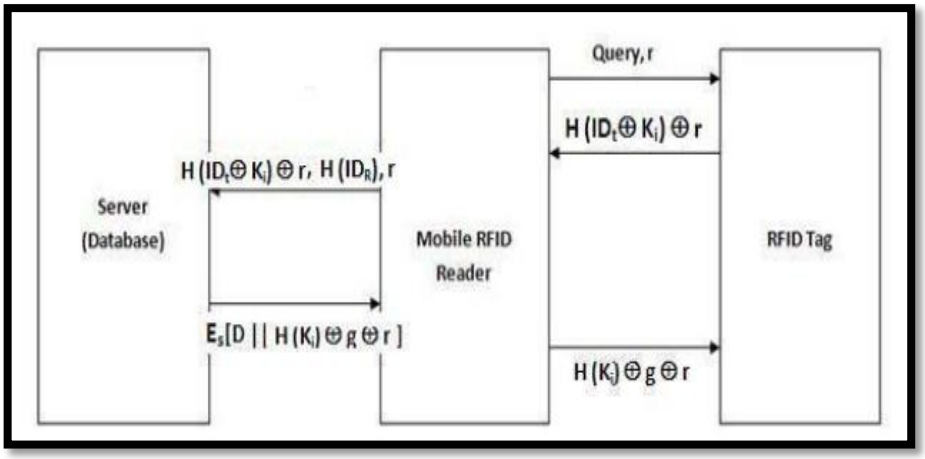

Figure III.18: Le protocole SR [36]

La spécification du protocole sous la forme Alice-Bob comme suit:

$$
\begin{aligned}
&1 : R \rightarrow T : Nr \\
&2 : T \rightarrow R : H\ (ID_t \oplus K_i)\ \oplus Nr) \\
&3 : R \rightarrow S : H(ID_t \oplus K_i) \oplus Nr),\ H(ID_r),Nr \\
&4 : S \rightarrow R : E_s[D,H(K_i)\ \oplus Nt\ \oplus\ Nr] \\
&5 : T \rightarrow R : H(K_i)\ \oplus Nt \oplus\ Nr
\end{aligned}
$$

La spécification HLPSL de ce protocole est présenté dans l'annexe F.

III.4.6.2. Les résultats de la vérification :

Scénario 1

Les outils AVISPA détectent une trace d'attaque sur l'authentification du lecteur. La figure III.19 montre la trace d'attaque du protocole SR avec l'outil CL-Atse. Cette attaque est du type " *man in the middle* ", tel que l'adversaire fait des modifications sur les messages transmets et de les retransmettre.

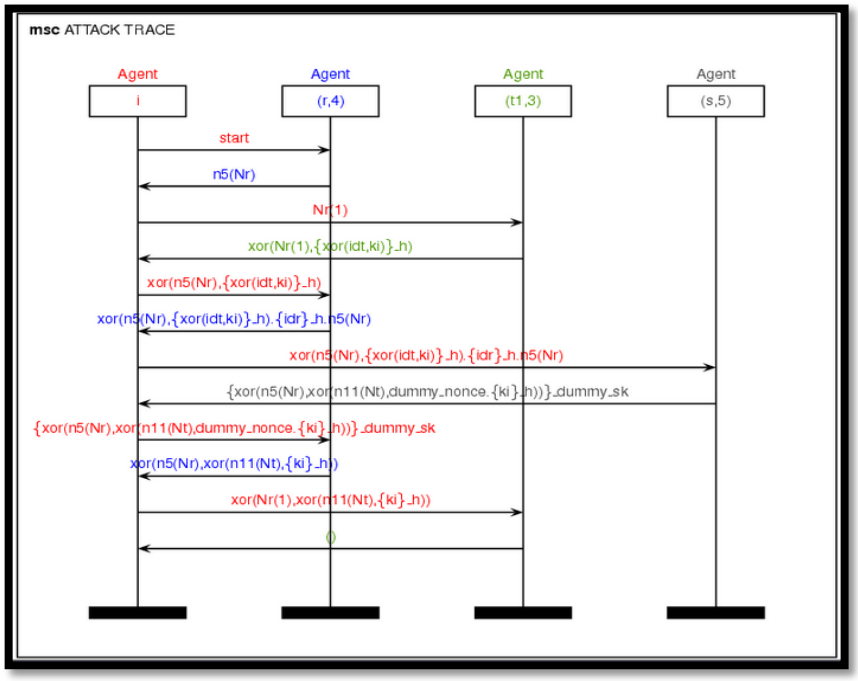

Figure. III.19. Trace d'attaque sur protocole de SR (CL-Atse)

Scenario 02 et 03

Pour le deuxième et troisième scénario, Les outils AVISPA détectent une trace d'attaque sur l'authentification du lecteur. La figure III.20 montre la trace d'attaque avec l'outil CL-Atse. On note que le principe de cette trace est le même principe de la trace d'attaque

avec le premier scénario, la seule différence est le nombre de sessions de lecteur. Pour ces scénarios, l'intrus exécute deux sessions en parallèle avec deux lecteurs.

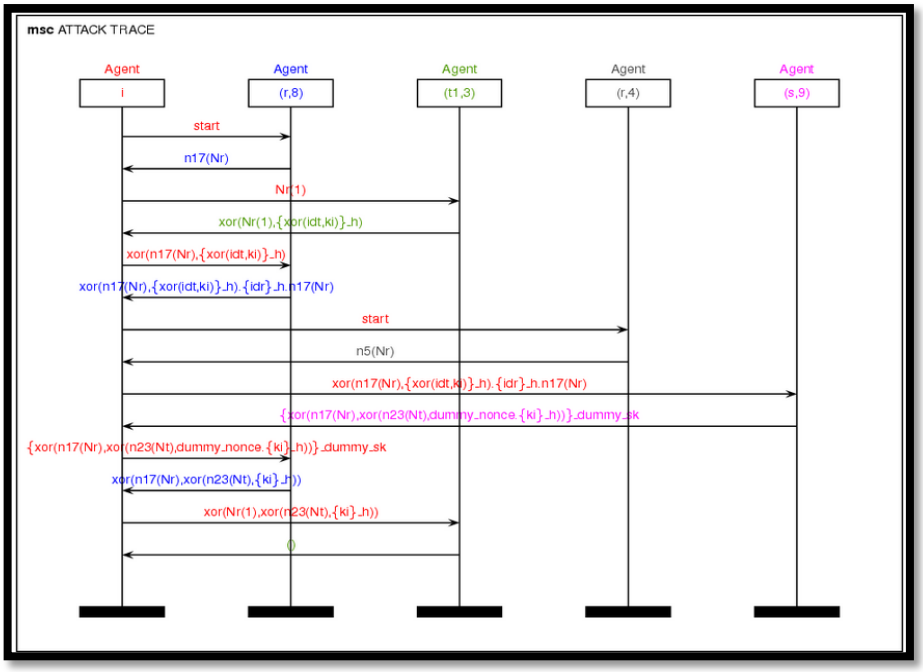

Figure III.20 : Trace d'attaque sur protocole de SR (CL-Atse)

III.5. Protocoles des systèmes carte à puce

Les systèmes de carte à puce est composé en deux entités : le client et le serveur, lorsque le client veut se connecter à un serveur distant, le serveur distant demande un mot de passe pour authentifier l'identité du client. Nous effectuons une étude de cas sur les protocoles des systèmes carte à puce pour présenter les attaques possibles contre le système carte à puce.

III.5.1. Protocole LH

III.5.1.1. Description du protocole :

Ce protocole est proposé par Li et Hwang [37], Ce protocole est partagé en deux phases : Phase de connexion et phase d'authentification.

Phase de connexion

Quand l'utilisateur veut connecter au serveur distant, il/elle doit exécuter les étapes suivantes :

- Étape 1: Premièrement, le client insère sa carte à puce dans le lecteur et entrées la biométrie personnelle B_i sur l'appareil spécifique pour vérifier la biométrie de l'utilisateur.

- Etape 2: Ensuite, vérifie $h(B_i) = f_i$.
- Étape 3: Si ce qui précède ne tient pas, cela signifie le client ne passe pas la vérification biométrique et le système d'authentification de l'utilisateur distant est terminée. Au contraire, si elle s'il se tient, le client passe la vérification biométrique. Puis il entrées du PWi pour exécuter les opérations suivantes dans l'étape 4.
- Étape 4:Après avoir reçu le mot de passe du client, la carte à puce va calculer les messages suivants: $r_i' = h(PW_i \| f_i)$, $M_1 = e_i \oplus r_i' = h(ID_i \| Xs)$, $M_2 = M_1 \oplus Rc$, Où Rc est un nombre aléatoire généré par l'utilisateur. Pour cette étape, la valeur aléatoire Rc est introduite pour masquer le hachage de la valeur secrète $h(D_i \| Xs)$.
- Etape5: Enfin, le client envoie le message (ID_i, M_2) vers le serveur S_i distant.

Phase d'authentification

Après avoir reçu le message de demande de connexion, le serveur doit exécuter les étapes suivantes pour authentifier l'utilisateur.

- Étape 1: Tout d'abord, le serveur vérifie si le format de l'ID_i est valide ou non.
- Étape 2: Si l'ID_i est valide, le serveur calcule les messages suivants pour fournir une authentification mutuelle entre client et serveur. Pour cette étape, M_4 est la valeur aléatoire Rc du client C_i et seul le serveur peut démasquer la valeur, parce que lui seul peut calculer $h(ID_i \| Xs) : M_3 = h(ID_i \| Xs), M_4 = M_2 \oplus M_3 = Rc, M_5 = M_3 \oplus Rs, M6 = h(M_2 \| M_4)$
- Étape 3: Ensuite, le serveur envoie le message (M_5, M_6) au client.
- Étape 4: Après avoir reçu le message du serveur, le client vérifie l'égalité de $M_6 = h(M_2 \| Rc)$.
- Étape 5: Si $M_6 = h(M_2 \| Rc)$, le client estime que le serveur est authentifié et puis calcule les messages suivants pour fournir une authentification mutuelle entre le serveur et le client. Pour cette étape, M_7 la valeur aléatoire Rs du serveur et seulement le client, qui sait $M_1 = h(ID_i \| Xs)$ peut renvoyer la valeur hachée correct de $M_8 = h(h(ID_i \| Xs) \oplus Rs) \| Rs)$, $M_7 = M_5 \oplus M_1 = Rs$, $M_8 = h(M_5 \| M_7)$
- Étape 6 : le client envoie le message M_8 au serveur.
- Étape 7 : Après avoir reçu le message de client, le serveur vérifie si $M_8 = h(M_5.Rs)$ Etape 8 : si $M_8 = H(M_5 \| Rs)$, le serveur accepte la demande de connexion de client. Etape 9 : Dans le contraire, le serveur rejette la demande de connexion de client.

La description de ce protocole par la notation Alice-Bob est comme ci-dessous:

1: C → S : ID_i, M_2 % tel que M_2=H(ID_i, Xs)⊕Nc
2: S → C : M_6, M_5 % tel que M_5=H(ID_i,Xs)⊕ Ns et
M_6=H (H (ID_i, Xs) ⊕Nc),Nc
3: C → S : M_8 % tel que M_8=H(H(ID_i, Xs)⊕ Ns),Ns

La spécification HLPSL complète de ce protocole est donnée dans l'annexe G.

III.5.1.2. Les résultats de la vérification :

Les Scénarios 1,..,3

Après la vérification de protocole LH par les outils AVISPA. On note que tous les scénarios donnent la même trace d'attaque. La figure III.21 illustre la trace d'attaque sur ce protocole détectée par l'outil OFMC. Cette trace d'attaque est la violation de l'authentification du serveur. Les données Ns(2) et X250 sont des instances du nonce Ns et Nc(1) est instance du nonce Nc. Dans ce dernier, l'adversaire est retransmis les messages des agents honnêtes et modifier des données sur le serveur. On note que le principe de cette trace est le même principe de la trace d'attaque du protocole KN. La seul différence est la modification du nonce Ns se fait par l'intrus.

```
% OFMC
% Version of 2006/02/13
SUMMARY
UNSAFE
DETAILS
  ATTACK_FOUND
PROTOCOL
  /home/avispa/web-interface-
computation/./tempdir/workfileThH3Ff.if
GOAL
authentication_on_aut_server
BACKEND
  OFMC
COMMENTS
STATISTICS
parseTime: 0.00s
searchTime: 0.03s
visitedNodes: 3 nodes
  depth: 2 plies
ATTACK TRACE
i -> (c,3): start
(c,3) -> i: idi.h(idi.xs) XOR Nc(1)
i -> (s1,3):idi.h(idi.xs) XOR Nc(1)
(s1,3) -> i:h(h(idi.xs) XOR
Nc(1)).Nc(1).h(idi.xs) XOR Ns(2)
i -> (c,3): h(h(idi.xs) XOR
Nc(1)).Nc(1).h(idi.xs) XOR x250
(c,3) -> i: h(h(idi.xs) XOR x250).x250
```

Figure III.21 : Trace d'attaque sur protocole de LH (OFMC)

III.5.2. Protocole LNMW

III.5.2.1. Description du protocole :

Ce protocole est proposé par Li et al [38], il est partagé en deux phases : phase de connexion et phase d'authentification.

Phase de connexion

Lorsque l'utilisateur C_i veut se connecter au serveur distant S_i, il a besoin d'effectuer les étapes suivantes:

- Étape 1: L'utilisateur C_i insère d'abord son carte à puce dans le lecteur et fournit sa biométrie personnelle B_i sur l'appareil spécifique pour vérifier la biométrie de l'utilisateur. Si les informations biométriques correspondant au modèle mémorisé dans le système, le client passe la vérification biométrique, puis exécute les étapes suivantes.
- Étape 2: le client entrées son mot de passe PW_i et l'identité ID_i. Après avoir reçu l'ID_i de client et son mot passe PW_i la carte à puce calcule $RPW_i = h (N \| PW_i)$, $r_i' = h(RPW_i \| f_i)$.
- Étape 3: La carte à puce vérifie l'égalité $r_i = r_i'$. Dans le cas négatif, cela signifié que l'utilisateur entre son mot de passe incorrectement. L'utilisateur est averti par le message d'erreur de mot de passe incorrect et la procédure se termine.
- Étape 4: Dans le cas positive, la carte à puce calcule $M_1 = e_i \oplus r_i'$,$M_2 = M_1 \oplus Rc$, où Rc est un nombre aléatoire généré par l'utilisateur, $M_3 = h (y \| Rc)$, $M_4 = RPW_i \oplus M_3$, $M_5 = h (M_2 \| M_3 \| M_4)$.
- Étape 5 : Enfin, le client envoie le message (ID_i, M_2, M_3, M_4, M_5) au serveur Si.

Phased'authentification

Après avoir reçule message de demande de connexion (ID_i, M_2, M_3, M_4, M_5) de l'utilisateur, le serveur distant et l'utilisateur effectuez les étapes suivantes pour l'authentification mutuelle.

- Étape 1: le serveur vérifie premièrement le format d'ID_i. Si elle est valide, le serveur calcule $M_6 = h (ID_i \| Xs)$, $M_7 = M_2 \oplus M_6$, $M_8 = h (y \| M_7)$, puis vérifie $M_8 = M_3$. Si il se tient est vérifiée, alors le serveur vérifie davantage si $M_5 = h (M_2 \| M_8 \| M_4)$. S'il se tient, le serveur stocke (ID_i, M_7) dans sa base de données. Ainsi, lorsque le serveur reçu le prochain message de connexion de l'utilisateur, le serveur calcule M_7' et compare avec le M_7 qui stockées dans la base de données. Si ces valeurs sont les mêmes, le serveur rejette

parce que c'est un message de relecture. Si non, le serveur remplace l'ancien M_7 par le nouveau calculée M_7' .

- Étape 2: S'il étape 1 ne tient pas, cela signifie que le client n'est pas légitime, et donc, le serveur rejette la demande de connexion et met fin à la session. Autrement, le serveur accepte la demande de connexion et donc le client est authentifié par le serveur comme un utilisateur valide. Puis le serveur calcule $M_9=M_4 \oplus M_8$, $M_{10}=h\ (M_9\ ||\ SID_i\ ||\ y) \oplus M_8 \oplus Rs$, $M_{11}= h\ (y\ ||\ Rs)$, $M_{12}= h\ (M_6||\ M_9||\ y||\ Rs)$ où Rs est un nombre aléatoire choisi par le serveur qui envoie le message (M_{10}, M_{11}, M_{12}) au client.
- Étape 3: Après avoir reçu le message dans l'étape 2, le client calcule $M_{13}= h\ (RPW_i\ ||\ SID||\ y) \oplus M_3 \oplus M_{10}$ et calcule $M_{14}= h\ (y||\ M_{13})$ et vérifie $M_{14} =M_{11}$ Si il se tient alors le client va davantage pour la vérification de l'égalité $M_{12}=h(M_1||RPW_i||y||M_{13})$. Si cette vérification est valable, la validité de serveur est authentifiée par le client. Sinon, le client termine la procédure.
- Étape 4: Après la phase d'authentification mutuelle, l'utilisateur calcule la clé de session *SK* partagé avec le serveur comme $SK= h\ (RPW_i||\ M_3||\ M_{13}||SID_i)$. Le serveur calcule également la même clé de session *SK* partagée avec l'utilisateur comme $SK=h\ (M_9||\ M_8||\ Rs||SID_i)$.

La notation Alice-Bob proposée est la suivante :

1: C → S : IDi, M_2, M_3, M_4, M_5
 % tel que M_2=H (IDi, Xs) ⊕ Nc,
 M_3=H (Y, Nc),
 M_4= H (N, PWi) ⊕ H (Y, Nc),
 M_5= H (H (IDi, Xs) ⊕ Nc, H (Y, Nc), H (N, PWi) ⊕ H (Y, Nc))

2: S → C : M_{10}, M_{11}, M_{12}
 % tel que M_{10}=H (H (N, PWi), SIDi, Y)
 M_{11}=H (Y, Ns)
 M_{12}= H (H (IDi, Xs), H (N, PWi), Y, Ns)

La spécification HLPSL complète de ce protocole est donnée dans l'annexe H.

III.5.2.2. Les résultats de la vérification :

Les Scénario 1,..,3:

Après la vérification de protocole LNMW avec tous les scénarios par les outils AVISPA. La Figure III.22 illustre le résultat de ce protocole. Ce résultat signifie en clair qu'il n y a aucune attaque. Donc ce protocole est sûr.

```
AVISPA Tool Summary
OFMC    : SAFE
CL-AtSe : SAFE
SATMC   : INCONCLUSIVE
TA4SP   : INCONCLUSIVE
Refer to individual tools output for details
```

Figure III.22: le résultat de vérification du protocole LNMW

III.6. Conclusion

Dans ce chapitre, nous avons spécifié et vérifié les protocoles de sécurité avec la plateforme AVISPA. Pour chaque protocole étudié, nous avons présenté la description formelle sous forme Alice-Bob, son spécification avec le langage HLPSL, et on a donné les résultats obtenus de la vérification.

Dans le prochain chapitre, Nous allons faire une comparaison entre les protocoles de sécurité étudiés de trois axes: l'analyse de résultats de la vérification avec AVISPA, la comparaison entre des travaux existants et les résultats expérimentaux, et la complexité du tag (ou carte).

Chapitre IV

Etude comparative

Sommaire

IV.1. Introduction

Dans ce chapitre, nous allons résumer les résultats des vérifications automatiques en précisant les types d'attaque pour chaque trace d'attaque détectée dans les protocoles étudiés. Ensuit nous allons faire une comparaison entre des travaux existants et les résultats d'AVISPA. Puis nous allons présenter une étude complémentaire consiste à comparer la complexité d'implémentation des primitives cryptographiques exigées dans ces protocoles sur les tags et carte à puce.

IV.2. Analyse des résultats

Dans cette section, on propose une analyse des résultats de l'aspect sécurité des protocoles. Cette analyse est basée sur les vérifications automatiques de la validation de la propriété d'authentification et de confidentialité de chaque protocole étudié, on résume les résultats d'expérimentation des protocoles de système RFID dans la table IV.1 et les résultats d'expérimentation des protocoles de système de carte à puce dans la table IV.2

IV.2.1. analyse de résultats des protocoles de système RFID

Pour la confidentialité des informations échangées, il est à considérer qu'elle sécrète pendant la transmission entre le tag et le lecteur. Pour les protocoles de système RFID, les informations confidentielles sont cryptées par une primitive cryptographique efficace telle que la fonction de hachage.

Protocole	Scénario de la vérification	Confidentialité		Authentification	
		Clé	Résultat	Tag	Lecteur
KN	1,2	ID, K	S	S	S
	3	ID, K	S	S	A
OTYT	1	K	S	S	S
	2	K	S	S	A
	3	K	S	A	S
CD	1,3	K, N_i, ID_R, EPC_T	S	S	A
	2	K, N_i, ID_R, EPC_T	S	S	A
YPL	1	K, K_1, K_2	S	S	S
	2	K, K_1, K_2	S	S	A
	3	K, K_1, K_2	S	A	S
KCL	1	ID, K	S	A	S
	3,2	ID, K	S	A	S
SR	1	ID_r, ID_t, K	S	S	A
	3,2	ID_r, ID_t, K	S	S	A

Table IV.1 : Expérimentation sur la plateforme AVISPA (système RFID)

Pour l'authentification du lecteur, L'attaque détectée dans le protocole KN est appelée attaque d'usurpation d'identité. Le but de l'intrus est de faire un relais sur les deux premiers messages, mais le dernier est inutile parce que l'objectif du dernier message est l'authentification du tag. Donc, l'intrus a réussi son attaque à la fin du deuxième message. L'attaque détectée dans le protocole CD est appelée attaque d'usurpation. Concernant l'attaque de protocole SR est de type "*man-in-the middle*" entre le tag légitime et le lecteur légitime, l'adversaire peut personnifier comme lecteur légitime et avoir l'information du tag, il peut aussi personnifier comme le tag légitime répondant au lecteur, donc, l'adversaire considère les deux cotés comme étant les parties légitimes. On détecte aussi des attaques de type "*Attaque par rejeu*" dans les protocoles OTYT, CD, YPL, avec le deuxième scénario.

Pour l'authentification du Tag, Les attaques détectées dans les protocoles OTYT et YPL avec le troisième scénario est appelé attaque d'usurpation d'identité. Dans ce cas l'intrus est retransmis les messages des agents honnêtes sans faite des modifications ou des destructions.

L'attaque détectée dans le protocole KCL pour tous les scénarios est de type passif, tel que l'intrus ne peut pas faire des modifications sur les messages mais de les retransmettre seulement.

IV.2.2. analyse de résultats des protocoles de système de carte à puce

Pour la confidentialité des informations privées (comme ID_i ou Xs…etc), elle est sécrète pendant la transmission entre le serveur et le client dans les deux protocoles de carte à puce étudiés. Pour ces protocoles, les informations confidentielles sont cryptées par une primitive cryptographique efficace fonction hachage.

Protocole	Scénario de la vérification	Confidentialité		Authentification	
		Clé	résultat	Serveur	Client
LH	1, 2,3	ID_i, Xs	S	A	S
LNMW	1, 2,3	ID_i,SID_i, PW_i,Xs	S	S	S

Table IV.2 : Expérimentation sur la plateforme AVISPA (système de carte à puce)

Pour l'authentification du serveur, L'attaque détectée dans le protocole LH est appelée attaque de l'usurpation identité. Le but de cette attaque est présenté dans la section IV.2.1.

Pour l'authentification du client, les outils AVISPA ne détectent pas des traces d'attaque, on peut ainsi déduire que les protocoles LH et LNMW sont *sûrs* pour cette propriété.

IV.3. Travaux existants

Dans cette section, on présente des travaux existants qui s'étudient les protocoles vérifiés.

IV.3.1 Protocole KN

Dans la référence [39], Les propriétés d'authentification et de non-traçabilité de protocole KN s'appuie sur le secret de la clé partagée. Dans certains cas, la révélation des parties d'une clé secrète peut déjà être assez de tracer le tag. Cela permetà un attaquant d'usurper l'identité de tag à un lecteur et casser l'authentification. Ce protocole n'utilise pas des primitives cryptographiques classiques, mais il s'appuya sur les opérateurs simples comme l'addition modulaire.

L'attaquant met en place des équations qui impliquent les termes du secret sur des quels le protocole dépend. Ces équations peuvent être obtenues par l'observation de plusieurs

exécutions de protocole. Alternativement, L'adversaire peut modifier des parties de messages et d'observer les réponses générées par le lecteur ou le tag. Le rétablissement du secret partagé casse tout de suite la non-traçabilité et l'authentification.

IV.3.2. Protocole OTYT

Dans [40], les auteurs ont fait une analyse de sécurité de base de protocole OTYT, l'analyse se focalise sur trois propriétés de sécurités : l'authentification, non-traçabilité et désynchronisation. Pour toutes ces propriétés, les auteurs ont découvert des attaques.

Pour authentification de lecteur, le tag ne connaît pas la nouvelle clé k1, le tag n'est pas capable de vérifier si le troisième message est effectivement k⊕k1. Puisque aucune vérification ne peut être effectuée par le tag, l'adversaire peut envoyer un message aléatoire r au tag qui va provoquer le tag pour remplacer k par k ⊕ r.

Pour la non-traçabilité, un attaquant qui observe l'exécution de protocole obtient un triplé (r, h (k ⊕r), k ⊕k1). Il peut maintenant défier un tag avec r⊕k⊕k1 lui donnant la même réponse qu'il a déjà observé, à condition que le tag est le même que celui qui a été espionné précédemment. L'attaque est représentée dans la figure IV.1.

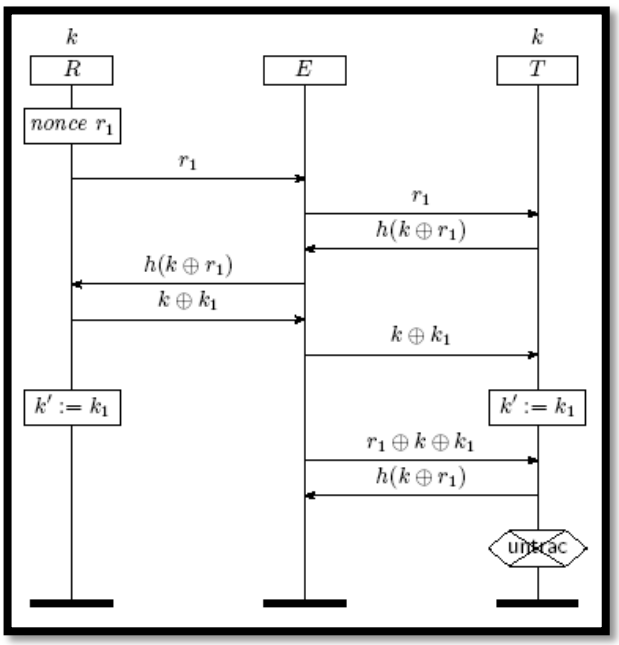

Figure IV.1: Attaque sur non-traçabilité sur le protocole OTYT [40]

IV.3.3. Protocole CD

Dans le papier [33], les auteurs est identifié des vulnérabilités dans le protocole d'authentification mutuelle de Chen et Deng. Le protocole ne révèle être non sécurisée de deux manières différentes, avec un adversaire ayant la capacité de faire passer à la foi le tag et le lecteur. Par conséquent, ce protocole ne peut pas garantir à ses objectifs déclarés de l'authentification sécurisée. Alor le protocole Chen et Deng n'est pas sûr.

Dans le papier [41], les auteurs montrent qu'un attaquant peut usurper l'identité d'un tag ou d'un lecteur, et la non-traçabilité n'est pas garantie étant possible et même facile d'associer un tag avec ses futures réponses.

IV.3.4. Le protocole YPL

En [40], un attaquant qui observe une session de communication du protocole obtient les messages r1, h (k1⊕k ⊕r1), h (k2). Le lecteur et le tag mettre à jour leurs secrets. L'attaquant peut reconnaître le tag en le défiant avec r1 ⊕ h (K2) sur lequel le tag précédemment observée répondra avec h(K1⊕r1⊕k). Les auteurs montrent que le protocole YPL ne fournit pas de non-traçabilité. Figure IV.2 représente la trace d''attaque.

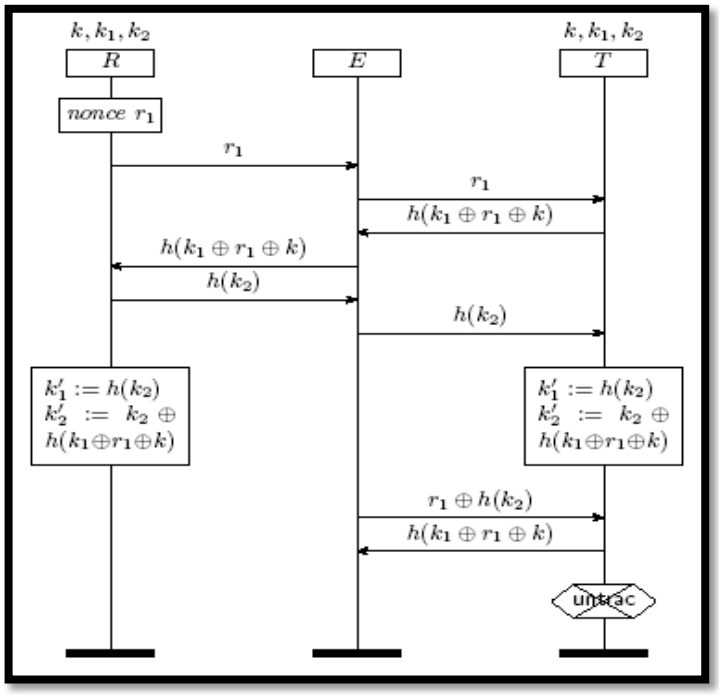

Figure IV.2 : Attaque sur la non-traçabilité sur le protocole YPL [40]

IV.3.5. Protocole KCL

Dans le papier [42], l'adversaire défie le tag deux fois avec le même nonce. Il peut calculer alors le XOR du deux parties ID⊕r2 et h (r1, k) ⊕r2 des réponses, l'adversaire obtient alors deux fois ID ⊕ h (r1, k), si et seulement si était deux fois le même tag qu'il a défiée. L'attaque est représentée dans la figure IV.3.

Dans Le protocole KCL, le résultat atteint que ce protocole garantit l'authentification et la confidentialité, mais n'est pas pour la propriété de la non-traçabilité.

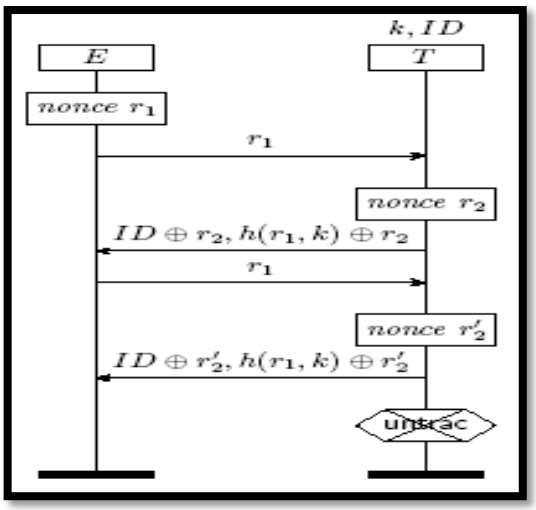

Figure IV.3: L'attaque de non-traçabilité sur le protocole KCL [40]

IV.3.6. Protocole LH

Dans le papier [43], les auteurs montrent que le protocole LH à quelque faiblesses de sécurité, c'est-à-dire le protocole de Li et Hwang ne fournit pas d'authentification correcte et il ne peut pas résister aux attaques de « main in the middle ». Si un attaquant contrôle le canal non sécurisé, il peut facilement fabriquer des messages pour passer l'authentification de l'utilisateur ou de serveur. En plus l'attaquant malveillant peut se faire passer à l'utilisateur pour tromper le serveur et peut usurper l'identité du serveur pour tricher l'utilisateur sans savoir des informations secrètes.

IV.4. Comparaison entre les résultats obtenus et les travaux existants :

Dans cette section, on a présenté un aperçu global sur les travaux existants au domaine de sécurité des protocoles des systèmes RFID, et des protocoles des systèmes carte à puce. Ces travaux réalisés particulièrement pour les protocoles de sécurité ont été étudiés dans le chapitre III.

Dans la table IV.3, on va donner quelque précisons sur les protocoles objet de notre étude en terme références d'analyse, les problèmes de sécurité, ainsi que les attaques éventuelles détectées par la plateforme AVISPA.

Protocole	Référence d'analyse	Problèmes de sécurité	Diagnostic d'AVISPA
KN	[39]	Authentification du lecteur Authentification du tag Non-traçabilité	Authentification du lecteur
OTYT	[40]	Authentification du lecteur Non-traçabilité	Authentification du tag Attaque par rejeu
CD	[33] [41]	Authentification du lecteur Authentification du tag	Authentification du lecteur Authentification du tag Attaque par rejeu
YPL	[40]	Non-traçabilité	Authentification du tag Attaque par rejeu
KCL	[42]	Non-traçabilité	Authentification du tag
SR			Authentification du lecteur
LH	[43]	Authentification du serveur	Authentification du serveur
LNMW			Sûr

Table IV.3: Comparaison entre les résultats obtenus et les travaux existants

A partir de cette table, on conclue que *le protocole LNMW* est le seul protocole qui sûr pour les propriétés de sécurité classiques telle que l'authentification, la confidentialité (secret), et pour les propriétés particulières telle que la non-traçabilité et la résistance à la désynchronisation.

IV.5. Complexité du tag / carte à puce :

La complexité de toutes les implémentations de primitives cryptographiques exigées devrait être la plus faible possible pour maintenir le nombre requis des portes logiques et celles-ci, le coût du tag est faible. La table IV.4 illustre les primitives cryptographiques exigées dans les phases login et l'authentification sur la carte à puce pour les protocoles

biométriques, ainsi que les primitives cryptographiques exigées sur le tag dans les protocoles RFID.

Les grandes remarques tirées sont :

- La plupart des protocoles étudiés, sauf un protocole (CD), exigeant un nombre important d'opérations pour la fonction de hachage, par ce que elle est efficace de côté sécurité et son coût est faible par rapport le chiffrement symétrique ou à clé publique.

Primitive / Protocole		Fonction Hash *(H)*	XOR *(X)*	PRNG *(R)*	Cryptage Symétrique *(C)*	Primitive non cryptographique	Nombre d'opérations
RFID	KN	x	x	x		Plus, Mod	3H+2R+2X
	OTYT	x	x				1H+2X
	CD		x	x		CRC	1R+5X
	YPL	x	x				2H+2X
	KCL	x	x	x			1H+1R+2X
	SR	x	x		x		3H+4X+1C
carte à puce	LH	x	x	x			4H+1R+2X
	LNMW	x	x	x			7H+1R+3X

Table IV.4 : Les primitives exigées dans la carte à puce et le tag

- Les protocoles de CD et KN utilisent des primitives non cryptographiques parce que ses coûts d'implémentation sont très bas et pour éliminer des failles de sécurité.
- La majorité des protocoles étudiés exigeant un générateur des nombres pseudo aléatoire (PRNG) qui sert à générer des nonces sauf les protocoles (OTYT, YPL, SR). La mise en œuvre de générateur des nombres pseudo aléatoire peut appeler une fonction de hachage avec clé HMAC.
- La plupart des protocoles étudiés, n'utilisent pas le cryptage symétrique sauf un protocole (SR) par ce que son mise en œuvre est haut.
- Tous les protocoles étudiés, exigent Les primitives «ou exclusif » (dénoté par *xor* et le symbole $\oplus$). Ces primitives sont utilisées dans nombreux importants protocoles cryptographiques et dans des domaines différents.

IV.6. Conclusion

Dans ce chapitre, nous avons traité le problème de la vérification des protocoles d'authentification en utilisant des outils AVISPA. Cette vérification est importante pour assurer les propriétés de confidentialité et d'authentification dans les protocoles de sécurité. La prise de décision pour sélectionner un protocole dépend de: la complexité, la performance, et la sécurité. En autre terme, cette décision dépend de domaine auquel il est associé.

CONCLUSION ET PERSPECTIVES

Ces dernières années, la vérification des protocoles de sécurité a été un sujet de recherche très actif et devenue nécessaire ou vu des actions malveillantes, de tentative d'intrusion et d'attaque diverses que subissent les réseaux informatique. C'est pourquoi des outils d'analyse comme AVISPA sont des moyens efficaces pour vérifier la robustesse des protocoles des sécurités.

Dans ce mémoire, nous somme intéressés au thème de la validation automatique des protocoles de sécurité, et de l'analyse de ces protocoles en utilisent la plateforme AVISPA et l'outil SPAN.

Tout d'abord, nous avons présenté les concepts fondamentaux de la sécurité et de la cryptographique et les hypothèses pour le recherche des failles dans les protocoles de sécurité, nous avons introduit l'outil de vérification formelle AVISPA et spécifié son architecture logicielle. Ensuit nous avons spécifie par le langage de spécification de haut niveau HLPSL certains protocoles de sécurité (les protocoles des systèmes RFID et les protocoles des systèmes carte à pus) afin de les vérifier par l'outil AVISPA.

Nous avons fait une étude comparative entre les protocoles étudiésdes deux côtés les résultats de la vérification automatique et la complexité des primitives cryptographiques implémentées dans les tags et carte à puce.La conclusion obtenue est : La prise de décision pour sélectionner un protocole qui dépend de: la complexité, la performance et la sécurité. En d'autres termes, cette décision dépend du domaine auquel il est associé.

Perspectives :

Primitives non cryptographiques dans les protocoles : Les nouveaux protocoles des systèmes RFID renoncés à l'utilisation des primitives cryptographiques classiques (e.g. chiffrement symétrique et fonction de hachage) sont remplacés par des opérateurs arithmétiques (somme et soustraction), logiques (et, ou), CRC, et autres, parce que ses coûts d'implémentation sont très bas. Le problème est posé au niveau de la spécification et la vérification où ces opérateurs ne sont pas supportés par les langages de spécification. Il faut donc mettre en place des algorithmes de vérification pour ces opérateurs en mettant en considération ses propriétés

Les propriétés de sécurités spécifiques : La vérification automatique est importante pour assurer les propriétés : secret et authentification dans les systèmes RFID; mais cela ne permettra en aucun cas de confirmer que ces protocoles sont totalement valides, à cause de l'existence de propriétés particulières (e.g. non-traçabilité) difficiles à vérifier d'une manière automatique.

Concernant de la vérification automatique des propriétés spécifiques de sécurité des systèmes RFID, il y a deux possibilités, soit on développe des nouveaux outils, soit on modifie les outils existants (e.g. plateforme AVISPA). Tout ça est relié avec les définitions formelles des ces propriétés.

BIBLIOGRAPHIE

[1] B. Geneviéve. ASPiC: un outil d'analyse symbolique automatisée de protocoles cryptographiques basé sur le modèle de flux d'information. Génie informatique. Montréal. Université de Montréal. 2004.

[2] V. Cortier. Vérification automatique des protocoles cryptographiques. Informatique. Cachan. École Normale Supérieure de Cachan. 2003.

[3] G. Labouret. introduction à la cryptographie. Hervé Schauer Consulants.2001

[4] SecuriteInfo. Les Fonctions de Hachage. http://www.securiteinfo.com. (page consultée le 10 mars 2013).

[5] mrproof.blogspot. Cryptographie : chiffrement et signature.http://mrproof.blogspot.com. (Page consultée le 05 mars 2013).

[6] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," Communications of the ACM, vol. 1(12), pp.993–999, 1978.

[7] W. Diffie and M. Helman. New directions in cryptography. *IEEE Transactions on Information Society*, 22(6):644–654, novembre 1976.

[8] M. Steiner, G. Tsudik, and M. Waidner. "Key agreement in dynamic peer groups," Technical report, Information Sciences Institute, January 1999.

[9] V. Cortier. Analyse des protocoles des cryptographique : des modèles symboliques aux modèles calculatoires. Informatique. Nancy. Université Nancy. 2009.

[10] B. Blanchet. Vérification automatique de protocoles cryptographiques : modèle formel et modèle calculatoire. Informatique. Paris. Université Paris-Dauphine. 2008.

[11] D. Dolev and A. Yao. On the Security of Public-Key Protocols. IEEE Transactions on Information Theory, 2(29), 1983.

[12] T. Okba. Uml Et Model Checking. Informatique. BATNA. Université El Hadj Lakhdar. 2009.

[13] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In Proceedings of "DISCEX" 2000, pages 237–250. IEEE Comp. Society Press, 2000.

[14] 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, Passau, Germany, march 1996. Springer-Verlag, Berlin Germany. Also in Software Concepts and Tools, 17:93-102, 1996.

[15] P. Lafourcade. Vérification de protocoles cryptographiques en présence de théories équationnelles. Informatique. CACHAN. École Doctorale de Sciences Pratiques de l'ENS Cachan. 2006.

[16] G. Lowe. Casper: A compiler for the analysis of security protocols. In Proceedings of 10th Computer Security Foundations Workshop (CSFW'97), Rockport, Massachusetts, USA, 1997. IEEE Computer Society Press. Also in Journal of Computer Security, Volume 6, pages 53-84, 1998.

[17] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In Logic for Programming and Automated Reasoning (LPAR'00), volume 1955 of Lecture Notes in Computer Science, November 2000.

[18] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò and L. Vigneron, "The AVISS Security Protocol Analysis Tool," Proc. of the 14th International Conference of Computer Aided Verification *(CAV'02), LNCS* 2404, Springer, pp. 349–354, 2002.

[19] V.CORTIER, MILLEN J., RUESS H., « Proving Secrecy is easy enough », Proc. of the 14th Computer Security Foundations Workshop (CSFW'01), IEEE Computer Society Press, 2001, p. 97-108

[20] B. Blanchet, "An effecient cryptographic protocol verifier based on prolog rules," In Proceedings of CSFW'01, IEEE Computer Society Press, p.p.82-96, 2001.

[21] AVISPA team, "HLPSL Tutorial The Beginner's Guide to Modelling and Analysing Internet Security Protocols," Technical report, AVISPA project, 2006.

[22] AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. Disponible sur demande en http://www.avispa-project.org,2003.

[23] A. Armando, D. Basin, Y. Boichut, Y.Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. H_eam, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron, "The AVISPA tool for the automated validation of internet security protocols and applications," In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005 LNCS. Springer, Heidelberg, vol.3576, p.p.281–285, 2005.

[24] AVISPA v1.1 User Manual, The AVISPA Team. Disponible sur demande en http://www.avispa-project.org, June 30, 2006

[25] L. Vigano',Automated Security Protocol Analysis With the AVISPA Tool, Electronic Notes in Theoretical Computer Science Vol. 155, pp.61-86, (2006).

[26] AVISPA. Deliverable 2.3: The Intermediate Format. Disponible sur demande en http://www.avispa-project.org , 2003.

[27] Y. Boichut – Approximations pour la vérification automatique de protocoles de sécurité – UFR Sciences ET Techniques, Université de Franche-Comté, 2006

[28] AVISPA. http://avispa-project.org/. (Page consultée le 20 avril 2013)

[29] Y. Glouche, T. Genet and Erwan HOUSSAY. SPAN – a Security Protocol ANimator for AVISPA User Manual. IRISA / Université de Rennes 1, 2008.

[30] D. Harel and P. S. Thiagarajan. Message sequence charts. UML for Real: Design of

Embedded Real-time Systems, 2003

[31] Jeonil Kang and Daehun Nyang. RFID authentication protocol with strong resistance against traceability and denial of service attacks. In Refik Molva, Gene Tsudik, and Dirk Westhoff, editors, European Workshop on Security and Privacy in Ad hoc and Sensor Networks – ESAS'05, volume 3813 of Lecture Notes in Computer Science, pages 164–175, Visegrad, Hungary, July 2005. Springer-Verlag.

[32] Kyosuke Osaka, Tsuyoshi Takagi, Kenichi Yamazaki, and Osamu Takahashi. An efficient and secure RFID security method with ownership transfer. In CIS, pages 778–787, 2006.

[33] G. Kapoor, S. Piramuthu Vulnerabilities in Chen and Deng's RFID mutual authentication and privacy protection protocol, Engineering Applications of Artificial Intelligence, Vol. 24, pp.1300-1302, (2011).

[34] Jeongkyu Yang, Jaemin Park, Hyunrok Lee, Kui Ren, and Kwangjo Kim. Mutual authentication protocol for low-cost RFID. Handout of the Ecrypt Workshop on RFID and Lightweight Crypto, July 2005.

[35] Il Jung Kim, Eun Young Choi, and Dong Hoon Lee. Secure mobile RFID system against privacy and security problems. In SecPerU 2007, 2007.

[36] Mrs. M. Sandhya and Dr.T.R.Rangaswamy, "A Practical Approach for Enhancing Security in Mobile RFID Environment", International Conference on Future Information Technology, Vol. 13, pp.75-80, (2011).

[37] C. T. Li and M. S. Hwang, "An efficient biometrics-based remote user authentication scheme using smart cards," Journal of Network and Computer Applications, Vol. 33, No. 1, pp. 1-5, 2010.

[38] X. Li, J. W. Niu, J. Ma, W. D. Wang. "Cryptanalysis and improvement of a biometrics-based remote user authentication scheme using smart cards," Journal of Network Security & Its Applications, Vol.3, No.2, March 2011.

[39] T. V. Deursen, "Security of RFID protocols. Informatique". Luxembourg, L'université de Luxembourg. 2011.

[40] T. V. Deursen, S. Radomirović. "Attacks on RFID Protocols". Version 1.1, August 6, 2009.

[41] P.Lopeza, C. H. Castrob, M. E. Tapiadorc, C.A. Lubbea, "Cryptanalysis of an EPC Class-1 Generation-2 Standard Compliant Authentication Protocol", University of Technology,), Faculty of Electrical Engineering, Mathematics, and Computer Science 2011

[42] T. V. Deursen, "50 ways to break RFID privacy ". University of Luxembourg.2007

[43] X. Li, J.W. Niu, J. Ma, W. D. Wang, C.L. Liu, "Cryptanalysis and improvement of a biometrics-based remote user authentication scheme using smart cards. Journal of Network and Computer Applications, Vol. 34, pp.73-79, (2011).

Annexe A : Spécification du protocole KN

```
role tag (T,R: agent,ID : text ,K: symmetric_key, H,Plus,Mod :hash_func,
Snd,Rec:channel(dy))
played_by T def=
local State : nat, Nt0,Nt1,Nr,N : text
const sec_k1 : protocol_id
init State := 0
transition
1. State = 0    /\ Rec(start) =|>
   State' := 1  /\ Nt0' := new() /\ Nt1' := new()
                /\ Snd(H(ID.Nt0').H(Nt1'.K).xor(ID,Nt1'))
                /\ witness(T,R,aut_tag,Nt1')
2. State = 1    /\ Rec(H(Nt1.Nr').xor(ID,Nr')) =|>
   State':= 2   /\ Snd(Mod(Plus(Nt1.Nr').exp(2,N)))
                /\ request(T,R,aut_reader,Nr') /\ secret(K,sec_k1,{R,T})
end role

role reader (R,T: agent,ID : text ,K: symmetric_key, H,Plus,Mod : hash_func
,Snd,Rec:channel(dy))
played_by R def=
local State : nat, Nt0,Nt1,Nr,N : text
 const sec_k2 : protocol_id
init State := 0
transition
1. State = 0    /\ Rec(H(ID.Nt0').H(Nt1'.K).xor(ID,Nt1')) =|>
   State':= 1   /\ Nr' := new()
                /\ Snd(H(Nt1'.Nr').xor(ID,Nr'))
                /\ witness(R,T,aut_reader,Nr')
2. State = 1    /\ Rec(Mod(Plus(Nt1.Nr).exp(2,N))) =|>
   State':= 2   /\ request(R,T,aut_tag,Nt1) /\ secret(K,sec_k2,{R,T})
end role

role session(T,R: agent,ID : text ,K: symmetric_key, H,Plus,Mod :
hash_func) def=
 local St,Rt,Sr,Rr : channel(dy)
const aut_reader,aut_tag : protocol_id
composition
   tag(T,R,ID,K,H,Plus,Mod,St,Rt)
/\ reader(R,T,ID,K,H,Plus,Mod,Sr,Rr)
end role

role environment() def=
const t1,t2,r : agent,id: text , k : symmetric_key,
      aut_reader,aut_tag :protocol_id,
      h,plus,mod : hash_func
intruder_knowledge = {t1,t2,r,h,plus,mod}
composition
session(t1,r,id,k,h,plus,mod) /\ session(t2,r,id,k,h,plus,mod)
end role

goal
secrecy_of sec_k1,sec_k2
authentication_on aut_reader
authentication_on aut_tag
end goal

environment()
```

Annexe B : Spécification du protocole OTYT

```
role reader (R,T: agent, K: symmetric_key, H : hash_func,Snd,
Rec:channel(dy))
played_by R def=
local State : nat, Nr, K1 : text
const sec_k : protocol_id
init State := 0
transition
1. State = 0   /\ Rec(start) =|>
   State':= 1  /\ Nr' := new()
               /\ Snd(Nr')
               /\ secret(K,sec_k,{R,T})
2. State = 1   /\ Rec(H(xor(K,Nr))) =|>
   State':= 2  /\ K1' := new()
               /\ request(R,T,aut_tag,Nr)
               /\ witness(R,T,aut_reader,K1')
               /\ Snd(xor(K,K1'))
end role

role tag ( T,R: agent, K: symmetric_key, H : hash_func,Snd,Rec:channel(dy))
played_by T def=
local State : nat,K1, Nr : text
init State := 0
transition
1. State = 0   /\ Rec(Nr') =|>
   State':= 1  /\ Snd(H(xor(K,Nr')))
               /\ witness(T,R,aut_tag,Nr')
2. State = 1   /\ Rec(xor(K,K1')) =|>
   State' :=2  /\ request(T,R,aut_reader,K1')
end role

role session(T,R : agent,K : symmetric_key,H : hash_func) def=
local St,Rt,Sr,Rr : channel(dy)
composition
   tag(T,R,K,H,St,Rt)
/\ reader(R,T,K,H,Sr,Rr)
end role

role environment() def=
const t1,t2,r : agent,
      k : symmetric_key,
      h : hash_func,
aut_reader,aut_tag : protocol_id
intruder_knowledge = {t1,t2,r,h}
composition
session(t1,r,k,h) /\ session(t2,r,k,h)
end role

goal
secrecy_of sec_k
% tag authenticates reader on aut_reader
authentication_on aut_tag
authentication_on aut_reader

end goal

environment()
```

Annexe C : Spécification du protocole CD

```
role reader (R,T: agent,IDr,EPCt : text ,Ki,Ni: symmetric_key, CRC:
hash_func,Snd,Rec:channel(dy)) played_by R def=
local State : nat,Nt,Nr : text,
Mreq, Mresp: message
const sec_k : protocol_id
init State := 0
transition
1. State = 0    /\ Rec(start) =|>
   State':= 1   /\ Nr' := new()
                /\ Snd(Mreq.Nr'.CRC(xor(Ni,Nr')))
                /\ witness(R,T,aut_reader,Nr')
2. State =      /\ Rec(Nt'.xor(xor(Ki,EPCt),Nt').CRC(xor(xor(Nt',Ni),
xor(xor(Ki,EPCt),Nt')))) =|>
   State':= 2   /\ Snd(Mresp)
                /\ request(R,T,aut_tag,Nt') /\ secret(Ki,sec_k,{R,T})
end role

role tag (T,R: agent,IDr,EPCt : text ,Ki,Ni: symmetric_key, CRC:
hash_func,Snd,Rec:channel(dy)) played_by T def=
local State : nat,Nt,Nr : text,
Mreq, Mresp: message
 const sec_k1 : protocol_id
init State := 0
transition
1. State = 0    /\ Rec(Mreq.Nr'.CRC(xor(Ni,Nr'))) =|>
   State':= 1   /\ Nt' := new()
                /\ Snd(Nt'.xor(xor(Ki,EPCt),Nt').CRC(xor(xor(Nt',Ni),
xor(xor(Ki,EPCt),Nt'))))
                /\ witness(T,R,aut_tag,Nt')
2. State = 1    /\ Rec(Mresp) =|>
   State':= 2   /\ request(T,R,aut_reader,Nr)/\ secret(Ki,sec_k1,{T,R})
end role

role session(T,R: agent,IDr,EPCt : text ,Ki,Ni: symmetric_key, CRC:
hash_func) def=
 local St,Rt,Sr,Rr : channel(dy)
const aut_reader,aut_tag : protocol_id
composition
   tag(T,R,IDr,EPCt,Ki,Ni,CRC,St,Rt)/\ reader(R,T,IDr,EPCt,Ki,Ni,CRC,Sr,Rr)
end role

role environment() def=
const t1,t2,r : agent, idr,epct:text,ki,ni: symmetric_key, crc : hash_func,
      aut_reader,aut_tag :protocol_id
intruder_knowledge = {t1,t2,r,crc}
composition
session(t1,r,idr,epct,ki,ni,crc) /\ session(t2,r,idr,epct,ki,ni,crc)
end role

goal
secrecy_of sec_k,sec_k1
authentication_on aut_reader
authentication_on aut_tag
end goal

environment()
```

Annexe D : Spécification du protocole YPL

```
role reader (R,T: agent,K,K1,K2: symmetric_key, H : hash_func,
Snd,Rec:channel(dy))
played_by R def=
local State : nat, Nr : text
 const sec_k : protocol_id
init State := 0
transition
1. State = 0    /\ Rec(start) =|>
   State':= 1   /\ Nr' := new()
                /\ Snd(Nr')
2. State = 1    /\ Rec(H(xor(xor(K1,Nr),K))) =|>
   State':= 2   /\ Snd(H(K2))
                /\ request(R,T,aut_tag,Nr)
                /\ witness(R,T,aut_reader,H(K2))
                /\ secret(K,sec_k,{R,T})
end role

role tag (T,R: agent,K,K1,K2: symmetric_key, H :
hash_func,Snd,Rec:channel(dy))
played_by T def=
local State : nat, Nr : text
init State := 0
transition
1. State = 0     /\ Rec(Nr') =|>
   State' := 1   /\ Snd(H(xor(xor(K1,Nr'),K)))
                 /\ witness(T,R,aut_tag,Nr')
2. State = 1     /\ Rec(H(K2)) =|>
   State':= 2    /\ request(T,R,aut_reader,H(K2))
end role

role session(R,T: agent,K,K1,K2: symmetric_key, H : hash_func) def=
 local St,Rt,Sr,Rr : channel(dy)
const aut_reader : protocol_id
composition
   tag(T,R,K,K1,K2,H,St,Rt)
/\ reader(R,T,K,K1,K2,H,Sr,Rr)
end role

role environment() def=
const t1,t2,r : agent,
      k,k1,k2: symmetric_key,
      aut_reader,aut_tag :protocol_id,
      h : hash_func
intruder_knowledge = {t1,t2,r,h}
composition
session(t1,r,k,k1,k2,h)  /\ session(t2,r,k,k1,k2,h)
end role

goal
secrecy_of sec_k
authentication_on aut_reader
authentication_on aut_tag
end goal

environment()
```

Annexe E : Spécification du protocole KCL

```
role reader (R,T: agent, ID :text, K: symmetric_key, H : hash_func,
Snd,Rec:channel(dy))
played_by R def=
local State : nat, Nr, Nt : text
init State := 0
transition
1. State = 0   /\ Rec(start) =|>
   State':= 1  /\ Nr' := new()
               /\ Snd(Nr')
2. State = 1   /\ Rec(xor(ID,Nt').xor(H(Nr.K),Nt')) =|>
   State':= 2  /\ request(R,T,aut_tag,Nt')
end role

role tag (T,R: agent, ID :text, K: symmetric_key, H : hash_func,
Snd,Rec:channel(dy))
played_by T def=
local State : nat, Nr, Nt : text
const sec_k : protocol_id
init State := 0
transition
1. State = 0   /\ Rec(Nr') =|>
   State':= 1  /\ Nt' := new()
               /\ Snd(xor(ID,Nt').xor(H(Nr'.K),Nt'))
               /\ witness(T,R,aut_tag,Nt')
               /\ secret(K,sec_k,{T,R})
end role

role session(T,R: agent, ID :text, K: symmetric_key, H : hash_func ) def=
local St,Rt,Sr,Rr : channel(dy)
composition
   tag(T,R,ID,K,H,St,Rt)
/\ reader(R,T,ID,K,H,Sr,Rr)
end role

role environment() def=
const t1,t2, r : agent,
      id : text,
      k : symmetric_key ,
      h : hash_func ,
     aut_reader, aut_tag :protocol_id
intruder_knowledge = {t1,t2,r,h}
composition
session(t1,r,id,k,h) /\ session(t2,r,id,k,h)
end role

goal
secrecy_of sec_k
authentication_on aut_tag
end goal

environment()
```

Annexe F : Spécification du protocole SR

```
role reader (R, T, S: agent,IDr,IDt :text,Ki : symmetric_key,H : hash_func,
Snd, Rcv: channel(dy))
played_by R def=
local State : nat,Nr,Nt,D : text,
Ks : symmetric_key
const sec_k : protocol_id
init State := 0
transition

0. State = 0    /\ Rcv(start) =|>
   State':= 1   /\ Nr' := new()
                /\ Snd(Nr')
                /\ witness(R,T,aut_reader,Nr')
1. State = 1    /\ Rcv(xor(H(xor(IDt,Ki)),Nr)) =|>
   State':= 2   /\ Snd(xor(H(xor(IDt,Ki)),Nr).H(IDr).Nr)
2. State = 2    /\ Rcv({xor(D.H(Ki),xor(Nt',Nr))}_Ks) =|>
   State':= 3   /\ Snd(xor(H(Ki),xor(Nt',Nr)))
                /\ request(R,S,aut_server,Nt')
                /\ secret(Ki,sec_k,{R,T,S})
end role

role tag (T, R, S: agent,IDt :text,Ki : symmetric_key,H : hash_func,
Snd, Rcv: channel(dy))
played_by T def=
local State : nat,Nr,Nt,D : text
const sec_k2 : protocol_id
init State := 0
transition

0. State = 0   /\ Rcv(Nr') =|>
   State':= 1  /\ Snd(xor(H(xor(IDt,Ki)),Nr'))
1. State = 1   /\ Rcv(xor(H(Ki),xor(Nt',Nr))) =|>
   State':= 2  /\ request(T,R,aut_reader,Nr)
               /\ secret(Ki,sec_k2,{R,T,S})
end role

role server (S, R, T: agent,IDr,IDt :text,Ki : symmetric_key,H : hash_func,
Snd, Rcv: channel(dy))
played_by S def=
local State : nat,Nr,Nt,D : text,
Ks : symmetric_key
const sec_k3 : protocol_id
init State := 0
transition
0. State = 0    /\ Rcv(xor(H(xor(IDt,Ki)),Nr').H(IDr).Nr') =|>
   State':= 1   /\ Nt' := new()
                /\ Snd({xor(D.H(Ki),xor(Nt',Nr'))}_Ks)
                /\ witness(S,R,aut_server,Nt')
                /\ secret(Ki,sec_k3,{R,T,S})
end role

role session(T, R, S : agent,IDr,IDt :text,Ki : symmetric_key,H :
hash_func)def=
local  St,Rt , Sr,Rr , Ss, Rs : channel (dy)
const aut_reader,aut_server : protocol_id
```

```
composition

   tag (T, R, S,IDt,Ki,H, St,Rt)
/\ reader (R, T, S,IDr,IDt, Ki,H, Sr,Rr )
/\ server(S, R, T,IDr,IDt, Ki,H,Ss, Rs)

end role

role environment() def=
const t,t1,r,s: agent,
      idt,idr: text ,
      ki : symmetric_key,
      h: hash_func,
      aut_reader,aut_server :protocol_id
intruder_knowledge = {t,t2,r,s,h}
composition
session(t,r,s,idr,idt,ki,h) /\ session(t1,r,s,idr,idt,ki,h)
end role

goal
secrecy_of sec_k,sec_k2,sec_k3
authentication_on aut_server
authentication_on aut_reader
end goal

environment()
```

Annexe G : Spécification du protocole LH

```
role client (C,S: agent,IDi :text,Xs : symmetric_key,H : hash_func,Snd,
Rcv: channel(dy))
played_by C def=
local State : nat,Nc,Ns : text
const sec_xs : protocol_id
init State := 0
transition
1. State = 0    /\ Rcv(start) =|>
   State':= 1   /\ Nc' := new()
                /\ Snd(xor(IDi.H(IDi.Xs),Nc'))
                /\ witness(C,S,aut_client,Nc')
2. State = 1    /\ Rcv(H(xor(H(IDi.Xs),Nc)).Nc.xor(H(IDi.Xs),Ns')) =|>
   State':= 2   /\ Snd(H(xor(H(IDi.Xs),Ns')).Ns')
                /\ request(C,S,aut_server,Ns')
                /\secret(Xs,sec_xs,{C,S})
end role

role serveur (S,C: agent,IDi:text,Xs : symmetric_key,H : hash_func,Snd,
Rcv: channel(dy))
played_by S def=
local State : nat, Nc,Ns : text
const sec_xs1 : protocol_id
init State := 0
transition
1. State = 0    /\ Rcv(xor(IDi.H(IDi.Xs),Nc')) =|>
   State':= 1   /\ Ns' := new()
                /\ Snd(H(xor(H(IDi.Xs),Nc')).Nc'.xor(H(IDi.Xs),Ns'))
                /\ witness(S,C,aut_server,Ns')
2. State = 1    /\ Rcv(H(xor(H(IDi.Xs),Ns)).Ns)=|>
   State':= 2   /\ request(S,C,aut_client,Nc)
                /\secret(Xs,sec_xs1,{S,C})
end role

role session (S,C: agent,IDi:text,Xs : symmetric_key,H : hash_func) def=
 local Ss,Rs,Sc,Rc : channel(dy)
const aut_client,aut_server : protocol_id
composition
    serveur(S,C,IDi,Xs,H,Ss,Rs)
/\ client(C,S,IDi,Xs,H,Sc,Rc)
end role

role environment() def=
const s1,s2, c : agent,idi: text, xs : symmetric_key,h : hash_func,
      aut_client,aut_server : protocol_id,
      intruder_knowledge = {s1,s2,c,h}
composition
session(s1,c,idi,xs,h) /\ session(s2,c,idi,xs,h)
end role

goal
secrecy_of sec_xs,sec_xs1
authentication_on aut_client
authentication_on aut_server
end goal

environment()
```

Annexe H : Spécification du protocole LNMW

```
role client (C,S: agent,IDi,SIDi :text,PWi,Xs : symmetric_key,H :
hash_func,Snd, Rcv: channel(dy)) played_by C def=
local State : nat,Nc,Ns,N : text,
Y:text
const sec_xs : protocol_id
init State := 0
transition
1. State = 0    /\ Rcv(start) =|>
   State':= 1   /\ Nc' := new()
                /\ Snd(IDi.xor(H(IDi.Xs),Nc').H(Y.Nc').xor(H(N.PWi),
H(Y.Nc')).H(xor(H(IDi.Xs),Nc').H(Y.Nc').xor(H(N.PWi),H(Y.Nc'))))

2. State = 1    /\ Rcv(H(H(N.PWi).SIDi.Y).H(Y.Ns').H(H(IDi.Xs).
H(N.PWi).Y.Ns')) =|>
   State':= 2   /\ request(C,S,aut_server,Ns')/\secret(Xs,sec_xs,{C,S})
end role

role serveur (S,C: agent,IDi,SIDi :text,PWi,Xs : symmetric_key,H :
hash_func,Snd, Rcv: channel(dy)) played_by S def=
local State : nat,Nc,Ns,N : text,
Y:text
const sec_xs1 : protocol_id
transition
1. State = 0    /\ Rcv(IDi.xor(H(IDi.Xs),Nc').H(Y.Nc').xor(H(N.PWi),
H(Y.Nc')).H(xor(H(IDi.Xs),Nc').H(Y.Nc').xor(H(N.PWi),H(Y.Nc')))) =|>
   State':= 1   /\ Ns' := new()
                /\ Snd(H(H(N.PWi).SIDi.Y).H(Y.Ns').H(H(IDi.Xs).H(N.PWi).
Y.Ns'))
                /\ witness(S,C,aut_server,Ns')/\secret(Xs,sec_xs1,{S,C})
end role

role session (S,C: agent,IDi,SIDi :text,PWi,Xs : symmetric_key,H :
hash_func) def=
local Ss,Rs,Sc,Rc : channel(dy)
const aut_server : protocol_id
composition
    serveur(S,C,IDi,SIDi,Xs,PWi,H,Ss,Rs)
/\ client(C,S,IDi,SIDi,Xs,PWi,H,Sc,Rc)
end role

role environment() def=
const s1,s2, c : agent,idi,sidi: text, pwi,xs : symmetric_key,
      aut_server : protocol_id,
      h : hash_func
intruder_knowledge = {s1,s2,c,h}
composition
      session(s1,c,idi,sidi,xs,pwi,h) /\ session(s2,c,idi,sidi,xs,pwi,h)
end role

goal
secrecy_of sec_xs,sec_xs1
authentication_on aut_server
end goal

environment()
```

الملخص:

عملنا يتعلق بالتحقق الآلي المبني على الصيغ لبروتوكولات التشفير. في هذه المذكرة، نقدم نمذجة بلغة HLPSL ونستخدم أرضية AVISPA للتحقق من سلامة فئتين من البروتوكولات الأمنية : بروتوكولات أنظمة التعرف بالترددات و بروتوكولات أنظمة البطاقات الذكية. الخصائص الأمنية التي تم تحليلها هي: السرية و المصادقة. في الأخير أجرينا مقارنة ما بين مختلف البروتوكولات المدروسة من ناحية درجة التعقيد في استعمال الأدوات التشفيرية والجبرية المستعملة في البروتوكولات.

الكلمات المفتاحية: البروتوكول الأمني، الخصائص الأمنية، التحقق الآلي، أرضية AVISPA ، لغة HLPSL

Résumé :

Notre travail s'article sur la vérification formelle automatique des protocoles cryptographiques. Dans ce mémoire, nous présentons une modélisation en langage HLPSL et nous utilisons la plate-forme AVISPA pour vérifier deux catégories des protocoles de sécurité : les protocoles des systèmes RFID et les protocoles des systèmes de carte à puce. Nous vérifions les propriétés de sécurités suivantes: la confidentialité des données secrètes et l'authentification des entités du système. Enfin nous présentons une étude comparative entre les différents protocoles étudies en terme de complexité d'implémentation des primitives cryptographique et algébrique.

Mots-clés : Protocole de sécurité, propriétés de sécurité, vérification automatique, plateforme AVISPA, langage HLPSL.

Abstract:

Our work focuses on the automatic formal verification of cryptographic protocols. In this paper, we present a modeling language in HLPSL and we using the AVISPA platform for verification of two categories of security protocols: the protocols of RFID systems and the protocols of smart card systems. We verify the following properties: the confidentiality of secret data and authentication of the entities of the system. Finally we present a comparative study between the different protocols studied in term complexity of implementation of cryptographic primitives and algebraic.

Key words: security protocol, security properties, automatic verification, AVISPA platform, HLPSL language.