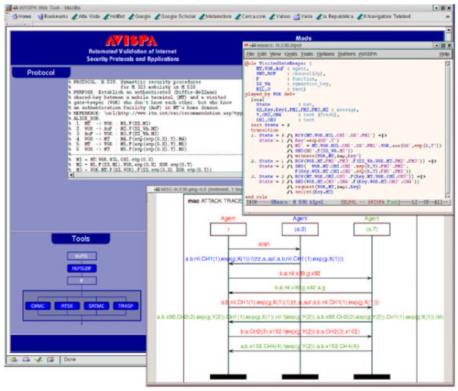
hierie des Protocoles et Logiciels Sécurisés

Conception et Vérification de Protocoles Cryptographiques

version 1



YACINE CHALLAL

17/09/2015

Table des matières

I

Conception et Vérification de Protocoles Cryptographiques

A. AVISPA: Automated Validation of Internet Security Protocols and Applications

Contexte et motivation

Les systèmes d'information sont de de plus en plus distribués d'où la nécessité de développer des protocoles de communication. Un problème majeur dans la conception de protocoles est les erreurs de sécurité :

- Coût : une erreur de sécurité peut causer des millions $ de perte
- Temps: le déploiement de protocoles est retardé
- Confiance : difficulté d'attirer la confiance des clients potentiels des applications développées

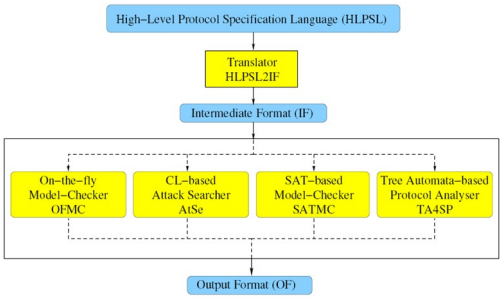
Valider et analyser rigoureusement les protocoles de sécurité en développement dépassent les capacités humaines, vu leur nombre et ampleur. Pour accélérer le développement de nouveaux protocoles de sécurité et améliorer leur niveau de sécurité, il est nécessaire de disposer d'outils qui supportent l'analyse rigoureuse des protocoles cryptographiques, en établissant leur correction, ou en démontrant leur failles. Ces outils doivent être complètement automatisés, robustes, expressifs, et facilement utilisable.

Objectifs du projet AVISPA

- Définition d'un langage de haut niveau pour la spécification de protocoles cryptographiques: HLPSL
- Développement de techniques d'analyse de protocoles cryptographiques
- Développer un outil (AVISPA) en se basant sur ces techniques
- Développement d'une bibliothèque de spécifications de protocoles avec HLPSL / CAS+, et les analyser avec l'outil AVISPA

Composants de AVISPA

La figure suivante illustre les composants de AVISPA :



Architecture de AVISPA

L'outil supporte quatre moteurs de vérifications dis "backends" :

- The On-the-fly Model-Checker (OFMC) : Utilise plusieurs techniques symboliques pour explorer l'espace d'états à la demande.
- CL-AtSe (Constraint-Logic-based Attack Searcher) : Applique la résolution de contraintes avec heuristiques de simplification et techniques d'élimination de redondances
- The SAT-based Model-Checker (SATMC) : Construit une formule propositionnelle qui encode toutes les attaques possibles (de longueure limitée) sur le protocole et soumet le résultat à un solveur
- SAT TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols) : Estime les connaissances de l'intrus en utilisant un langage d'arbre régulier avec réécriture pour produire des sous et sur-approximations

Complément : Environnement graphique

AVISPA offre également un environnement graphique de mise au point des protocoles comme illustré sur la figure suivante :

AVISPA : Environnement Graphique

Fondamental : Modèle d'intrus Dolev-Yao

- L'intrus a le contrôle complet du réseau, c'est le réseau. Tous les messages passent par l'intrus. Les opérations de l'intrus sur les messages sont: Transfert, rejeu, suppression, dcomposition et analyse (si clés connues), modifier synthétiser, envoyer à n'importe qui
- L'intrus ne peut pas avoir accès aux messages chiffrés sans connaissance des clés
- L'intrus peut jouer le rôle de n'importe quel participant
- L'intrus récupère les connaissances d'un participant compromis

B. CAS+ Specification language

Structure d'une spécification

La spécification d'un protocole se fait en 6 parties:

- Déclaration d'identifiants
- Séquence de messages
- Connaissances des agents
- Instances de sessions
- Connaissances de l'intrus
- Objectifs de la vérification

Exemple

Ci-après un exemple de spécification d'un protocole avec Cas+

```
protocol TV; % symmetric key
identifiers
C,D : user;
Ins : number;
K : symmetric_key;

messages
1. D -> C : D,{Ins}K
2. C -> D : C,D,{Ins}K

knowledge
D : C,K;
C : K;

session_instances
 [D:tv,C:scard,K:onekey];

intruder_knowledge
 scard;

goal
 C authenticates D on Ins;
```

Exemple de spécification avec Cas+

Déclaration d'identifiants

Les types d'identifiants possibles sont :

- user
- public_key
- symmetric_key
- function: one-way hash function
- ` : Kd' est la clé privée associée à Kd
- number: abstraction pour n'importe quel type de données (nombre, texte, record, ...)

Exemple : Déclaration d'identifiants

```
protocol TV; % public key
identifiers
C,D : user;
Ins : number;
Kc,Kd : public_key;
```

Exemple de déclaration d'identifiants

Spécification des messages échangés

C'est un ensemble de règles décrivant les messages échangés durant l'exécution du protocole: (i. Si ->i Ri : Mi) (1<=i<=n)

- .i. est le numéro de l'échange

- Si et Ri sont la source et destination du message Mi
- Mi peut contenir des identifiants (déclarés), des « numbers », l'opérateur `, [] (accès à un tableau), () (pour fonction de hashage), « , » (pairing), accolades (chiffrement), ^ exponentiel, # xor
- ->i : représente le type du canal utilisé pour l'envoie du message:
  - -> : Canal Dolev Yao
  - => : canal protégé en lecture et en écriture
  - ~> : canal protégé en écriture

Exemple : Spécification des messages échangés

```
messages
1. D -> C : D,{Ins}Kd'
2. C -> D : C,{Ins}Kc'
```

Exemple de spécification de messages

Spécification des connaissances

Cette section permet la description des connaissances de chaque agent avant le début du protocole : Noms, clés, fonctions, etc.

Le nom de l'agent lui-même est supposé implicitement connu

Remarque

Dans l'exemple précédent, Ins n'est pas déclaré dans les connaissances de D. Il s'agit d'une nouvelle valeur générée à chaque exécution du protocole. Ins n'est pas persistent. Ins est un nonce (oNly once).

Exemple : Déclaration des connaissances

```
knowledge
D : C,Kc,Kd;
C : D,Kc,Kd;
```

Exemple de déclaration des connaissances des agents

Instances de sessions

Décrire les instances de valeurs que peuvent prendre les identifiants persistants. Des sessions peuvent être ouvertes avec un intrus (i)

Exemple : Instanciation de sessions

```
session_instances
 [C:scard,D:tv,Kc:kc,Kd:kd]
 [C:i,D:tv,Kc:ki,Kd:kd];
```

Exemple d'instantiation de sessions

Connaissances de l'intrus

Ensemble de valeurs des instances de sessions (pas d'identifiants persistants)

Exemple : Connaissances de l'intrus

```
intruder_knowledge
 tv,scard,ki,ki',kc,kd;
```

Exemple de spécification de connaissances de l'intrus

Spécification des objectifs de sécurité du protocole (goals)

Propriétés de sécurité que l'on souhaite prouver

- Secrecy : Un identifiant (qui doit rester confidentiel) et un ensemble d'agents qui sont autorisés à avoir accès à l'identifiant confidentiel
- Authentication : Un agent authentifie un autre agent sur un identifiant

Exemple : Spécification des objectifs de sécurité du protocole (goals)

```
goal
 D authenticates C on Ins;
 secrecy_of Ins [C,D];
```

Exemple de spécification d'objectifs de sécurité

Exemple : Spécification complète

La figure suivante illustre deux spécifications complètes de protocoles cryptographiques

```
protocol TV; % symmetric key
identifiers
C,D : user;
Ins : number;
K : symmetric_key;

messages
1. D -> C : D,{Ins}K
2. C -> D : C,D,{Ins}K

knowledge
D : C,K;
C : K;

session_instances
 [D:tv,C:scard,K:onekey];

intruder_knowledge
 scard;

goal
 C authenticates D on Ins;
```

```
protocol TV; % public key
identifiers
C,D : user;
Ins : number;
Kc,Kd : public_key;

messages
1. D -> C : D,{Ins}Kd'
2. C -> D : C,{Ins}Kc'

knowledge
D : C,Kc,Kd;
C : D,Kc,Kd;

session_instances
 [C:scard,D:tv,Kc:kc,Kd:kd]
 [C:i,D:tv,Kc:ki,Kd:kd];

intruder_knowledge
 tv,scard,ki,ki',kc,kd;

goal
 D authenticates C on Ins;
 secrecy_of Ins [C,D];
```

Exemples de spécifications complètes

Méthode : Comment effectuer une vérification d'une spécification cas+ ?

- Translation de CAS+ vers HLPSL : ./alicebob2hlpsl –f output.hlpsl input.cas
- Ouvrir output.hlpsl dans SPAN
- Lancer l'exécution du protocole
- Lancer la vérification du protocole
- Simuler une intrusion

Série d'exercices

II

A. Confidentialité

Considérons le protocole cryptographique suivant :

1. A->B :A,PKa

2.B->A :{M}PKa

Ce protocole devrait assurer la confidentialité du message M.

Question 1

Faites une spécification de ce protocole avec le langage Cas+, puis convertir cette spécification vers le langage HLPS avec la commande :

alicebob2hlpsl -f ex0.hlpsl ex0.cas

Question 2

Faites une vérification du protocole avec SPAN. Que remarquez vous ? Déroulez l'attaque.

Question 3

Apportez les corrections qui s'imposent au protocole puis refaite la vérification.

B. Protocole d'authentification Needham Schroeder

Considérons le protocole d'authentification de Needham Schroeder suivant :

1. A->B :{Na,A}PKb

2.B->A :{Na,Nb}PKa

3.A->B :{Nb}PKb

Question 1

Qu'assure ce protocole comme service de sécurité ?

Question 2

Réalisez une spécification de ce protocole avec le langage Cas+, puis convertir cette spécification vers le langage HLPS avec la commande :

alicebob2hlpsl -f NSPK.hlpsl NSPK.cas

Question 3

Faites une vérification du protocole avec SPAN. Que remarquez vous ? Déroulez l'attaque.

Question 4

Apportez les corrections qui s'imposent au protocole puis refaite la vérification.

C. Echange de clé et confidentialité

Considérons le protocole suivant qui a pour objectif de faire partager une clé de session Kab entre A et B puis l'utiliser pour assurer la confidentialité du message M en le chiffrant avec cette clé Kab.

1.A->B :{Na}PKb

2.B->A :{Na}PKa

3.A->B :{Kab}PKb

4.B->A :{M}Kab

Question 1

Faites une spécification et une vérification du protocole avec SPAN.

Question 2

Décrivez l'attaque contre ce protocole qui remet en considération la confidentialité de M.

Afin d'éviter l'attaque trouvé ci-dessus, on se propose d'ajouter un challenge qui sera envoyé par B pour authentifier A :

1.A->B :{Na}PKb

2.B->A :{Na,Nb}PKa

3.A->B :{Kab,Nb}PKb

4.B->A :{M}Kab

Question 3

Faites une spécification et une vérification du protocole avec SPAN.

Question 4

Décrivez l'attaque contre ce protocole qui remet en considération la confidentialité de M.

Question 5

Proposez une amélioration qui corrige le protocole