RAPPORT TECHNIQUE PROUVÉ

Formalisme de Sortie des Outils de Vérification de Protocoles Cryptographiques PROUVÉ

Auteur : Olivier Bettan, Bertrand Tavernier

Date : December 9, 2005

Rapport PROUVÉ numéro : 8

Version : 1.1

Loria
CNRS UMR 7503,
Campus Scientifique - BP 239
54506 Vandoeuvre-lès-nancy cedex
www.loria.fr

Laboratoire Spécification Vérification
CNRS UMR 8643, ENS Cachan
61, avenue du président-Wilson
94235 Cachan Cedex, France
www.lsv.ens-cachan.fr

Laboratoire Verimag
CNRS UMR 5104,
Univ. Joseph Fourier, INPG
2 av. de Vignate,
38610 Gières, France
www-verimag.imag.fr

Cril Technology
9/11 rue Jeanne Braconnier
92360 Meudon La Foret Cedex, France
www.cril.fr

France Telecom
Div. Recherche et Développement
38, 40 rue du Général Leclerc
92794 Issy Moulineaux Cedex
www.rd.francetelecom.fr

Adresse : Cril Technologie,
Immeuble Avenir - 1ê étage
18 rue Grange Dame Rose
78147 VELIZY CEDEX BP 259
www.criltechnology.com

Résumé : La définition d'un formalisme de sortie pour les outils de vérification de protocoles cryptographiques, propre à PROUVÉ est l'une des étapes obligatoires vers la réalisation de l'interface graphique de visualisation des attaques et protocoles.

PROUVÉ a clairement exprimé sa volonté d'ouverture à la communauté scientifique et amateur dans le domaine de la cryptographie. Il était donc important de rendre ce formalisme suffisamment général et facilement compréhensible pour une utilisation intuitive. Nous nous sommes donc inspirés du langage HMSC (High level Messages Sequences Charts), langage standardisé utilisé pour modéliser les communications sur un réseau, tant pour la syntaxe que pour la représentation graphique, mais sa trop grande richesse nous a obligé à développer notre propre modèle.

Afin de faciliter l'intégration éventuelle de futurs outils, nous avons opté pour un formalisme XML, modulable à volonté et adapté à la plupart des logiciels et langages actuellement sur le marché.

La syntaxe adoptée se devait de satisfaire aux exigences et limitations des trois outils retenus pour PROUVÉ : ISpi, CasRul et Hermès, et sa réalisation à nécéssité une étude approfondie du fonctionnement de chacun d'entre eux.

Finalement, il en a résulté un formalisme construit pour nos outils mais évolutif.

Chapter 1

Introduction

Le projet Prouvé veut mettre à la disposition de tous un ensemble d'outils pour la conception de protocoles cryptographiques, allant de la vérification à la visualisation des attaques trouvées.
Pour pouvoir représenter le résultat de différents logiciels de manière unique et automatique, le besoin d'un format de sortie commun s'est vite imposé à nous.
Nous travaillons actuellement avec trois outils de vérifications principaux : ISpi, CasRul et Hermès, et le formalisme ici présenté a été conçu spécialement pour eux, en conservant cependant une optique généraliste et adaptative afin de le rendre facilement utilisable par d'autres logiciels.
Nous nous sommes pour cela, inspirés du langage HMSC (High level Messages Sequences Charts), langage normalisé créé pour décrire les communications sur un réseau, et du package LaTeX utilisé pour les définir (msctopng et msctops).
Mais une étude approfondie des algorithmes mis en oeuvres par nos outils de vérification a aboutie à la définition de nôtre propre syntaxe, à la fois plus ciblée et plus modulable. Nous l'avons développée en XML, langage largement répendu, pour permettre de connecter simplement de futurs logiciels et faire face aux modifications ultérieures des outils.

Chapter 2

Présentation des Outils

Le langage de sortie PROUVÉ doit en premier lieu tenir compte des spécifications et algorithmes des outils utilisés sur le projet. Ces derniers disposent de méthodes de résolution variées qui font appel aux principaux outils mathématiques actuellement utilisé dans le milieu de la vérification des protocoles cryptographiques. Ainsi, on s'assure d'une meilleure compatibilité avec les logiciels existants ou à venir et d'une plus grande fléxibilité.
Chaque outil a donc apporté son lot de limitations et de contraintes, mais leur étude combinée a servi à la définition de la partie commune et des éléments optionnels.

2.1 ISpi

ISpi a été développé au LSV de l'ENS CACHAN. ISPi utilise la boîte à outils h1, ensemble d'outils de manipulation d'automates (déterministe ou non) et de clauses de Horn (de la classe H_1 définie par Nielson).
Une documentation est disponible en ligne sur le serveur de l'ENS.
ISpi est donc un prouveur automatique de théorèmes pour la classe H_1 (avec garantie de terminaison) ou de clauses générales. Cependant, du fait des sur-approximations effectuées (à l'image des autres outils PROUVÉ) il peut générer des attaques non-réalisables dans un contexte pratique.

ISpi est écrit en Himml et nécessite un traducteur spécifique une fois le parseur PROUVÉ lancé sur les scenarii et assertions (propriétés) souhaités.
Lors de la modification dans son langage interne, un certain nombre d'informations sur les données sont perdues ou dispersées dans différents fichiers de trace, principalement :

- Le nom des instances, variables et messages
- Le type des données
- La méthode de calcul des fakes par l'intrus

En prolongement de ses conventions de nommage et afin de répondre aux besoins de l'interface graphique, Jean Goubault-Larrecq a adapté son outil et enrichi ses fichiers de traces d'exécution.

2.2 CasRul

Un tutoriel pour CasRul est proposé sur une page web dédiée [?]. CasRul est un analyseur de protocole cryptographique, il transforme un tel protocole en règles de réécriture. Puis il utilise daTac un theorem-prover en logique du premier ordre pour la détection automatique des preuves.
Le modèle y est appliqué dynamiquement, l'intrus vérifiant à chaque étape l'état de ses connaissances.

Une particularité de CasRul consiste à ne fournir que des traces d'attaques et pas de preuves si le protocole se révèle sûr. D'autres caractérstiques internes ont fait évoluer le formalisme :

- Les connaissances de l'intrus son modifiées et connues à chaque calcul, ou échange de message

- CasRul construit une table contenant toutes les constantes, données initiales et crées avec leur type. Cette table est disponible en sortie

- Il utilise un mode de communication binaire (deux intervenants uniquement), entre une instance honnête et l'intrus

- Les actions des participants sont détaillées et facilement accessibles en sortie

2.3 Hermès

Un rapport technique EVA ([?]) est disponible dans les délivrables de ce projet.
Hermès calcule une abstraction de la propriété et du protocole extrait du fichier d'entré fournit pas le traducteur EVATRANS. Hermès est basé sur le principe des messages protecteurs, qui conservent le secret. Lors de l'analyse il ajoute au secret les messages qui permettent d'obtenir une attaque et génère des contraintes permettant d'utiliser le protocole sans invalider la propriété de secret souhaitée.

Hermès produit également des traces justifiant l'ajout de chaque nouveau secret et contrainte (chaque trace est une tentative d'attaque) et un arbre décisionnel pour représenter les preuves de sûreté.

Les informations disponibles en sortie de l'outil sont nombreuses et précises, elles ont fortement contribué à la définition du formalisme PROUVÉ, on y trouve entre autre :

- L’instance de rôle effective
- L’émetteur et le déstinataire des messages
- Le calcul est détaillé pour les messages et fakes de l’intrus
- Les connaissances de l’intrus peuvent être précisées à chaque étape

Parfois plusieurs attaques possibles sont obtenues pour un protocole donné, chacune nécessitera alors un fichier de trace distinct.

Chapter 3

Un nouveau format de sortie commun

De l'étude des trois outils précédents, de leurs différences et aspects communs, est née l'idée d'un format de sortie spécifique à PROUVÉ. D'autres projets, dans le même domaine d'expertise, ont adopté un fonctionnement semblable ([**?**]), mais nous souhaitions aller au-delà en permettant que de futurs outils puissent se greffer facilement à PROUVÉ, ou simplement utiliser une partie de notre boîte à outil. Il fallait donc utiliser un langage suffisamment répandu parmi la communauté des développeurs de logiciels de vérification de protocoles cryptographiques, souple et adapté aux structures objets.

3.1 Son rôle

Définir un langage de sortie pour PROUVÉ nous permet de concevoir un ensemble d'outils de traitement de trace desvérificateurs de protocoles cryptographiques, sans se soucier de leur syntaxe ou limitations respectives.La représentation et la simulation des attaques ne dépendent donc plus des outils qui les ont générées.
Actuellement, les renseignements à fournir à l'interface graphique sont la concaténation des informations disponiblesavec nos trois outils de référence. Mais grâce à la structure conditionnelle de notre formalisme, il est d'ors et déjà possible de les restreindre et il sera aisé d'en ajouter au besoin (nouvelles méthodes de vérification, nouvelles données significatives, ...).
Nous espérons ainsi encourager l'utilisation de PROUVÉ par d'autres logiciels, moyennant un travail préalable de génération de trace au format XML présenté ci-après. Ceci permettra à l'utilisateur soucieux de la diversité des techniques de vérification employées, d'intégrer dans ses tests le résultat d'une multitude d'outils.

3.2 Le modèle HMSC

Les Messages Sequences Charts (MSC) forment un langage graphique de trace qui présente la chronologie des séquences des messages entre les composants d'un système et de son environnement. La communication représentée est asynchrone avec les MSC mais peut être modélisée de façon synchrone à l'aide des HMSC (High Messages Sequence Charts).

Il existe d'autres outils qui à partir de la description textuelle (nos fichiers de trace) d'un diagramme MSC génèrent une représentation graphique de celui-ci. Plusieurs solutions ont été envisagées mais celle qui répond le mieux à nos besoin correspond au mode de fonctionnement d'un package LaTeX [?] définit par Victor Bos et Sjouke Mauw. C'est une description dite « par niveau » :

- On fournit d'abord les instances
- un message est donné avec son nom, origine et destination à un « moment » ou « niveau » donné
- De même pour les actions, connaissances et autres événements ...

D'où une définition unique des événements qui correspond mieux à la représentation de nos traces d'exécution.

Cette syntaxe est cependant trop riche et plusieurs aspects ne permettent pas de l'utiliser pour PROUVÉ :

- Tout d'abord on ne peut pas modifier les balises existantes à notre gré, or le modèle formel adopté utilise d'autres attributs et objets (cf. la partie "Le formalisme XML PROUVÉ")
- Le changement de niveau est implicite pour PROUVÉ : un événement quelqu'il soit entraine un nouveau niveau, on préfèrera donc une simple séquence ordonnée
- Les échanges de messages sont instantanés selon les hypothèse de notre projet ...

Ceci nous a conduit à choisir un formalisme proche de celui du package présenté, mais facilement modulable, accessible à tous sans apprentissage fastidieux et disponible pour tous. Le langage XML s'est ainsi imposé à nous.

3.3 Une syntaxe souple avec XML

Le langage XML (eXtensible Markup Language) est à la base de plusieurs langage à balise. Il utilise un balisage structurel à base de tag.
Il permet de définir une grammaire pour un ensemble de documents, à l'aide de fichier DTD (Document Type Definition), ceux-ci sont alors analysables, manipulables ou transformables par des APIs comme SAX et DOM. On peut donc aisément vérifier que le fichier de trace est conforme aux exigences.

L'intérêt principal du langage XML pour PROUVÉ réside dans sa souplesse. Tous les éléments, destinés à la représentation de notre modèle de données, sont modulables. Chaque balise dispose

d'attributs et de paramètres (type prédéfinis ou éléments du modèle) dont la prśence peut être conditionnée et la quantité précisée par des opérateurs du langage.
Ainsi, lorsque certains caractères de notre formalisme ne s'appliquent qu'à un outil donné, il est possible de les spécifier "optionnels" (#IMPLIED) ou "obligatoires" (#REQUIRED). De même, à l'image des expressions régulières , on peut indiquer leur multiplicité.

Toute modification à venir du modèle pourra se répercuter aisément sur le DTD. Et les changements subséquents dans l'interface graphique, qui utilise un langage fortement orienté objet "Java", ne nécessiteront pas de remaniement de structure.

L'XML est très présent dans la communauté des développeurs de logiciels et sert souvent de format d'échange entre outils. Nous pouvons donc espérer une plus grande interopérabilité et une utilisation de nos outils par d'autres projets en cryptographie.

Chapter 4

Le formalisme XML PROUVÉ

Le format de sortie des outils de vérification de protocoles cryptographiques pour PROUVÉ devait se baser sur les possibilités et limitations des outils présentés ci-avant.
Nous avons dû lister puis modéliser l'intégralité des informations utiles à la représentation graphique. Cette étape primordiale s'est révélée délicate, mais a été grandement facilité par l'exemple du projet AVISPA [] et du package LaTeX [**?**].

4.1 Le modèle de données

Voici le modèle de données obtenu, le fichier DTD correspondant se trouve en annexe :

MSC

- Nom (Protocole)
- Type (Preuve, Attaque, Algo)
- INSTANCE+
- MESSAGE+
- CONNAISSANCE+
- ACTION*

INSTANCE

- Nom
- Param_effectif (l'instance de rôle effective)

- Role (intruder, operator)

CONNAISSANCE

- Instance (instance associée)
- DONNEES+ (liste d'objet)

DONNEE

- Id
- Type (message, Integer, pub_key, priv_key, Id, agent, Bool, Algo)
- #PCDATA (la valeur)

MESSAGE

- In
- Out
- Support (normal, codé, blindé) (inutilisé pour l'instant)
- DONNEE

ACTION

- Nom
- METHODE ? (option)
- DONNEE
- Instance (id le l'instance attachée)

METHODE

- Description (la méthode à afficher)
- CALCUL+

CALCUL

- Fonction (le prototype du calcul)
- DONNEE+

Chapter 5

Conclusion

Nous avons construit un formalisme de sortie adapté aux besoins actuels du projet, mais dans une optique évolutive.
Nos outils de vérification de protocoles cryptographiques ont imposé leurs contraintes et ont guidé le développement du modèle de données, mais ils ont surtout mis en évidence leurs disparités. Nous avons donc opté pour une structure et un langage souples et accessibles à tous pour conserver l'aspect open-source de PROUVÉ et faciliter les échanges avec d'autres outils existants.

Appendix A

DTD

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

<!-- définition des éléments -->

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

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

                      <!ELEMENT role (\#PCDATA)>

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

                      <!ATTLIST message out CDATA \#REQUIRED >

                      <!ATTLIST message support ( normal, coded, armoured) "normal">

           <!-- Une action est composée d’un nom, de la méthode utilisée, du résultat et de l’instance à laquelle elle est rattachée-->
           <!ELEMENT action (methode?, donnee)>

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

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

                               <!ATTLIST methode description CDATA \#REQUIRED >

<!ELEMENT calcul (fonction, donnee+)>

                                          <!ELEMENT fonction (\#PCDATA)>

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

                    <!ATTLIST knowledge instance  CDATA \#REQUIRED >


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

                               <!ATTLIST donnee id CDATA \#IMPLIED >

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

Appendix B

Protocole Needham-Schroeder


```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE msc SYSTEM "Prouve.dtd">


<msc name="Needham-Schroeder" type="algo" outil="ISpi">

            <instance name="A" param_effectif="A1" >
                        <role>intruder</role>
            </instance>

            <instance name="B">
                        <role>operator</role>
            </instance>

<knowledge instance="A">
            <donnee id="d1" type=" pub_key"> pub(B) </donnee>
            <donnee id="d2" type=" priv_key"> prv(A) </donnee>

</knowledge>

<knowledge instance="B">
            <donnee id="d3" type=" pub_key"> pub(A) </donnee>
            <donnee id="d4" type=" priv_key"> prv(B) </donnee>

</knowledge>


            <message in="A" out="B" support="normal">
            <donnee id="msg1" type="message "> {A,Na}pub(B) </donnee>

            </message>
```

```
            <message in="B" out="A" support="normal">
            <donnee id="msg2" type="message "> {Na,Nb}pub(A) </donnee>

            </message>


<message in="A" out="B" support="normal">
                        <donnee id="msg3" type="message"> {Nb}pub(B) </donnee>

            </message>

</msc>
```

Appendix C

Attaque Neddham-Schroeder


```
 <?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE msc SYSTEM "Prouve.dtd">

<msc name="Needham-Schroeder" type="attaque" outil="Hermes">

             <instance name="A" param_effectif="A1">
                          <role>operator</role>
             </instance>

             <instance name="I" param_effectif="I">
                          <role>intruder</role>
             </instance>

<instance name="B" param_effectif="B1">
                          <role>operator</role>
             </instance>


<knowledge instance="A">
             <donnee id="d1" type=" pub_key"> pub(B) </donnee>
             <donnee id="d2" type=" priv_key"> prv(A) </donnee>

</knowledge>

<knowledge instance="B">
             <donnee id="d3" type=" pub_key"> pub(A) </donnee>
             <donnee id="d4" type=" priv_key"> prv(B) </donnee>

</knowledge>

<knowledge instance="I">
```

```
            <donnee id="di1" type=" pub_key"> pub(A) </donnee>
            <donnee id="di2" type=" pub_key"> pub(B) </donnee>
            <donnee id="di3" type=" priv_key"> prv(I) </donnee>

</knowledge>


            <message in="A" out="I" support="normal">
            <donnee id="msg1" type="message "> {A,Na}pub(I) </donnee>

            </message>

<action name="Récupération de {A,Na} par l’intrus" instance="I">
            <donnee id="d5" type="message "> {A,Na}pub(A) </donnee>

<methode description="décrypter {A,Na} avec pub(I)">
            <calcul>
                        <fonction>decrypt{d6,d7}_d8</fonction>

<donnee id="d6" type="Agent"> A </donnee>

                        <donnee id="d7" type="Integer"> Na </donnee>

                        <donnee id="d8" type="pub_key"> pub(I) </donnee>

            </calcul>
</methode>
            </action>

<knowledge instance="I">
            <donnee id="di4" type=" Integer"> Na </donnee>

</knowledge>


            <message in="I" out="B" support="normal">
            <donnee id="msgi1" type="message "> {A,Na}pub(B) </donnee>

            </message>


            <message in="B" out="A" support="normal">
            <donnee id="msg2" type="message "> {Na,Nb}pub(A) </donnee>

            </message>
```

```
<message in="A" out="I" support="normal">
                          <donnee id="msg3" type="message"> {Nb}pub(I) </donnee>

             </message>

<action name="Récupération de Nb par l'intrus" instance="I">
             <donnee id="d9" type="message "> {Nb}pub(I) </donnee>

<methode description="décrypter {Nb} avec pub(I)">
             <calcul>
                          <fonction>decrypt{d10}_d11</fonction>

<donnee id="d10" type="Integer"> Nb </donnee>

                          <donnee id="d11" type="pub_key"> pub(I) </donnee>

             </calcul>
</methode>
             </action>

<knowledge instance="I">
             <donnee id="di5" type=" Integer"> Nb </donnee>

</knowledge>


</msc>
```

Bibliography

[BLP02] Liana Bozga, Yassine Lakhnec, and Michaël Périn. L'outil de vérification HERMES. *Rapport Technique EVA*, 2002.

[BM02] Victor Bos and Sjouke Mauw. A Latex macro package for Message Sequence Charts. juillet 2002.

[JRV] Florent Jacquemart, Michael Rusinowitch, and Laurent Vigneron. Narrowing semantics for cryptographic protocols.

[lUIE] AI lab (UNIGE), Cassis (INRIA), and Information Security Group (ETHZ). Automated Validation of Internet Security Protocols and Applications.