RAPPORT TECHNIQUE EVA

Outil de vérification SECURIFY

Auteur : Véronique Cortier
Laboratoire Spécification et Vérification, CNRS UMR 643, ENS Cachan
61, Avenue du Président Wilson, 94230 Cachan, France
cortier@lsv.ens-cachan.fr

Date : 10 mai 2002

Rapport EVA numéro : 7

Version : 1

TRUSTED LOGIC S.A.
5 rue du Bailliage
78000 Versailles, France
www.trusted-logic.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

Adresse : Laboratoire Spécification et Vérification,
CNRS UMR 8643, ENS Cachan
61, Avenue du Président Wilson, 94230 Cachan, France
cortier@lsv.ens-cachan.fr

Résumé : Ce document a pour but de présenter rapidement l'outil de vérification des protocoles cryptographiques SECURIFY. Cet outil est connecté au langage du projet EVA.

Outil de vérification SECURIFY

Véronique Cortier
Laboratoire Spécification et Vérification, CNRS UMR 643, ENS Cachan
61, Avenue du Président Wilson, 94230 Cachan, France
cortier@lsv.ens-cachan.fr

10 mai 2002

Ce document a pour but de présenter rapidement l'outil de vérification des protocoles cryptographiques SECURIFY. Cet outil est connecté au langage du projet EVA décrit dans [3].

1 Définition de l'outil ajouté

SECURIFY prend en entrée un fichier `.cpl`, dérivé par le parser EVA d'un fichier `.eva` dans lequel est décrit un protocole ainsi que les propriétés qu'il doit vérifier. Notre outil répond `oui` lorsque la propriété est vérifiée ou `don't know` s'il n'a pas pu statuer. En théorie, il peut aussi ne pas terminer même si cela ne s'est jamais produit sur des cas pratiques. La présentation de l'algorithme ainsi que la preuve de sa correction est décrite dans [2].
Notre algorithme ne vérifie que des propriétés de secret, c'est-à-dire des propriétés du type `*A*G secret (K@s1.A)` (dans la syntaxe EVA). De plus, comme nous ne considérons que la connaissance de l'intrus et non la connaissance de chacun des participants, `K` doit être une valeur créée par l'agent `A` considéré et non une valeur que l'agent reçoit. Si une propriété ne vérifiant pas les hypothèses ci-dessus est décrite dans le fichier passé en entrée à notre outil, il répond immédiatement `don't know` ainsi que l'explication correspondante.
Enfin, notre algorithme ne fonctionne bien que sur des protocoles typés ce qui a été rendu possible dans la syntaxe EVA par des déclarations du type `basetype key`.

2 Etat d'avancement

2.1 Réalisation en Ocaml

Dans un premier temps, l'algorithme a été entièrement programmé en Ocaml et testé sur des protocoles rentrés dans un format *ad hoc*. La figure 1 décrit les résultats obtenus sur différents protocoles (typés) décrits dans [1].

2.2 Sortie graphique

Notre outil permet également à l'utilisateur d'obtenir les graphes successifs des preuves (ou de l'échec des preuves) qui ont permis d'obtenir la réponse `oui` ou `don't`

Protocole	Réponse	Temps ms
Otway-Rees	Oui	0,35
Woo and Lam	Oui	0,11
Denning-Sacco	Don't know	0,07
Needham-Schroeder-Lowe	Oui	1,08
Needham-Schroeder	Don't know	1,23
Wide-Mouthed-Frog (modifié)	Oui	4,76
Kao-Chow	Oui	8,94
Yahalom	Oui	21,06

FIG. 1 – Réponses du programme sur différents protocoles

`know`. Un tel graphe permet de comprendre pourquoi la preuve du secret a échoué. Ainsi, la figure de l'annexe A est une capture d'écran d'un des graphes obtenus sur le protocole de Needham-Schroeder (non corrigé) : lorsque l'algorithme cherche à vérifier que le nonce N_b de la troisième règle est bien protégé, il arrive en deux étapes sur un échec de la preuve. Nous expliquons précisément dans [2] comment retrouver la véritable attaque du protocole de Needham-Schroeder.

2.3 Connexion avec le langage du projet EVA

Dans un deuxième temps, nous avons connecté notre programme au langage EVA. Ce travail a demandé l'ajout de la notion de `basetype` dans le langage d'EVA. De plus, nous avons mis en évidence quelques bugs du traducteur d'EVA qui, depuis, ont été corrigés. Toutes ces corrections (du 4 février au 19 mars) sont répertoriées dans le fichier :
`http ://www.lsv.ens-cachan.fr/ goubault/EVA/history`.

La connexion entre notre outil et le langage EVA est désormais presque terminée. Elle fonctionne déjà sur le protocole d'Otway-Rees en un temps négligeable. Le fichier `.eva`, dont on a dérivé le `.cpl` fourni en entrée à notre outil se trouve dans l'annexe B.
Le test de notre outil sur des protocoles plus compliqués a montré un bug du parser d'EVA. Une fois ce bug corrigé, l'outil sera très rapidement opérationnel.

2.4 Conformité

Nous avons apporté une attention toute particulière au respect de la sémantique du langague EVA ([3]) lors de la connexion de notre outil. La principale difficulté était d'obtenir que la propriété à prouver, décrite en EVA, corresponde bien à la propriété prouvée par notre outil. Une correspondance exacte n'était pas possible. Aussi, nous avons établi la connexion de manière à ce que si la propriété prouvée par notre outil est vraie, alors celle décrite en EVA l'est aussi. Ainsi, lorsque notre outil répond `oui`, c'est que la propriété énoncée en EVA est vérifiée.

Références

[1] J. Clark and J. Jacob. A survey of authentication protocol literature. `http ://www.cs.york.ac.uk/~jac/ papers/drareviewps.ps`,

1997.

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

[3] Jean Goubault-Larrecq. Les syntaxes et la sémantique du langage de spécifi cation EVA. In *Technical Report*, Novembre 2001.

A Sortie graphique de SECURIFY

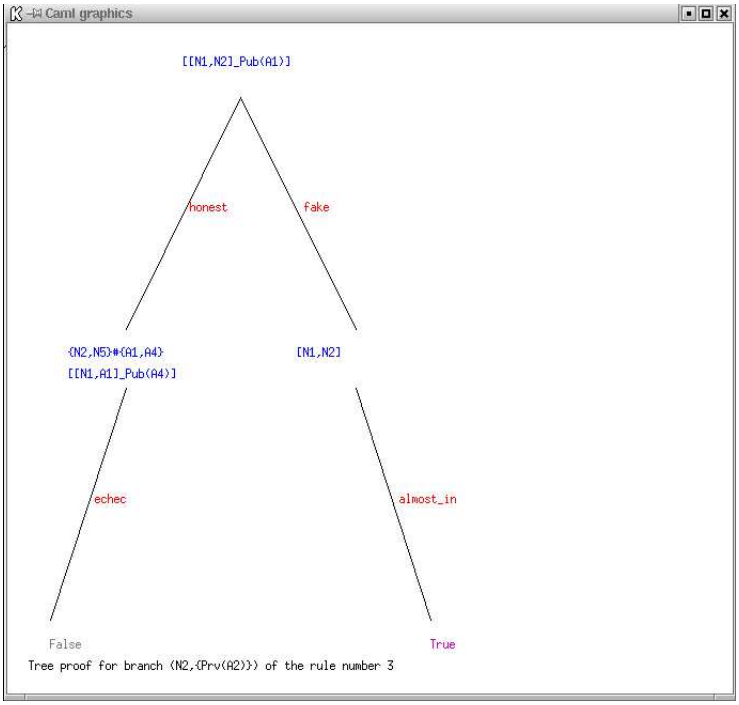

FIG. 2 – Une des sorties graphiques obtenues pour le protocole de Needham-Schroeder

B Protocole d'Otway-Rees décrit en langage EVA

```
Otway_Rees
  A, B, S : principal
  Kas, Kbs, Kab : number

  basetype key

  N, Na, Nb : key

  shr (principal) : number secret
```

```
everybody knows shr
alias Kas = shr(A)
alias Kbs = shr(B)

A knows A, B
B knows S

a, b, s : principal

{
  1. A -> B : N, A, B, {Na, N, A, B}_Kas
  2. B -> S : N, A, B, {Na, N, A, B}_Kas,
                       {Nb, N, A, B}_Kbs
  3. S -> B : N, {Na, Kab}_Kas, {Nb, Kab}_Kbs
  4. B -> A : N, {Na, Kab}_Kas
}

s1. session *{Kas, Kbs} A=a, B=b, S=s

assume secret (Kas@s1.A), secret (Kbs@s1.B),
      secret (Kas@s1.S), secret (Kbs@s1.S)
claim *A*G secret (Kas@s1.A),
      *A*G secret (Kbs@s1.B),
      *A*G secret (Kas@s1.S),
      *A*G secret (Kbs@s1.S),
      *A*G secret (Kab@s1.S)
```