RAPPORT TECHNIQUE PROUVÉ

Retour d'Expérience sur la Validation du Porte-Monnaie Électronique

Auteurs : Liana Bozga, Verimag
Stéphanie Delaune, France Télécom, LSV
Francis Klay, France Télécom
Laurent Vigneron, Loria

Date : 14 Mars 2005

Rapport numéro : 5

Version : 1.0

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

Résumé : Le domaine de la modélisation et de la vérification est une activité délicate et importante qui a connu une véritable explosion dans les années 1990. On dispose à l'entrée des années 2000 de toute une gamme de modèles et de méthodes plus ou moins avancés en ce qui concerne l'expressivité et l'automatisation.

Afin de définir les besoins et les priorités à mettre sur les outils consacrés à la vérification de protocoles cryptographiques qui seront développés au sein du projet RNTL PROUVÉ, nous proposons de travailler en situation réelle, sur des protocoles plutôt « durs », en effectuant le cycle suivant: modélisation, formalisation puis validation dans des outils existants. Ce travail est effectué ici pour deux versions d'un protocole de porte-monnaie électronique, dont l'une a été développée récemment par une équipe de France Télécom. Les outils retenus pour la réalisation de cette étude sont PROVERIF, HERMÈS et CASRUL, en raison de leurs caractéristiques très différentes.

Retour d'Expérience sur la Validation du Porte-Monnaie Électronique

Liana Bozga, Verimag
Stéphanie Delaune, France Télécom, LSV
Francis Klay, France Télécom
Laurent Vigneron, Loria

14 Mars 2005

1 Introduction

La communication par des canaux publics comme Internet s'est beaucoup développée ces dernières années. Les transactions utilisant ce médium sont de plus en plus nombreuses : communication client-fournisseur, services audiovisuels (vidéo à la demande), services bancaires, porte-monnaie électronique, protocoles d'enchères, vote électronique, ...

Parmi les différents aspects de la sécurité informatique, la cryptologie, c'est-à-dire l'étude des moyens permettant de transmettre des messages secrets, et l'étude des protocoles cryptographiques ont pris une importance considérable avec le développement du commerce électronique. Ce dernier terme recouvre pour commencer les échanges chiffrés d'information qui ont lieu lorsque l'on retire de l'argent dans un distributeur de billets à l'aide d'une carte bancaire, où le distributeur de billets s'engage dans un dialogue avec la carte du client et la banque pour vérifier que l'utilisateur de la carte est honnête, dispose de la somme demandée sur son compte et ne pourra pas récupérer les billets demandés sans que son compte en soit débité. Le commerce électronique recouvre également les paiements par carte à l'aide de terminaux portables où l'utilisateur doit confirmer son identité en entrant son code secret à quatre chiffres, et aussi la monnaie électronique qui a pour but d'émuler électroniquement la monnaie courante. Cette monnaie est, pour des montants peu élevés, plus intéressante qu'une transaction bancaire par carte qui a un coût élevé pour le commerçant.

Toutes ces applications demandent des garanties de sécurité élevées, portant sur des propriétés de secret et d'authenticité, mais aussi de nombreuses autres propriétés, parmi lesquelles, la non-duplication (des factures, dans l'intérêt du client), la non-révocation (des commandes, dans l'intérêt du commerçant), ... Pour assurer ces propriétés de sécurité, des moyens algorithmiques, tels que les chiffrements et les fonctions à sens unique ont été mis au point ; ils permettent d'assurer certaines propriétés, par exemple qu'il est très improbable qu'un individu puisse obtenir un message en clair à partir d'un chiffré sans connaître la clef de déchiffrement.

Depuis les années 80, on dispose d'algorithmes cryptographiques suffisamment sûrs, mais même si ces moyens algorithmiques remplissent parfaitement leur spécification, les propriétés sécuritaires ne sont pas pour autant toujours satisfaites. Les communications « sécurisées » sont en effet assurées par des protocoles dits *cryptographiques* qui utilisent ces moyens algorithmiques mais sont constitués de plusieurs messages. Plusieurs exemples célèbres ont montré qu'ils peuvent être attaqués (« man in the middle attack », « replay attack », « dictionary attack », ...) même en présence d'un chiffrement parfait. (Voir l'article de synthèse [7] : presque tous les protocoles d'authentification comportent des failles). Les failles qui permettent de telles attaques sont qualifiées de failles logiques et on s'accorde pour penser que l'étude des failles logiques des protocoles est orthogonale à l'étude des failles du système de cryptographie sous-jacent. Ces failles sont relativement subtiles, difficiles à déceler à la simple vue du texte du protocole.

Ces protocoles interviennent dans des communications électroniques en assurant des propriétés de sécurité, ils ont des caractéristiques spécifiques. On les trouve en très grand nombre, souvent sous la forme

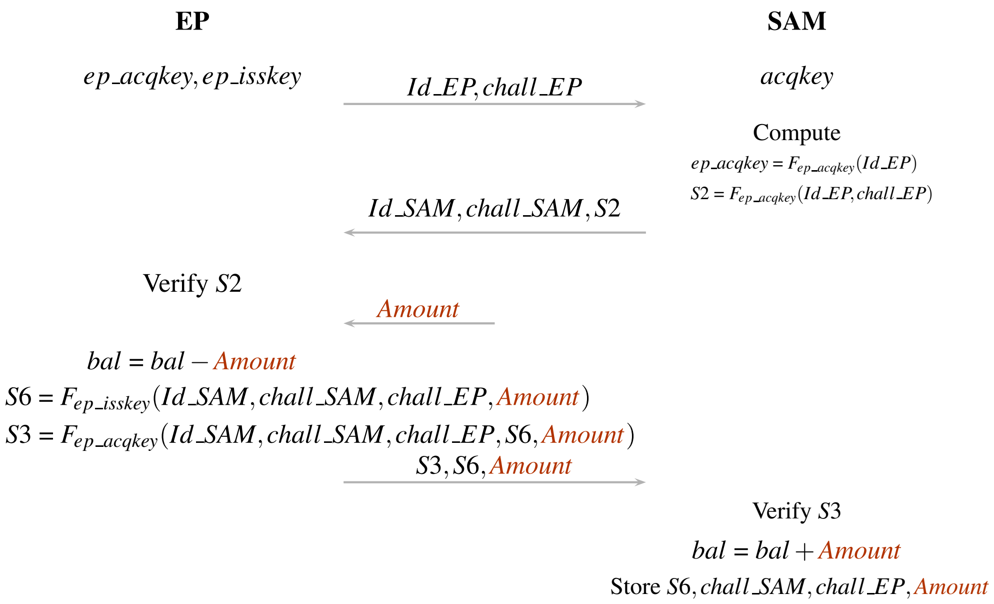


FIG. 1 – Description du Protocole de Porte-Monnaie Électronique – Approche Symétrique

de petites variantes d'un protocole connu, les failles de sécurité dans l'un de ces protocoles peuvent avoir des conséquences économiques graves, en particulier à cause de leur déploiement à grande échelle. De plus, d'autres aspects tels que le respect de la vie privée entre en ligne de compte, il est donc crucial de pouvoir vérifier formellement les propriétés de ces protocoles. Cependant ces vérifications sont dures et laborieuses, il en résulte donc des coûts non négligeables, une approche automatique de ce travail est importante.

Nous nous intéresserons à deux versions d'un protocole de commerce électronique permettant d'émuler électroniquement une transaction entre le porte-monnaie d'un client et la caisse d'un commerçant. Ces deux études de cas ont pour but de définir les besoins des logiciels de vérification « à venir ». Nous commencerons par une description de ces deux protocoles (Section 2). Les problèmes liés à la modélisation ayant déjà été abordés dans [2], nous procéderons ensuite à l'étape de validation dans les outils PROVERIF, HERMÈS et CASRUL. Ces outils possèdent des caractéristiques très différentes : PROVERIF et HERMÈS sont des outils consacrés à la preuve et considèrent un nombre de sessions non borné alors que CASRUL est un outil dédié à la recherche d'attaques pour un nombre borné de sessions. Une petite description de chacun de ces outils permettra au lecteur de comprendre l'analyse des résultats donnée dans les sections 3, 5 et 4.

2 Description des Protocoles

Nous décrivons deux versions d'un protocole permettant la réalisation d'une transaction bancaire : le but est d'émuler électroniquement une transaction faite avec de la monnaie courante. Cette application demande des garanties de sécurité élevées, portant sur de nombreuses propriétés, parmi lesquelles la non création de fausse monnaie, la non duplication des factures, ... Bien que les deux protocoles aient pour but la même fonctionnalité, nous allons voir qu'ils sont très différents en raison des primitives cryptographiques utilisées.

2.1 Protocole de Porte-Monnaie Électronique - Approche Symétrique

Ce protocole permet la réalisation d'une transaction entre un porte-monnaie électronique et un serveur : le but est de garantir un bon niveau de sécurité, et ce avec un bon niveau de performance grâce à l'utilisation du chiffrement symétrique (moins coûteux que les mécanismes de chiffrement asymétrique).

EP — **SAM**

ep_acqkey, ep_isskey — $acqkey, P_A$

$s, p = b^{-s}, Cert$

$\xrightarrow{Id\_EP, chall\_EP, p, Cert}$

Verify $Cert$ with P_A

Compute

$ep\_acqkey = F_{ep\_acqkey}(Id\_EP)$

$S2 = F_{ep\_acqkey}(Id\_EP, chall\_EP)$

$\xleftarrow{Id\_SAM, chall\_SAM, S2}$

$\xleftarrow{Amount}$

Verify $S2$

Compute $S6 = F_{ep\_isskey}(Id\_SAM, chall\_SAM, chall\_EP, Amount)$

Choose one coupon $(x, b^x mod\ r)$

$t = h(b^x mod\ r, Id\_SAM, chall\_SAM, chall\_EP, S6, Amount)$

$\xrightarrow{t}$

$\xleftarrow{c}$

$bal = bal - Amount$

$y = x + sc$

$\xrightarrow{y, S6, Amount}$

Compute $p^c b^y mod\ r == b^x mod\ r$

Verify t

$bal = bal + Amount$

Store $S6, chall\_SAM, chall\_EP, Amount$

FIG. 2 – Description du Protocole de Porte-Monnaie Électronique – Approche Asymétrique

Description du Protocole

Le protocole donné dans [9] est décrit Figure 1. Au cours d'un premier échange, le porte-monnaie électronique *EP* envoie au serveur *SAM* (Secure Application Module) son identité *Id_EP* et un challenge *chall_EP* (msg 1). *SAM* calcule la valeur *ep_acqkey*, celle-ci sera utilisée pour « chiffrer » les prochains messages. *SAM* répond au challenge de *EP* et donne son identité accompagnée d'un nouveau challenge (msg 2). *EP* vérifie la réponse fournie par *SAM* et reçoit le montant de la transaction (msg 3). À ce moment, *EP* débite sa balance et répond à *SAM* (msg 4) en construisant un message contenant l'identité de SAM, les valeurs des deux challenges, le montant de la transaction ainsi qu'un message noté *S6*. *SAM* vérifie *S3* en reconstruisant le message et crédite sa balance si la vérification se passe bien. De plus, il stocke certaines informations (message *S6*, *chall_SAM*, ...) pour résoudre d'éventuels litiges ultérieurs (F_{ep_isskey} est une donnée secrète entre le porte-monnaie et un tiers de confiance).

Description des Propriétés

Parmi, les propriétés que l'on souhaite vérifier sur ce protocole, on trouve la *non création de fausse monnaie* et la *non création de faux litiges*.

La *non création de fausse monnaie* passe par un certain équilibre dans les balances des différents agents, mais l'on peut choisir d'énoncer et de vérifier une propriété plus forte en s'intéressant à l'énoncé suivant : « Lorsque *SAM* termine une session apparemment avec *EP* en créditant sa balance d'un montant *Amount*, *EP* a bien débité sa balance de ce même montant *Amount*. ».

La *non création de faux litiges* doit assurer au serveur *SAM* qu'il ne stocke pas (à son insu) des faux *S6*. En effet, en cas de réclamations de la part d'un porte-monnaie qui aurait été débité d'un montant erroné, le serveur doit prouver sa bonne foi en exhibant le message *S6* qui a été généré au cours de la session correspondante. Si de faux *S6* peuvent être stockés sur le terminal *SAM*, le message *S6* n'a alors aucune valeur pour résoudre les éventuels litiges.

2.2 Protocole de Porte-Monnaie Électronique - Approche Asymétrique

Ce protocole permet la réalisation d'une transaction entre un porte-monnaie électronique et un serveur : le but est de garantir un bon niveau de sécurité et de gagner en ouverture par l'utilisation de méthodes de chiffrement asymétrique, et ce à faible coût. Ces besoins spécifiques ont conduit à développer une solution originale décrite dans [9].

Description du Protocole

Le protocole est décrit Figure 2. Au cours d'un premier échange, le porte-monnaie électronique envoie au serveur *SAM* son identité *EP*, un challenge *chall_EP*, sa clef publique b^{-s} et *Cert* pour certifier sa clef publique (msg 1). *SAM* vérifie le certificat et calcule la « clef symétrique » *ep_acqkey*, celle-ci sera utilisée pour « chiffrer » les prochains messages. *SAM* répond au challenge de *EP* et donne son identité accompagnée d'un nouveau challenge (msg 2). *EP* vérifie la réponse fournie par *SAM* et reçoit le montant de la transaction (msg 3). Ensuite, *EP* construit un message contenant $b^x \ mod \ r$, l'identité de *SAM*, son identité, les valeurs des deux challenges ainsi que le montant de la transaction, il le hache et l'envoie à *SAM* (msg 4). *SAM* stocke le message et génère un nombre aléatoire c qu'il envoie à *EP*. À ce moment, *EP* débite sa balance et répond à *SAM* (msg 6) pour lui permettre de vérifier (en le reconstruisant) le hash qu'il a reçu à l'étape 4. Ceci est possible car $b^x mod \ r$ n'est rien d'autre que $p^c b^y mod \ r$ compte tenu des propriétés algébriques de l'exponentielle. Si la vérification se passe bien, il crédite sa balance. De plus, il stocke certaines informations (message *S*6, *chall_SAM*, ...) pour résoudre d'éventuels litiges ultérieurs (F_{ep_isskey} est une donnée secrète entre le porte-monnaie et un tiers de confiance).

Description des Propriétés

Bien que le protocole proposé soit très différent du précédent (approche symétrique), les propriétés à vérifier restent les mêmes.

La vérification des protocoles cryptographiques s'articule en trois axes. D'une part le test qui consiste à essayer des scénarios dans le but de trouver une attaque. Mais celui-ci est peu intéressant car il n'offre aucune garantie sérieuse. Certains outils comme le logiciel CASRUL s'intéressent donc à la recherche d'attaques pour un nombre borné de sessions permettant de garantir la propriété souhaitée pour un nombre de sessions borné (à la fois en parallèle et en séquence). De nombreux travaux théoriques récents [4, 5, 8, 12, 13] dans lesquels des procédures de décision sont décrites, montrent que ces outils, dédiés à la recherche d'attaques pour un nombre borné de sessions, pourront être étendus pour prendre en compte (au moins en partie) les propriétés algébriques de certains opérateurs plus ou moins complexes tels que le xor et l'opérateur d'exponentiation modulaire. Une troisième approche, consistant à faire de la preuve, est également développée (outils PROVERIF et HERMÈS). En effet, l'idéal est de prouver la propriété que l'on souhaite vérifiée dans un cadre général (nombre non borné de sessions). Mais le problème étant indécidable dans ce contexte, les démonstrateurs automatiques procèdent en général par abstraction [1, 3] et ne fournissent aucune garantie de terminaison, ni de résultats. En effet, en cas d'échec de la preuve, on ne pourra rien dire sur le protocole. Cette dernière approche est donc intéressante, mais semble moins prometteuse que la précédente.

3 Outil PROVERIF

Description de l'Outil

PROVERIF est un outil consacré à la preuve [1], et les protocoles y sont exprimés sous forme de règles Prolog. Une syntaxe proche du π-calcul (chaque rôle est décrit comme un petit programme) est également acceptée en entrée, elle est alors automatiquement convertie en clauses de Horn. L'outil permet de vérifier des propriétés de secret et aussi des propriétés d'authentification pour un nombre non borné de sessions. La taille des messages n'est pas bornée non plus. Les nonces sont abstraits par une fonction des paramètres reçus dans les messages précédents ce qui peut conduire à de « fausses » attaques. Il se peut, en effet, qu'un protocole ne soit pas prouvé correct car sujet à des attaques qui utilisent l'abstraction faite sur les nonces alors qu'une telle attaque n'est pas reproductible dans la réalité. Cet outil type, au moins en partie, les messages.

3.1 Étude du Protocole – Approche Symétrique

Modélisation du Comportement

La spécification du protocole telle qu'elle peut être donnée en entrée à l'outil ProVerif est relativement proche de la description du protocole donnée en Section 2. La modélisation n'a pas posé de problème majeur (pour plus de détails, cf. [2]) et le lecteur trouvera le codage en Annexe A.1.

Modélisation des Propriétés

Nous allons nous intéresser à la propriété de non création de fausse monnaie. Celle-ci peut se « traduire » en une propriété d'authentification forte, encore appelée « injective agreement » [11]. Nous allons vérifier qu'à chaque fois que le serveur *SAM* termine une session avec *EP* en créditant sa balance d'un certain montant, le porte-monnaie *EP* a joué une session avec *SAM* et a débité sa balance de ce même montant.

Résultats Obtenus

ProVerif conclut en prouvant la propriété souhaitée sur le protocole étudié.

Synthèse

L'installation, la mise en œuvre et l'apprentissage de ProVerif ont été relativement faciles. En ce sens, cet outil constitue une bonne référence de la qualité qu'il faudrait atteindre avec l'implantation issue du projet Prouvé.

Le formalisme de description du comportement issu des algèbres de processus semble naturel. Cet aspect, bien que subjectif, justifie les choix faits lors de la définition du langage de notre projet. Nous avons opté pour un formalisme de même nature enrichi avec des notions telles que scénarios, ressources partagées ou portée des variables. Le lecteur intéressé trouvera dans le premier livrable de la tâche 1 une description de ce langage. L'utilisation d'étiquetages (`begin`/`end`) pour la spécification de la propriété d'authentification s'est également révélée intuitive : elle identifie très naturellement les positions du protocole où les parties de la propriété sont évaluées. Ce point sera pris en compte lors de la définition du langage de propriétés.

Un point plus délicat concerne la justification des verdicts positifs. En l'état l'outil ne donne que très peu d'arguments justificatifs. D'un point de vue industriel, il n'est pas envisageable de déployer une application critique sur un simple acte de foi dans un outil. Pour contourner ce problème, des solutions, comme la génération de scripts de preuves pouvant être vérifiés par un outil certifié, sont possibles. Même si cette tâche peut s'avérer difficile, rien n'exclut une évolution de ProVerif dans ce sens.

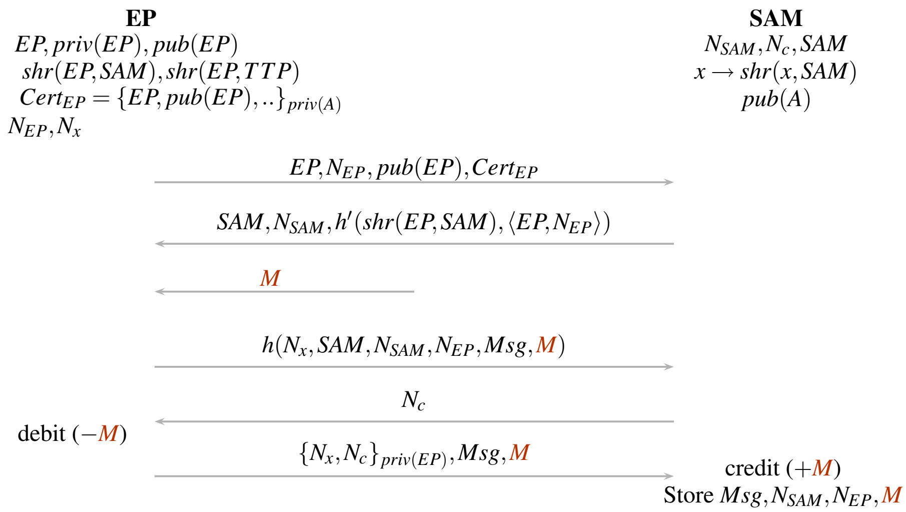
3.2 Étude du Protocole – Approche Asymétrique

Modélisation du Comportement

La modélisation du protocole proposée Figure 3 a pour but de formaliser au mieux le protocole en contournant les difficultés dues à l'utilisation de l'exponentielle modulaire et de ses propriétés puisqu'aucun outil ne permet à l'heure actuelle de prendre en compte cet opérateur. Bien que la modélisation faite des 3 derniers messages soit assez différente de la spécification initiale du protocole, elle reflète au mieux l'esprit du protocole dans le sens où le dernier message $\{N_x, N_c\}_{priv(EP)}$ dépend bien du nonce N_c engendré à l'étape précédente, comme y dépend de c dans le protocole. De même, le secret $priv(EP)$ permet de vérifier le hash reçu à l'étape 4, tout comme le secret s le permettait.

Remarquons tout de même que dans la modélisation proposée les rôles de x et $b^x \ mod \ r$ sont fusionnés dans le rôle de N_x. À la fin de la session, le nonce N_x est connu de l'intrus contrairement à x, et les propriétés algébriques de l'exponentielle utilisées lors de la vérification sont remplacées par un chiffrement asymétrique dans la modélisation. Cette solution n'a pas été retenue par les concepteurs du protocole pour des raisons évidentes de coûts.

Le protocole tel qu'il peut être donné en entrée à l'outil ProVerif est donné en Annexe A.2.



$$Msg = h'(shr(EP,TTP), \langle SAM, N_{SAM}, N_{EP}, M\rangle)$$

FIG. 3 – Formalisation du Protocole de Porte-Monnaie Électronique – Approche Asymétrique

Modélisation des Propriétés

Nous allons étudié la même propriété que pour le protocole version symétrique (cf. Section 3.1).

Résultats Obtenus

L'outil PROVERIF n'arrive pas à prouver la propriété souhaitée et fournit en sortie une trace. L'étude de cette dernière révèle qu'il s'agit en fait d'une « fausse attaque » due à l'abstraction faite par l'outil sur les nonces. Rappelons que les nonces sont modélisés par une fonction des messages reçus par l'agent créant le nonce : deux nonces engendrés à deux sessions différentes peuvent alors être égaux. Cette faiblesse, dans la génération des nonces, permet de monter l'attaque décrite Figure 4. Dans ce scénario, $I(SAM)$ (resp. $I(EP)$) désigne l'intrus se faisant passer pour le serveur SAM (resp. le porte-monnaie EP).

Description de l'Attaque :
Cette attaque demande pour être réalisée une implantation très particulière du générateur de nombres aléatoires. Celui-ci doit être une fonction locale à chaque agent, dépendant uniquement des messages précédemment reçus au cours de la session.

Cette attaque nécessite l'exécution de deux sessions `(i)` et `(ii)` en séquence : les deux sessions n'ayant pas besoin de s'entrelacer ni côté EP, ni côté SAM. La première session `(i)` commence et se déroule normalement jusqu'au message `(i4)` où l'intrus prend alors la place de SAM pour terminer la session. En particulier, le nonce N_I est généré par l'intrus. À ce stade d'exécution du protocole, le porte-monnaie EP a déjà été débité d'un montant M. L'exécution de la fin du protocole (côté serveur) devrait permettre à celui-ci d'être crédité de ce même montant. Mais l'intrus, en déchiffrant $\{Nx, N_I\}_{skEP}$ avec la clef publique du porte-monnaie EP peut récupérer le nonce Nx et construire le hash $h(Nx, SAM, N\_SAM, N\_EP, S, M_I)$ (correspondant à une transaction d'un montant M_I). SAM accepte alors ce message et envoie à l'intrus (ce dernier se faisant passer pour EP) un challenge Nc qu'il ne peut résoudre.

Afin de répondre à la requête du serveur, l'intrus commence une nouvelle session `(ii)` avec EP. L'intrus joue alors le rôle du serveur. L'intrus est capable de répondre au challenge de EP (message `(ii.2)`) puisqu'il s'agit des mêmes nonces que lors de la première session `(i)`. L'intrus peut alors demander à EP de résoudre le challenge Nc, récupérant ainsi $\{Nx, Nc\}_{skEP}$, la réponse au défi que SAM lui avait demandé de résoudre. SAM termine alors la session `(i)` en créditant sa balance d'un montant M_I choisi par l'intrus.

(i1). $EP \rightarrow I(SAM)$: $EP, N\_EP, CertEP$

(i1). $I(EP) \rightarrow SAM$: $EP, N\_EP, CertEP$
(i2). $SAM \rightarrow I(EP)$: $SAM, N\_SAM, Fepacqey(EP, N\_EP)$

(i2). $I(SAM) \rightarrow EP$: $SAM, N\_SAM, Fepacqey(EP, N\_EP)$
(i3). $EP \rightarrow I(SAM)$: $h(Nx, SAM, N\_SAM, N\_EP, S6, M)$
(i4). $I(SAM) \rightarrow EP$: N_I
débit(M)
(i5). $EP \rightarrow I(SAM)$: $\{Nx, N_I\}_{skEP}$

(i3). $I(EP) \rightarrow SAM$: $h(Nx, SAM, N\_SAM, N\_EP, S, M_I)$
(i4). $SAM \rightarrow I(EP)$: Nc

(ii1). $EP \rightarrow I(SAM)$: $EP, N\_EP, CertEP$
(ii2). $I(SAM) \rightarrow EP$: $SAM, N\_SAM, Fepacqey(EP, N\_EP)$
(ii3). $EP \rightarrow I(SAM)$: $h(Nx, SAM, N\_SAM, N\_EP, S6, M)$
(ii4). $I(SAM) \rightarrow EP$: Nc
débit(M)
(ii5). $EP \rightarrow I(SAM)$: $\{Nx, Nc\}_{skEP}$

(i5). $I(EP) \rightarrow SAM$: $\{Nx, Nc\}_{skEP}$
crédit(M_I)

FIG. 4 – Reconstitution de l'Attaque à partir de la Trace Fournie par PROVERIF.

Remarque :
Cette attaque est entièrement due à la mauvaise implantation des nonces et non au codage du protocole que nous avons considéré. Une attaque similaire peut en effet être montée sur le protocole original.

Synthèse

Bien que l'attaque découverte n'en soit pas une, l'interprétation de la trace critique révèle une sensibilité du protocole. Elle indique en particulier que le remplacement du générateur de nonces par un compteur aboutirait à un protocole dangereux. L'origine de l'erreur, *i.e.* l'abstraction trop brutale de l'ensemble des nonces par l'outil, met en avant la difficulté de la prise en compte de la fraîcheur dès que l'on s'intéresse a un nombre de sessions non borné. Un point délicat a été le dépouillement de la trace litigieuse. Pour éviter ce travail laborieux, des outils d'aide à l'interprétation de traces auraient été fort utiles.

En ce qui concerne les propriétés, deux points sont à noter. Initialement les propriétés à vérifier semblaient difficiles à formaliser puisqu'elles concernaient des notions de haut niveau telles que la non création de fausse monnaie. L'expérience à montré qu'une propriété, a priori technique et de bas niveau, telle que l'authentification valuée permettait de modéliser de façon naturelle et pertinente des notions de cette nature. Dans le même ordre d'idée, les propriétés algébriques de l'exponentielle semblaient interdire l'utilisation d'outils ne prenant pas en compte cet aspect. Pour ce protocole, ce problème a en fait pu être contourné d'une façon raisonnablement convaincante en adaptant la spécification.

4 Outil Hermès

Description de l'Outil

HERMÈS est un outil dédié à la vérification de protocoles cryptographiques. Il permet de prouver le secret pour un nombre non borné de sessions, de participants et de nonces. De plus, la taille des messages est également non bornée.

HERMÈS est un outil fonctionnant par abstraction et réduisant ainsi le domaine des participants et des nonces à un nombre fini de constantes. HERMÈS calcule un invariant qui s'interprète comme une limite imposée sur les connaissances de l'intrus dans le système abstrait. Si l'invariant abstrait est initialement vrai dans le système abstrait, alors la propriété de secret est satisfaite par le protocole concret pour un nombre arbitraire de sessions exécutées en parallèles par un nombre arbitraire de participants générant un nombre arbitraire de données fraîches à chaque session. Pour plus de détails, le lecteur peut consulter [3].

Quand la propriété est satisfaite, HERMÈS produit un arbre correspondant à des preuves et dans le cas contraire il fournit des traces d'attaques potentielles. Comme pour PROVERIF du fait de l'abstraction, il se peut que HERMÈS exhibe une attaque abstraite qui ne correspond pas à une trace exécutable dans le modèle réel.

4.1 Étude du Protocole – Approche Symétrique

Modélisation du Comportement

La description de la version symétrique du protocole donnée en Section 2 a pu être modélisée sans aucune difficulté dans LAEVA [10], le langage d'entrée pour l'outil HERMÈS. Le codage complet de ce protocole peut être trouvé en Annexe B.1.

Modélisation des Propriétés

Du fait que l'outil HERMÈS permet exclusivement de vérifier des propriétés de secret, nous sommes obligés de nous restreindre à ce type de propriétés. Nous allons nous intéresser à l'impossibilité de créer de faux litiges. Cela revient à s'assurer qu'il n'est pas possible d'amener *SAM* à accepter des messages $S3$ dont la partie $S6$ contient un *Amount* différent de celui de $S3$.
La propriété consiste donc à vérifier qu'il est impossible de forger des messages $S3$ de la forme

$$F_{ep\_acqkey}(Id\_SAM, chall\_SAM, chall\_EP, F_{ep\_isskey}(Id\_SAM, chall\_SAM', chall\_EP', Amount'), Amount)$$

avec $Amount \neq Amount'$.

Le langage de propriétés actuel ne permet pas d'exprimer une propriété de secret portant sur un terme avec variables libres ($chall\_SAM'$, $chall\_EP'$) et des conditions ($Amount \neq Amount'$). En revanche, HERMÈS est capable de traiter les variables libres ; il suffit d'ajouter le terme $S3$ à l'ensemble des secrets abstraits à vérifier. Pour traiter la contrainte $Amount \neq Amount'$, nous avons instancié les variables *Amount* et $Amount'$ avec des constantes abstraites différentes. Le codage des propriétés peut être trouvé en Annexe B.1.

Résultats Obtenus

D'abord nous avons travaillé avec l'abstraction par défaut de HERMÈS dans laquelle il n'y a que deux participants, un honnête h et un malhonnête I. HERMÈS produit plusieurs traces d'attaque dans le système abstrait. L'une d'entre elles est présentée Figure 5.

Description de l'attaque :
L'attaque nécessite l'exécution de deux sessions en parallèles. La première `(i)` entre *EP* et *SAM* et la deuxième `(ii)` entre l'intrus (jouant le rôle de *EP*) et *SAM*. À la fin de la session `(i)` *SAM* va stocker (à son insu) un message $S6$ avec pour montant A' à la place de A. HERMÈS propose différentes abstractions :

1. un unique honnête h et un malhonnête I ;
2. autant d'honnêtes que de rôles dans le protocole et un malhonnête I.

La première abstraction confond les identités des participants honnêtes. Ainsi, la traduction dans le modèle concret de l'attaque décrite Figure 5 demande que l'égalité suivante soit vérifiée : id(*EP*) = id(*SAM*). Autrement dit, pour réaliser cette attaque, un terminal et une carte de paiement doivent avoir le même identificateur.

La seconde abstraction associe des noms différents à chacun des rôles. L'attaque présentée Figure 5 n'existe donc plus dans le modèle abstrait et la propriété est vérifiée. La transition `(ii2)` va produire le

$$
\begin{array}{llll}
\texttt{(i1).} & EP & \rightarrow SAM & : h, chall\_EP \\
\texttt{(i2).} & SAM & \rightarrow I(EP) & : h, chall\_SAM, F_{ep\_acqkey}(h, chall\_EP), A
\end{array}
$$

$$\texttt{(i2).}\ I(SAM) \rightarrow EP : h, chall\_SAM', F_{ep\_acqkey}(h, chall\_EP), A'$$

$$
\begin{array}{llll}
\texttt{(i3).} & EP & \rightarrow I(SAM) & : F_{ep\_acqkey}(h, chall\_SAM', chall\_EP, F_{ep\_isskey}(h, chall\_SAM', chall\_EP, A'), A'), \\
& & & F_{ep\_isskey}(h, chall\_SAM', chall\_EP, A'), A' \\
\texttt{(ii1).} & I(EP) & \rightarrow SAM & : h, (chall\_SAM, chall\_EP, F_{ep\_isskey}(h, chall\_SAM', chall\_EP, A'), A) \\
\texttt{(ii2).} & SAM & \rightarrow I(EP) & : h, chall\_SAM'', \\
& & & F_{ep\_acqkey}(h, chall\_SAM, chall\_EP, F_{ep\_isskey}(h, chall\_SAM', chall\_EP, A'), A), A'' \\
\texttt{(i3).} & I(EP) & \rightarrow SAM & : F_{ep\_acqkey}(h, chall\_SAM, chall\_EP, F_{ep\_isskey}(h, chall\_SAM', chall\_EP, A'), A), \\
& & & F_{ep\_isskey}(h, chall\_SAM', chall\_EP, A'), A
\end{array}
$$

FIG. 5 – Reconstitution d'une attaque à partir de la sortie de HERMÈS

message :$F_{ep_acqkey}(EP, chall_SAM, chall_EP, F_{ep_isskey}(SAM, chall_SAM', chall_EP, A'), A), A''$qui ne correspond pas au message attendu par *SAM* pour finir la session `(i)`. En effet, *SAM* s'attends à recevoir :$F_{ep\_acqkey}(SAM, chall\_SAM, chall\_EP, F_{ep\_isskey}(SAM, chall\_SAM', chall\_EP, A'), A)$.

Synthèse

La formalisation du protocole dans le langage d'entrée de HERMÈS semble naturelle. Cependant, la formalisation de la propriété a nécessité quelques codages pour rentrer dans le cadre de HERMÈS, outil dédié à la preuve de propriétés de secrets.

La vérification avec HERMÈS a permis d'obtenir une (fausse) attaque, *i.e.* une trace critique. Les conditions nécessaires pour la mise en oeuvre de cette attaque étant peu probables. Cette trace critique montre que certaines hypothèses sur l'implantation du protocole sont nécessaires pour sa correction. L'attaque décrite n'est possible que si une confusion de type corroborée avec une confusion de la taille de termes peuvent avoir lieu. Ainsi, $(chall\_SAM, chall\_EP, F_{ep\_isskey}(h, chall\_SAM', chall\_EP, A'), A)$ peut être confondu avec un nonce. et les noms des participants qui jouent les rôles *EP* et *SAM* peuvent être confondus.

4.2 Étude du Protocole – Approche Asymétrique

Modélisation du Comportement

La modélisation du protocole dans le langage LAEVA a nécessité quelques codages :

1. Les fonctions de hachage ont été modélisées par un chiffrement avec une clé publique dont la clé privée n'est connu par aucun des participants : c'est le rôle des paires de clés `(Kpriv, K)` et `(Hpriv, H)`.
2. Nous avons contourné les difficultés dues à l'utilisation de l'exponentielle modulaire et de ses propriétés en utilisant le codage présenté en Section 3.2.

La description complète du protocole peut être trouvée en Annexe B.2.

Modélisation des Propriétés

Nous avons étudié les mêmes propriétés que pour la version symétrique du protocole (cf. Section 4.1).

Résultats Obtenus

HERMÈS conclut en prouvant la propriété.

Synthèse

Nous avons modifié le protocole car HERMÈS ne permet pas de prendre en compte les propriétés algébriques de l'opérateur d'exponentiation modulaire. Cette modélisation est une sous-approximation, le codage ne préserve pas toutes les propriétés mais seulement celles qui assurent l'exécution de protocole. En conséquence, la correction prouvée par HERMÈS sur ce modèle ne garantit pas forcément la correction du protocole initial.

La méthode de vérification implantée par HERMÈS ne permet pas de test en retard sur des messages. Ceci implique que le test effectué par *SAM* (au dernier pas il vérifie le message S_3 reçu au pas d'avant) ne correspond pas exactement à la spécification. Cette approximation est un peu grossière et il sera souhaitable d'avoir une modélisation appropriée.

L'étude de ce protocole montre le besoin de disposer d'un langage de description plus expressif permettant entre autres l'expression des opérateurs algébriques, ainsi que de leurs propriétés. En plus, cette étude montre la nécessite des modèles basés sur des scénarios, afin de prendre en compte les hypothèses spécifiques à l'exécution, comme par exemple l'absence de sessions parallèles.

5 Outil CASRUL

Description de l'Outil

CASRUL est un environnement d'analyse de protocoles de sécurité composé de plusieurs outils. Les protocoles y sont décrits dans un langage de spécification haut niveau, modulaire et très expressif ; ainsi, chaque rôle d'un protocole est décrit comme un processus indépendant, et une session d'un protocole est décrite comme la composition parallèle de rôles. Un traducteur convertit automatiquement cette spécification en un ensemble de règles de réécriture, qui sont utilisées par l'outil de vérification ATSE (Attack Searcher). ATSE est l'implantation directe en Objective CAML et l'approche basée sur une logique contrainte décrite dans [6, 14]. Cet analyseur de protocoles utilise le modèle de Dolev-Yao pour l'intrus, et peut chercher des attaques à différents niveaux : en mode typé, ou en mode non typé, exploitant par exemple l'associativité de la concaténation de messages.
Lorsqu'une attaque est trouvée, elle est décrite très clairement par l'échange des messages entre les agents et l'intrus.

5.1 Étude du Protocole – Approche Symétrique

Pour cette étude nous avons procédé en deux temps. Dans le premier nous avons formalisé classiquement la spécification standard du porte-monnaie à clé symétrique puis nous avons essayé d'évoluer vers une spécification la plus complète et réaliste possible pour déterminer les limites de l'outil. C'est ce dernier travail qui est présenté ici, les ajouts portent essentiellement sur la nature des informations échangées entre les rôles qui est plus fidèle à la réalité et sur la modélisation d'un processus de non répudiation électronique imaginaire car il semble que dans la réalité ce processus ne soit pas réalisé en ligne.

Modélisation du Comportement

Le lecteur trouvera la modélisation du comportement dans l'annexe C.1. Parmi les points particuliers nous pouvons noter :

- Dans la spécification du protocole, *SAM* contient une clé principale qui est diversifiée en fonction de l'identifiant de *EP* pour obtenir la clé de ce dernier. Ce point a pu être modélisé de façon réaliste en définissant la fonction FSHR_EPSAM qui retourne pour chaque identifiant d'*EP* sa clé. C'est cette fonction qui modélise la clé principale qui est passée en argument de rôle *SAM* alors que *EP* ne reçoit que la valeur de cette fonction pour son identifiant.

- Classiquement lorsqu'un identifiant d'agent est inclus dans un message, c'est l'équivalent d'un pointeur sur cet agent qui est inclus dans la modélisation du message. Procéder de la sorte n'est pas fidèle en particulier si différents agents partagent un même identifiant. En pratique une telle situation ne peut pas toujours être totalement exclue, des agents pourraient par exemple partager un même identifiant dans les cas suivants :
  - L'identifiant est initialisé par le revendeur d'un équipement (comme par exemple le numéro de téléphone sur une carte SIM) et cette opération est sujette à erreurs.
  - Deux agents sont des instances de rôles distincts auxquels sont associés des équipements de nature différente (par exemple un serveur et un terminal).
  - Deux agents sont des instances d'un même rôle mais l'équipement associé provient de fournisseurs différents.

  Pour pouvoir jouer sur la relation entre agents et identifiants et évaluer son impact sur la sûreté du protocole le principe qui a été suivi est le suivant. Sur les canaux ne transitent que des identifiants et la relation entre agents et identifiants est modélisée par une liste de couples. Lors de la réception d'un identifiant, l'agent correspondant est extrait de la relation s'il existe ou identifié à l'intrus si ce n'est pas le cas. Cet agent n'intervient jamais dans le comportement du protocole, il n'est utilisé que pour la modélisation des propriétés.
- Le processus de non répudiation imaginaire a été modélisé en ajoutant un rôle TTP pour le tiers de confiance. Ce dernier reçoit en paramètre une référence sur la liste des $S6$ construite par SAM. Ce passage par paramètre correspond à une communication entre SAM et TTP via un canal sûr. EP peut interroger TTP en lui transmettant les paramètres de la transaction chiffrés avec une clé partagée ainsi que deux challenges : l'un pour la réponse positive et l'autre pour la négative. Si TTP trouve dans sa liste un $S6$ correspondant aux paramètres transmis il retourne le premier challenge sinon le second.

Modélisation des Propriétés

En utilisant l'authentification forte, l'authentification faible et le secret nous avons pu modéliser indirectement mais simplement les propriétés suivantes :

1. Si SAM crédite son compte d'un certain montant alors EP a débité le sien du même montant dans la sessions engagée avec SAM (authentification forte AUTH_AMOUNT de la spécification).
2. Si EP a débité son compte d'un certain montant dans une session terminée alors SAM a crédité le sien du même montant (authentification forte AUTH_AMOUNT1 de la spécification).
3. Si EP a débité son compte dans une session alors il obtient une réponse positive de TTP pour ses paramètres de session qui incluent le montant débité (secret BAD_TTP_ANSWER de la spécification).
4. Si TTP retourne une réponse positive pour des paramètres de session qui incluent le montant alors SAM a stocké un $S6$ et débité son compte en cohérence avec ces paramètres (authentification faible AUTH_CHECKREP de la spécification).

L'authentification AUTH_CHECKREP est faible car le verdict de TTP peut-être demandé plusieurs fois pour une même session. On remarquera également que la première propriété assure qu'à tout élément de la liste de $S6$ partagée entre TTP et SAM est associé un unique débit par un EP. En effet, seul SAM peut modifier cette liste et dans SAM l'ajout de $S6$ et le débit du compte est une opération atomique. Combiné avec la quatrième propriété ça nous donne : Si TTP retourne une réponse positive pour des paramètres de session qui incluent le montant alors EP a effectué la session correspondante.

Résultats Obtenus

La spécification a été testée dans l'outil et aucune attaque n'a été trouvée avec le mode par défaut qui implique une vérification typée afin d'éviter des détections multiples le plus souvent sans intéret.
L'introduction de sessions testant le bon fonctionnement de la spécification, comme par exemple en mettant pour EP un agent inconnu de SAM, ou pour SAM un agent inconnu de TTP, provoque les attaques attendues. De même, si l'intrus joue le rôle de EP, il peut sans difficulté tromper l'authentification de SAM par TTP sur les informations stockées.

Avec la vérification de type désactivée nous aurions du retrouver la pseudo attaque mise en évidence dans l'outil HERMÈS. Ce travail n'a pas pu être réalisé suite à une restriction de la prise en compte des règles conditionnelles dont la résolution est en cours. Nous avons également effectué la vérification en activant la propriété d'associativité de la concaténation de messages. Dans ces conditions une attaque est systématiquement trouvée, ce qui montre qu'une très mauvaise implantation du protocole pourrait nuire à son bon fonctionnement.

Synthèse

L'outil ne traite que des exécutions bornées, c'est pour une bonne part l'origine de sa souplesse qui nous a permis de spécifier facilement tous les aspects qui nous intéressaient grâce en particulier aux fonctions, variables globales et données structurées. Notons cependant que le protocole ne requière pas de propriétés algébriques particulières, si ça avait été le cas le travail aurait été plus délicat.

La spécification que nous avons traitée est relativement lourde mais l'évaluation symbolique des exécutions permet de prendre en compte un nombre de sessions entrelacées utile dans la pratique (de l'ordre de trois ou quatre). De façon générale les fondements de l'outil en font une aide précieuse pour la mise au point de protocoles car il fournit rapidement des traces d'attaques facilement compréhensibles tout en évitant les fausses attaques issues des outils mettant en oeuvre des abstractions.

Un point qui aurait pu nous poser des problèmes est la pauvreté de scénarii possibles. En particulier, il n'a pas été possible de spécifier la séquentialité des sessions qui est physiquement imposée par le contexte : à un instant donné un seul *EP* est inséré dans un *SAM*. D'un autre côté des propriétés d'invariance et une logique du passé auraient simplifié la modélisation des propriétés du processus de non répudiation.

5.2 Étude du Protocole – Approche Aymétrique

L'étude de cette variante du protocole a volontairement été effectuée sur une spécification standard, sans entrer dans les détails comme nous l'avons fait pour l'approche symétrique.

Modélisation du Comportement

La modélisation est lisible dans l'annexe C.2. Par rapport à l'approche symétrique, les points particuliers suivants sont à noter :

- Bien que CASRUL permette d'utiliser l'exponentiation dans les spécifications, l'outil de vérification ATSE, dans sa version actuelle, ne peut pas la traiter.Ainsi, l'expression b^{-s} représentant la clé publique de *EP* a été remplacée par la variable *Kep* (dont la clé privée, $inv(Kep)$, est également connue de *EP*) ; les expressions $b^x \ mod \ r$ et c ont été remplacées par des nonces, *Nx* et *Nc* respectivement ; quant à y dont la valeur, avec l'aide de b^{-s} et c, permet de retrouver $b^x \ mod \ r$, nous l'avons remplacée par la signature de *Nx* et *Nc* par $inv(Kep)$.
- Comme pour l'approche symétrique, le montant de la transaction est donné à *EP* dès le départ, et non engendré en cours d'exécution. Cela permet de tester des sessions à montants égaux.
- Nous n'avons pas représenté la balance des comptes de *EP* et de *SAM* car nous ne disposons pas dans le langage de spécification d'opérateurs arithmétiques. Cependant, la vérification s'effectuant sur le montant de la transaction cela n'est pas gênant.
- Comme pour l'approche symétrique, les chiffrements par *ep_acqkey* et *ep_isskey* sont représentés par des chiffrements symétriques à l'aide de clés composées, calculées à l'aide de fonctions de hachage.
- Le certificat *Cert* émis par *EP* dans le premier message est décrit par la signature du nom de *EP* et de son challenge par une clé privée (*Ksign*). Cette clé n'est connue que de *EP* ; la clé publique correspondante (*Kcert*) n'est connue que de *SAM*.

Modélisation des Propriétés

Nous n'avons modélisé que deux propriétés pour cette variante du protocole ; elles utilisent toutes deux l'authentification forte.

1. Si *SAM* crédite son compte d'une certain montant alors *EP* a débité le sien du même montant dans la sessions engagée avec *SAM* (authentification forte AUTH_AMOUNT de la spécification).
2. Si *EP* a débité son compte d'une certain montant dans une session terminée alors *SAM* a crédité le sien du même montant (authentification forte AUTH_AMOUNT1 de la spécification).

Résultats Obtenus

La spécification a été testée dans l'outil et aucune attaque n'a été trouvée avec le mode par défaut qui implique une vérification typée. Ces tests ont aussi considéré plusieurs sessions avec des montants de transaction identiques.
Si l'intrus est un acteur officiel du protocole, que ce soit dans le rôle de *EP* ou celui de *SAM*, il pourra facilement fausser les propriétés d'authentification sur le montant de la transaction.

Lorsque la vérification de type est désactivée, aucune attaque n'est trouvée lors de sessions entre agents honnêtes. Par contre, si la propriété d'associativité de la concaténation de messages est activée, une attaque est systématiquement trouvée, ce qui montre qu'une très mauvaise implantation du protocole peut nuire à son bon fonctionnement.

Synthèse

Cette variante asymétrique du protocole a pu être facilement spécifiée avec CASRUL, malgré la non disponibilité d'opérations importantes comme l'exponentiation et les opérateurs arithmétiques (addition, soustraction, multiplication). La modélisation obtenue reste fidèle à l'esprit du protocole.

Aucune attaque significative n'a été trouvée, que ce soit en mode typé ou non. Il est vrai que seules deux propriétés ont été étudiées, mais elles correspondent à la vérification du montant de la transaction, ce qui est l'un des buts essentiels dans ce protocole.

L'ajout d'un tiers sûr, comme dans l'approche symétrique, aurait pû être très facilement effectué ici également, mais il nous a semblé que cela aurait nuit à la bonne lisibilité de la spécification, sans apporter un intérêt supplémentaire à ce qui a été effectué dans la première approche.

6 Synthèse globale

Du point de vue du comportement, c'est essentiellement la prise en compte du contexte d'exécution des protocoles qui a posé des problèmes dans la plupart des outils. La séquentialité des sessions, issue d'une contrainte physique (à un instant donné, une seule carte peut-être insérée dans le lecteur) n'a pas pu être prise en compte par les différents outils. De ce point de vue, PROVERIF est l'outil le mieux placé, il utilise une algèbre de processus pour décrire le contexte, ce formalisme est flexible, simple et naturel à appréhender. L'utilisation de variables qui perdurent au fil des sessions est un aspect qui n'est pas présent dans tous les outils, or le solde *Credit* des différents rôles est une variable de ce type. Si nous avions voulu vérifier par exemple la décroissance des soldes de *EP*, une telle variable aurait été nécessaire. Les variables partagées entre plusieurs rôles, pas toujours disponibles dans les outils, sont également utiles pour modéliser par exemple des canaux sûrs ou un transfert d'information nécessaire pour la modélisation d'une propriété. Les fonctions, relations et plus largement les données structurées se sont révélées utiles pour modéliser par exemple la liste des justificatifs $S6$ que stocke *SAM* ou la clé de *SAM* qui doit être diversifiée en fonction de l'identifiant de *EP*.

Du point de vue de la vérification, les propriétés algébriques de l'exponentielle n'ont pu être prises en compte par aucun des outils. Une vérification approchée a cependant été réalisée. Le principe de la modélisation est de conserver les dépendances des différents éléments qui composent le calcul. Même si dans ce cas particulier la démarche est relativement convaincante, elle est peu naturelle et probablement pas généralisable. En conséquence la prise en compte des propriétés algébriques semble importante car beaucoup de protocoles s'appuient sur des propriétés arithmétiques. On peut classer les propriétés algébriques qui seraient utiles en trois catégories : les propriétés des opérateurs arithmétiques utilisés dans le domaine de la cryptographie, les propriétés des structures usuelles de modélisation comme les listes ou les ensembles, et des propriétés génériques mais simples pour les opérateurs qui sont parfois utilisés pour structurer les messages. Un point intéressant à noter est que bien que le solde *Amount* soit impliqué dans des opérations arithmétiques, ceci n'a pas posé de problème car les propriétés de sûreté du protocole se ramènent simplement à la sûreté du transfert du seul argument *Amount* (le montant à débiter) du calcul pour une session donnée. Bien que dans le meilleur des cas les seules propriétés offertes par les outils sont le secret et l'authentification, les expérimentations ont montré que des propriétés de haut niveau pouvaient se ramener à ces propriétés ou tout au moins être fortement étayées par des propriétés basiques. Néanmoins, un formalisme plus flexible de bas niveau comme un calcul d'invariants combiné à une logique temporelle de passé permettrait par exemple de formaliser plus simplement des propriétés de non répudiation. Un tel langage permettrait également la spécification d'une bibliothèque de définitions pour les propriétés usuelles du domaine comme par exemple l'authentification. En outre, une telle démarche aurait le mérite de donner à l'utilisateur une sémantique précise de ces propriétés pour un outil donné.

En ce qui concerne le processus de vérification lui-même, les outils utilisant des abstractions ont beaucoup de peine à fournir les justifications d'une vérification réussie. De même lors d'un échec, les informations retournées rendent laborieux le travail de compréhension de l'échec. Or, ce travail doit être réalisé pour déceler d'éventuelles erreurs de spécification ou de fausses attaques dues à la sur-approximation faites par les abstractions. À l'inverse, les outils fondés sur une exécution symbolique bornée retournent des informations concises et pertinentes mais sont incapables de traiter un nombre de sessions non borné. Pour la même raison ces outils offrent des primitives de spécification riches et une utilisation flexible très utile pour la mise au point des spécifications. Notons encore que ces outils peuvent être complets pour un nombre de sessions borné d'où leur intérêt lorsque le problème sort du spectre des outils traitant un nombre non borné de sessions. Le travail réalisé a mis en évidence une complémentarité indiscutable de ces deux approches : les outils restreints à un nombre de sessions borné pour la mise au point et la prise en compte de protocole nécessitant des primitives évoluées, et les outils traitant le cas non borné pour les autres cas.

Une faiblesse de tous les outils utilisés concerne la vérification de l'exécutabilité d'un protocole ou mieux, la vérification de l'accessibilité de certains points de contrôle. La plupart des outils n'offrent que très peu d'aide pour cet aspect qui est indispensable pour une mise au point rapide des spécifications. En effet, une telle vérification permettrait en particulier d'éliminer instantanément des erreurs de spécification qui autrement demandent un travail laborieux pour leur détection. De plus, l'absence d'une telle vérification est un danger réel car une spécification non exécutable vérifie trivialement des propriétés telles que l'authentification. En outre, dans la plupart des cas une restriction de cette vérification à des cas simples comme l'existence d'une session exécutable serait déjà d'une grande aide.

Pour terminer notons que durant ce travail les mécanismes d'abstraction ont montré un effet positif intéressant et inattendu : la surjection qui est à l'origine des fausses attaques permet parfois de mettre en évidence une faiblesse potentielle du protocole,

7 Conclusion

Sur la version symétrique du porte-monnaie électronique, nous avons proposé des modélisations convaincantes puisque relativement proche de la description fournie par les concepteurs du protocole. Ceci a été possible car les primitives utilisées sont des primitives du modèle classique : chiffrement, fonctions de hachage,... Cette étude de cas, relativement simple, a ensuite pu être traitée par des outils dédiés à la vérification de protocoles cryptographiques.

Du côté des propriétés a vérifier, nous avons pu vérifier la non création de fausse monnaie et une forme partielle de non-répudiation à l'aide des primitives de vérification usuelles que sont le secret et l'authentification. Des propriétés telles que la non-répudiation ou la non-duplication sont importantes pour ce genre de protocoles. À notre connaissance, aucun outil ne propose, à ce jour, un processus de vérification pour ces propriétés.

La version asymétrique du porte-monnaie électronique fait intervenir des propriétés algébriques de l'exponentiation modulaire. Cet opérateur n'étant accepté par aucun outil, seul un effort au niveau de la modélisation du protocole a pu nous permettre d'obtenir des résultats sur ce protocole. Mais l'étape de modélisation est une tâche délicate qui ne peut se justifier que de manière informelle. Il est donc nécessaire de disposer de modèles suffisamment expressifs pour être convaincants et ainsi pouvoir par la suite avoir confiance dans le résultat fourni par la vérification formelle.

La synthèse des expérimentations réalisées met en évidence un ensemble de lignes directrices pour le projet RNTL PROUVÉ. Les problèmes spécifiques rencontrés lors de ce travail ont permis d'alimenter les autres tâches du projet comme la mise au point du langage : ajout de variables mutables, équations permettant d'exprimer les propriétés algébriques des opérateurs, gestion des différents modes multi-sessions... Pour la suite du projet, c'est essentiellement les aspects liés aux propriétés à valider et leur vérification qui pourront être exploités pour la définition du langage de propriétés et l'évolution des outils de vérification du projet.

Références

[1] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW)*, pages 82–96, Cape Breton (Nova Scotia, Canada), 2001. IEEE Comp. Soc. Press.

[2] L. Bozga, S. Delaune, F. Klay, and R. Treinen. Spécification du protocole de porte-monnaie électronique. Rapport Technique 1 du projet RNTL PROUVÉ, June 2004. 12 pages.

[3] L. Bozga, Y. Lakhnech, and M. Périn. HERMÈS : An Automatic Tool for Verification of Secrecy in Security Protocols. In *Proc. 15th International Conference of Computer Aided Verification (CAV)*, volume 2725 of *LNCS*, pages 219–222, Boulder (Colorado, USA), 2003. Springer-Verlag.

[4] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 261–270, Ottawa (Canada), 2003. IEEE Comp. Soc. Press.

[5] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Deciding the security of protocols with Diffie-Hellman exponentiation and product in exponents. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, volume 2914 of *LNCS*, pages 124–135, Mumbai (India), 2003. Springer-Verlag.

[6] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In E. Brinksma and K. Guldstrand Larsen, editors, *Proceedings of CAV'02*, LNCS 2404, pages 324–337. Springer, 2002.

[7] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature. 1997.

[8] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 271–280, Ottawa (Canada), 2003. IEEE Comp. Soc. Press.

[9] M. Girault and J. Paillès. Contactless EP : A Public-Key Solution with Good Performances. 2001.

[10] J. Goubault-Larrecq. La syntaxe et la sémantique du langage de spécification eva. Technical report, Projet EVA, `http://www-eva.imag.fr/`, November 2002.

[11] G. Lowe. A Hierarchy of Authentication Specifications. In *Proc. 10th Computer Security Foundations Workshop (CSFW)*, pages 31–44, Rockport (Massachusetts, USA), 1997. IEEE Comp. Soc. Press.

[12] J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proc. 16th Computer Security Foundation Workshop (CSFW)*, pages 47–62, Pacific Grove (California, USA), 2003. IEEE Comp. Soc. Press.

[13] V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *Proc. 13th European Symposium On Programming (ESOP)*, volume 2986 of *LNCS*, pages 355–369, Barcelona (Spain), 2004. Springer-Verlag.

[14] M. Turuani. *Sécurité des Protocoles Cryptographiques : Décidabilité et Complexité*. Thèse de doctorat, Université Henri Poincaré, Nancy, Décembre 2003.

A ProVerif

A.1 Protocole Approche Symétrique

```
free c.

(*************** Configuration Parameters *****************)
param attacker = active.
param keyCompromise = none.
param injectiveAg = true.
param predicatesImplementable = check.

(******************** Functions ****************************)
private fun facqkey/1.
private fun fisskey/1.
fun hash/2.
authquery authentification/3.

(********************* Process EP **************************)
let processEP =
        in(c, IdX);
        new Nep; out(c, (IdX, Nep));
        in(c, m);
        let (IdY, NY, S2)= m in
        if S2 = hash(facqkey(IdX),(IdX, Nep)) then
        new M;
        begin authentification(IdX,IdY,M);
        let S6 = hash(fisskey(IdX), (IdY, NY, Nep, M)) in
        let S3 = hash(facqkey(IdX), (IdY, NY, Nep, S6, M)) in
        out(c, (S3,S6,M)).

(********************** Process SAM ************************)
let processSAM =
        in(c, IdY); in(c, m1);
        let (IdX, Nep) = m1 in
        new Nsam;
        let fepacqkey = facqkey(IdX) in
        let s2 =  hash(fepacqkey, (IdX, Nep)) in
        out(c, (IdY, Nsam, s2));
        in(c, m2);
        let (S3, S6, M) = m2 in
        if S3 = hash(fepacqkey, (IdY, Nsam, Nep, S6, M)) then
        end authentification(IdX, IdY, M).

(********************* Protocol ****************************)
process new Idep; out(c, Idep);
        new Idsam; out(c, Idsam);
        (!processSAM | !processEP)
```

A.2 Protocole Approche Asymétrique

```
(*****************************************************************
Declaration d'un canal public de communication
```

```
****************************************************************)
free c.

(***************************************************************
Configuration Parameters:
attacker: active /passive
keyCompromise: none/approx/strict
injectiveAg: false/true
param predicatesImplementable: check/nocheck
****************************************************************)
param attacker = active.
param keyCompromise = none.
param injectiveAg = true.
param predicatesImplementable = check.

(***************************************************************
Functions:
****************************************************************)
private fun facqkey/1.
private fun fisskey/1.
fun hash/2.
fun h/1.
fun pk/1.
fun encrypt/2.
reduc decrypt(encrypt(x,y),pk(y)) = x.

(***************************************************************
Property: injective agreement on Amount.
****************************************************************)
authquery  agreement_amount/1.

(***************************************************************
Process EP:
***************************************************************)
let processEP =
     in(c, pkX);
     new Nep;
     out(c, (pkX, Nep));
     in(c, m1);
     let (IdY, NY, S2)= m1 in
     if S2 = hash(facqkey(pkX),(pkX, Nep))
     then new M;
     let S6 = hash(fisskey(pkX), (IdY, NY, Nep, M)) in
     new Nx;
     out(c, h((Nx, IdY, NY, Nep, S6, M)));
     in(c,nc);
     begin agreement_amount(M);
     let y = encrypt((Nx, nc), skEP) in
     out(c, (y,S6,M)).

(***************************************************************
Process SAM:
***************************************************************)
let processSAM =
```

```
     in(c, IdY);
     in(c, m3);
     let (pkX, NX) = m3 in
     let fepacqkey = facqkey(pkX) in
     new Nsam;
     let s2 =  hash(fepacqkey, (pkX, NX)) in
     out(c, (IdY, Nsam, s2));
     in(c, m4);
     new Nc;
     out(c, Nc);
     in(c, m5);
     let (Y, S6X, MX) = m5 in
     let (Nx2, Z) = decrypt(Y, pkX) in
     if Z = Nc then
     if m4 = h((Nx2, IdY, Nsam, NX, S6X, MX)) then
     end agreement_amount(MX).

(*****************************************************************
Process Prinipal:
*****************************************************************)

process new skEP;
let pkEP = pk(skEP) in
out(c, pkEP);
new Idsam;
out(c, Idsam);
new secret;
(!processEP | !processSAM)
```

B Hermès

B.1 Protocole Approche Symétrique

Description LAEVA :

```
Contactless_Electronic_Purse

***********************************************************
type declaration
***********************************************************
EP,SAM: principal

Nep,Nsam,A : number

acq (principal) : number secret
alias Kes = acq(EP)

iss (principal) : number secret
alias Ke = iss(EP)

************************************************************
participants knowledge
************************************************************
```

```
EP  knows EP, SAM, Kes, Ke
SAM knows SAM, acq

************************************************************
protocol transitions
************************************************************
   {
    1. EP  -> SAM : EP, Nep
    2. SAM -> EP  : SAM, Nsam, {EP,Nep}_Kes, A
    3. EP  -> SAM : {SAM,Nsam,Nep,A}_Ke,
                    {SAM,Nsam,Nep,{SAM,Nsam,Nep,A}_Ke,A}_Kes
   }

  s.session* {Nsam} EP=EP, SAM=SAM

************************************************************
verification hypothesis
************************************************************

  assume secret (Kes@s.EP),
         secret (Kes@s.SAM),
         secret (Ke@s.EP)
```

Les secrets spécifiques au système abstrait de HERMÈS :
$S_6 = \{h, Nsam', Nep', A^s\}_{Ke}$ et $S_3\{h, Nsam, Nep, \{h, Nsam', Nep', A^s\}_{Ke}, A\}_{Kes}$
où s appartient à l'ensemble $\{hh; Ih; hI\}$ et $Nsam', Nep'$ sont des variables libres.
L'attaque présentée dans la figure 5 a été obtenue pour les deux secrets suivant : $S6 = \{h, Nsam', Nep', A^{ih}\}_{Ke}$
$S3 = \{h, Nsam, Nep, \{h, Nsam', Nep', A^{ih}\}_{Ke}, A\}_{Kes}$

B.2 Protocole Approche Asymétrique

```
Contactless_Electronic_Purse_Asymmetric_keys

alg : asym_algo
everybody knows alg

EP,SAM,TTP: principal

Nep,Nsam,Nx,Nc,A : number

keypair^alg SK, PK (principal)

keypair^alg Sa, Pa (principal)
everybody knows Pa

keypair^alg Kpriv, K
keypair^alg Hpriv, H

everybody knows K, H

shr (principal,principal) : number secret
alias Kes = shr(EP,SAM)
alias Ket = shr(EP,TTP)
```

```
alias Cert = {EP, PK(EP)}_Sa(TTP)^alg
alias S6 = {Ket,SAM, Nsam,Nep,A}_K^alg

EP  knows EP, SAM, Kes,Ket,SK(EP),PK(EP),Cert
SAM knows SAM, Kes
TTP knows Sa(TTP)

   {
    1. EP  -> SAM : EP, Nep, PK(EP), Cert
    2. SAM -> EP  : SAM, Nsam, {Kes,EP,Nep}_K^alg
    3. EP -> SAM  : {Nx,SAM, Nsam,Nep,S6,A}_H^alg
    4. SAM -> EP  : Nc
    5. EP  -> SAM : {Nx,Nc}_SK(EP)^alg, S6, A
    6. SAM -> TTP : A
   }

  s.session* {Nsam} EP=EP, SAM=SAM

  assume secret (Kpriv),
         secret (Hpriv),
         secret (Sa(TTP)@s.TTP),
         secret (Kes@s.EP),
         secret (Ket@s.EP),
         secret (SK(EP)@s.EP),
         secret (SK(EP@s.SAM))
```

C CASRUL

C.1 Protocole Approche Symétrique

```
% Symmetric Key Electronic Purse
%
% ------------------
% Alice-Bob Notation
% ------------------
% EP:  Electronic Purse,
% SAM: Secure Application Module (Server)
% TTP: Trusted Third Party
% IN:  Input device
%
%  EP  --> SAM : EP,ChEP
%  SAM --> EP  : SAM,ChSAM,S2
%  IN  --> EP  : Amount
%  EP  --> SAM : S3,S6,Amount
%  SAM **>>TTP : S6
%  SAM **> EP  : ep_do_repcheck
%  EP  **> TTP : EP,{EP,SAM,ChSAM,ChEP,Amount,ChYES,ChNO}_shr(EP,TTP)
%  TTP **> EP  : {ChYES "or" ChNO}_shr(EP,TTP)
%
%  S2 = {EP,ChEP}_shr(EP,SAM)
```

```
%  S3 = {SAM,ChSAM,ChEP,S6,Amount}_shr(EP,SAM)
%  S6 = {SAM,ChSAM,ChEP,Amount}_shr(EP,TTP)
%
%  **>  is a dummy part for the unspecified non repudiation process where
%       '**>>' is a secure channel.
%------------------------------------- EP -----------------------------------
role ElectronicPurse (EP, SAM, TTP : agent,
                      EP_ID   : text,
                      ShrEPSAM: message,
                      ShrEPTTP: message,
                      Amount  : text,
                      SND, RCV: channel(dy)) played_by EP def=
local
      SAM_ID    : text,
      ChEP      : text (fresh),
      ChSAM     : text,
      S2,S3,S6  : message,
      ChYES,ChNO: text (fresh),
      ANSW      : text,
      State     : protocol_id

const initial, terminal: protocol_id,
      wait_server_challenge, wait_do_checkrep, wait_ttp_answer: protocol_id,
      auth_amount, auth_amount1: protocol_id,
      ep_do_checkrep, bad_ttp_answer: text

init   State = initial
accept State = terminal

transition
1. State = initial
   /\ start()
   --|>
      SND(EP_ID.ChEP')
   /\ State' = wait_server_challenge

2. State = wait_server_challenge
   /\ RCV(SAM_ID'.ChSAM'.S2')
   /\ S2' = {EP_ID.ChEP}_ShrEPSAM
   =|>
      S6'={SAM_ID'.ChSAM'.ChEP.Amount}_ShrEPTTP
   /\ S3'={SAM_ID'.ChSAM'.ChEP.S6'.Amount}_ShrEPSAM
   /\ SND(S3'.S6'.Amount)
   /\ witness(EP,SAM,auth_amount,ChEP.Amount)
   /\ secret(bad_ttp_answer,EP)
   /\ State'= wait_do_checkrep

3. State = wait_do_checkrep
   /\ RCV(ep_do_checkrep)
   =|>
      SND(EP_ID.{EP_ID.SAM_ID.ChSAM.ChEP.Amount.ChYES'.ChNO'}_ShrEPTTP)
   /\ State'= wait_ttp_answer

4. State = wait_ttp_answer
```

```
   /\ RCV({ChYES}_ShrEPTTP)
   =|>
      request(EP,SAM,auth_amount1,ChEP.Amount)
   /\ State' = terminal

5. State = wait_ttp_answer
   /\ RCV({ChNO}_ShrEPTTP)
   =|>
      SND(bad_ttp_answer)
   /\ request(EP,SAM,auth_amount1,ChEP.Amount)
   /\ State' = terminal

end role

%------------------------------------ SAM ------------------------------------
role ServerAM (SAM,TTP  : agent,
               SAM_ID   : text,
               LAGID    : (agent.text) list,
               FShrEPSAM: function,
               LS6      : (message) list,
               SND, RCV : channel(dy)) played_by SAM def=

local
      EP_ID       : text,
      ChSAM       : text(fresh),
      ChEP, Amount: text,
      S2, S3, S6  : message,
      EP_X        : agent,
      DontCare    : agent,
      State       : protocol_id

const ready, assign_ep_ag, wait_amount, finished: protocol_id,
      auth_amount,auth_amount1,auth_checkrep: protocol_id,
      ep_do_checkrep: text

init   State = ready
accept State = finished

transition
1. State = ready
   /\ RCV(EP_ID'.ChEP')
   =|>
      S2'={EP_ID'.ChEP'}_FShrEPSAM(EP_ID')
   /\ SND(SAM_ID.ChSAM'.S2')
   /\ State' = assign_ep_ag

2. State = assign_ep_ag
   /\ in(EP_X'.EP_ID, LAGID)
   --|>
      State' = wait_amount

3. State = assign_ep_ag
   /\ not(in(DontCare'.EP_ID, LAGID))
   --|>
```

```
      EP_X' = i
   /\ State' = wait_amount

4. State = wait_amount
   /\ RCV(S3'.S6'.Amount')
   /\ S3'={SAM_ID.ChSAM.ChEP.S6'.Amount'}_FShrEPSAM(EP_ID)
   =|>
      LS6'=cons(S6',LS6)
   /\ request(SAM,EP_X,auth_amount,ChEP.Amount')
   /\ witness(SAM,EP_X,auth_amount1,ChEP.Amount')
   /\ witness(SAM,TTP,auth_checkrep,EP_X.EP_ID.SAM_ID.ChSAM.ChEP.Amount')
   /\ SND(ep_do_checkrep)
   /\ State'= finished

end role

%------------------------------------- TTP ------------------------------------
role TrustedThirdParty (TTP      : agent,
                        LAGID    : (agent.text) list,
                        FShrEPTTP: function,
                        LS6      : (message) list,
                        SND, RCV : channel(dy)) played_by TTP def=
local
      SAM_ID, EP_ID: text,
      SAM_X, EP_X  : agent,
      DontCare     : agent,
      ChSAM, ChEP  : text,
      ChYES, ChNO  : text,
      Amount       : text,
      S6           : message,
      State        : protocol_id

const ready,  assign_ep_ag, assign_sam_ag, reply_ok, finished: protocol_id,
      auth_checkrep: protocol_id

init   State = ready
accept State = finished

transition
1. State = ready
   /\ RCV(EP_ID'.{EP_ID'.SAM_ID'.ChSAM'.ChEP'.Amount'.ChYES'.ChNO'}_FShrEPTTP(EP_ID'))
   /\ in({SAM_ID'.ChSAM'.ChEP'.Amount'}_FShrEPTTP(EP_ID'), LS6)
   =|>
      State' = assign_ep_ag

2. State = assign_ep_ag
   /\ in(EP_X'.EP_ID, LAGID)
   --|>
      State' = assign_sam_ag

3. State = assign_ep_ag
   /\ not(in(DontCare'.EP_ID, LAGID))
   --|>
      EP_X' = i
```

```
   /\ State' = assign_sam_ag

4. State = assign_sam_ag
   /\ in(SAM_X'.SAM_ID, LAGID)
   --|>
      State' = reply_ok

5. State = assign_sam_ag
   /\ not(in(DontCare'.SAM_ID, LAGID))
   --|>
      SAM_X' = i
   /\ State' = reply_ok

6. State = reply_ok
   --|>
      SND({ChYES}_FShrEPTTP(EP_ID))
   /\ wrequest(TTP,SAM_X,auth_checkrep,
               EP_X.EP_ID.SAM_ID.ChSAM.ChEP.Amount)
   /\ State' = finished

7. State = ready % reply_nok
   /\ RCV(EP_ID'.{EP_ID'.SAM_ID'.ChSAM'.ChEP'.Amount'.ChYES'.ChNO'}_FShrEPTTP(EP_ID'))
   /\ not(in({SAM_ID'.ChSAM'.ChEP'.Amount'}_FShrEPTTP(EP_ID'), LS6))
   =|>
      SND({ChNO'}_FShrEPTTP(EP_ID'))
   /\ State' = finished

end role

%----------------------------------- Exchange ---------------------------------
role Echange (EP, SAM, TTP: agent,
              ID     : agent -> text,
              LAGID  : (agent.text) list,
              Amount : text,
              LS6    : (message) list) def=

local SND,RCV: channel(dy)

const fshr_epsam,fshr_epttp: function,
      snd,rcv: channel(dy)

init SND=snd /\ RCV=rcv

composition
    ServerAM(SAM,TTP,ID(SAM),
             LAGID,fshr_epsam,LS6,SND,RCV)
    /\ ElectronicPurse(EP,SAM,TTP,ID(EP),
                       fshr_epsam(ID(EP)),fshr_epttp(ID(EP)),Amount,SND,RCV)
    /\ TrustedThirdParty(TTP,
                         LAGID,fshr_epttp,LS6,SND,RCV)
end role

%--------------------------------- Environment --------------------------------
role Environment () def=
```

```
local ID: agent -> text,
      LAGID: (agent.text) list,
      LS6: (message) list

const a,b,s,ttp,u,i: agent,
      a_id,b_id,s_id,u_id,i_id: text,
      amount1,amount2,dummy_amount: text

init LS6 = []
     /\ ID =   {[a,a_id],[b,b_id],[s,s_id],[u,u_id],[i,i_id]}
     /\ LAGID = [a.a_id, b.b_id, s.s_id, i.i_id ]

knowledge(i) = {a_id,b_id,u_id,s_id,dummy_amount}

composition
%%% Normal session (no attack):
         Echange(a,s,ttp,ID,LAGID,amount1,LS6)

%%% Session with EP unknown to SAM and TTP (attack on auth_amount1):
%         Echange(u,s,ttp,ID,LAGID,amount1,LS6)

%%% Session with SAM unknown to TTP (secrecy attack):
%         Echange(a,u,ttp,ID,LAGID,amount1,LS6)

%%% Session with EP played by the intruder (attack on auth_checkrep):
%         Echange(i,s,ttp,ID,LAGID,amount1,LS6)

%%% Two normals sessions in parallel (no attack):
%         Echange(a,s,ttp,ID,LAGID,amount1,LS6)
%      /\ Echange(b,s,ttp,ID,LAGID,amount1,LS6)

%      /\ Echange(a,s,ttp,ID,LAGID,amount1,LS6)
%      /\ Echange(b,s,ttp,ID,LAGID,amount2,LS6)
end role


%------------------------------------- Goal -----------------------------------
goal
      secrecy_of bad_ttp_answer
      ServerAM authenticates ElectronicPurse on auth_amount
      ElectronicPurse authenticates ServerAM on auth_amount1
      TrustedThirdParty weakly authenticates ServerAM on auth_checkrep
end goal

%------------------------------------------------------------------------------
Environment ()
```

C.2 Protocole Approche Asymétrique

```
% Asymmetric Key Electronic Purse
%
% ------------------
% Alice-Bob Notation
```

```
% ------------------
% EP:  Electronic Purse,
% SAM: Secure Application Module (Server)
% TTP: Trusted Third Party
% IN:  Input device
%

%  EP  --> SAM : EP,Nep,Kep,CertEP
%  SAM --> EP  : SAM,Nsam,S2
%  EP  --> SAM : H(Nx,SAM,Nsam,Nep,S6,Amount)
%  SAM --> EP  : Nc
%  EP  --> SAM : {Nx,Nc}_inv(Kep),S6,Amount
%
%  CertEP = {EP,Kep}_inv(Kcert)
%  S2 = {EP,Nep}_shr(EP,SAM)
%  S6 = {SAM,Nsam,Nep,Amount}_shr(EP,TTP)
%
%------------------------------------ EP -----------------------------------
role ElectronicPurse (EP: agent,
                      Kep: public_key,
                      Ksign: inv(public_key),
                      ShrEPSAM: message,
                      ShrEPTTP: message,
                      Amount: text,
                      H: function,
                      SND, RCV: channel(dy)) played_by EP def=

local
      SAM: agent,
      ChEP, Nx: text(fresh),
      ChSAM, Nc: text,
      CertEP: {agent.public_key}_inv(public_key),
      S2, S6: message,
      State: protocol_id

const initial, wait_server_challenge, wait_nonce, wait_confirmed, terminal: protocol_id,
      auth_amount, auth_amount1: protocol_id

init State = initial
accept State = terminal

knowledge(EP) = {inv(Kep)}

transition

1. State = initial
   /\ start()
   --|>
      CertEP' = {EP.Kep}_Ksign
   /\ SND(EP.ChEP'.Kep.CertEP')
   /\ State' = wait_server_challenge

2. State = wait_server_challenge
   /\ RCV(SAM'.ChSAM'.S2')
```

```
   /\ S2' = {EP.ChEP}_ShrEPSAM
   =|>
      S6' = {SAM'.ChSAM'.ChEP.Amount}_ShrEPTTP
   /\ SND(H(Nx'.SAM'.ChSAM'.ChEP.S6'.Amount))
   /\ State' = wait_nonce
   /\ witness(EP,SAM',auth_amount,ChEP.ChSAM'.Amount)

3. State = wait_nonce
   /\ RCV(Nc')
   =|>
      SND({Nx.Nc'}_inv(Kep).S6.Amount)
   /\ State' = wait_confirmed

4. State = wait_confirmed
   /\ RCV({ChEP.Amount}_ShrEPSAM)
   =|>
      State' = terminal
   /\ request(EP,SAM,auth_amount1,ChSAM.ChEP.Amount)

end role

%----------------------------------- SAM ------------------------------------
role ServerAM (SAM: agent,
               Kcert: public_key,
               FShrEPSAM: function,
               H: function,
               SND, RCV: channel(dy)) played_by SAM def=

local
      EP: agent,
      Kep: public_key,
      CertEP: {agent.public_key}_inv(public_key),
      Amount, ChEP, Nx: text,
      ChSAM, Nc: text(fresh),
      X, S2, S3, S6: message,
      State: protocol_id

const ready, wait_amount, send_nonce, finished: protocol_id,
      auth_amount, auth_amount1: protocol_id

init State = ready
accept State = finished

knowledge(SAM) = {}

transition

1. State = ready
   /\ RCV(EP'.ChEP'.Kep'.CertEP')
   /\ CertEP' = {EP'.Kep'}_inv(Kcert)
   =|>
      S2' = {EP'.ChEP'}_FShrEPSAM(EP')
   /\ SND(SAM.ChSAM'.S2')
   /\ State' = wait_amount
```

```
2. State = wait_amount
   /\ RCV(S3')
   =|>
      SND(Nc')
   /\ State'= send_nonce

3. State = send_nonce
   /\ RCV({Nx'.Nc}_inv(Kep).S6'.Amount')
   /\ S3 = H(Nx'.SAM.ChSAM.ChEP.S6'.Amount')
   =|>
      State' = finished
   /\ SND({ChEP.Amount'}_FShrEPSAM(EP'))
   /\ witness(SAM,EP,auth_amount1,ChSAM.ChEP.Amount')
   /\ request(SAM,EP,auth_amount,ChEP.ChSAM.Amount')

end role

%----------------------------------- Exchange ---------------------------------
role Exchange (EP, SAM: agent,
               Kep, Kcert: public_key,
               Amount: text) def=

local
      SND,RCV: channel(dy)

const shrEPSAM, shrEPTTP, h: function,
      snd,rcv: channel(dy)

init SND=snd /\ RCV=rcv

knowledge(i) = {h}

composition
   ServerAM(SAM,Kcert,shrEPSAM,h,SND,RCV)
   /\ ElectronicPurse(EP,Kep,inv(Kcert),shrEPSAM(EP),shrEPTTP(EP),Amount,h,SND,RCV)
end role

%-------------------------------- Environment --------------------------------
role Environment () def=

const a, b, s, i: agent,
      ka, kb, ki: public_key,
      kcert1, kcert2: public_key,
      amount1, amount2, dummy_amount: text

knowledge(i) = {a,b,s,ka,kb,ki,inv(ki),dummy_amount}

composition
%%% Normal session (no attack):
      Exchange(a,s,ka,kcert1,amount1)

%%% Session with EP played by the intruder (attack for auth_amount):
%      Exchange(i,s,ki,kcert1,amount1)
```

```
%%% Session with SAM played by the intruder (attack for auth_amount1):
%      Exchange(a,i,ka,kcert1,amount1)

%%% Two normals sessions in parallel (no attack):
%      Exchange(a,s,ka,kcert1,amount1)
%   /\ Exchange(b,s,kb,kcert2,amount1)

%      Exchange(a,s,ka,kcert1,amount1)
%   /\ Exchange(b,s,kb,kcert2,amount2)

end role

%------------------------------------ Goal -----------------------------------
goal
     ServerAM authenticates ElectronicPurse on auth_amount
     ElectronicPurse authenticates ServerAM on auth_amount1
end goal

%------------------------------------------------------------------------------
Environment ()
```