RAPPORT TECHNIQUE PROUVÉ

Spécification du Protocole de Porte-Monnaie Électronique

Auteur : Liana Bozga, Verimag
Stéphanie Delaune, France Télécom, LSV
Francis Klay, France Télécom
Ralf Treinen, LSV

Date : 22 Juin 2004

Rapport PROUVÉ numéro : 1

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

Résumé : Cette étude de cas a pour but de contribuer à une première évaluation des besoins pour l'aspect description formelle des protocoles cryptographiques. Cet aspect est un préalable obligé avant d'aborder des points tels que la sémantique et la vérification. Le résultat de ce travail a guidé la définition de la syntaxe du langage de spécification développé dans la tâche 1 du projet:« Sémantique des protocoles et des propriétés ».

Parmi la multitude de protocoles existants celui qui a été étudié est un porte-monnaie électronique à clé publique développé récemment par France Télécom R&D car il reflète fidèlement les ambitions du projet. Ce protocole, sortant sans surprise, du spectre de tous les outils développés à l'heure actuelle, notre travail a consisté à modéliser au mieux le porte-monnaie électronique dans un sous ensemble représentatif d'outils existants. Cette étude met évidence, sur un cas réel, les carences et les faiblesses des outils actuels et permet ainsi d'affiner et de valider les objectifs du projet. D'un autre côté, ce travail montre que des lacunes importantes peuvent parfois être raisonnablement contournées modulo un codage adapté.

Spécification du Protocole de Porte-Monnaie Électronique

Liana Bozga, Verimag
Stéphanie Delaune, France Télécom, LSV
Francis Klay, France Télécom
Ralf Treinen, LSV

22 Juin 2004

1 Introduction

Les exemples de protocoles fournis comme étude de cas sont en général des protocoles simples [4], car n'utilisant que des primitives du modèle classique. Nous nous proposons ici d'étudier un protocole de porte-monnaie électronique développé récemment par une équipe de France Télécom. Cette étude de cas, contrairement aux protocoles que l'on peut trouver dans [4], présente de nombreuses caractéristiques justifiant son étude au sein du projet PROUVÉ.

Il s'agit en effet d'un protocole réel faisant intervenir un certain nombre de propriétés algébriques de l'opérateur d'exponentiation modulaire, ce qui motive l'affaiblissement de l'hypothèse du chiffrement parfait proposé dans le projet. Du côté des propriétés à vérifier, ce protocole devrait permettre la prise en compte de nouvelles propriétés puisqu'il ne s'agit apparemment pas d'un problème classique de confidentialité ou d'authentification. Enfin, des problèmes plus spécifiques sont également soulevés par cette étude de cas tels que la gestion des variables (portée, durée de vie,...) et la gestion de canaux de natures différentes.

Cette étude de cas est donc un protocole plutôt dur, permettant de mettre en avant les nouvelles fonctionnalités souhaitées pour le projet PROUVÉ.

2 Description du Protocole

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 [5].

2.1 Protocole

Le protocole donné dans [5] est décrit Figure 1 : au cours d'un premier échange, le porte-monnaie électronique *EP* envoie au serveur *SAM* son identité *Id_EP*, un challenge *chall_EP*, sa clé publique et *Cert* pour certifier sa clé publique (msg 1). *SAM* vérifie le certificat et calcule la « clé 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 *Amount* (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 mettant à profit les propriétés algébriques de l'exponentielle, le hash qu'il a reçu à l'étape 4 et de créditer sa balance si la vérification se passe bien. De plus, il stocke le message *S_6* pour résoudre d'éventuels litiges ultérieurs (F_{ep_isskey} est une fonction de hachage paramétrée par une donnée secrète entre le porte-monnaie et un tiers de confiance).

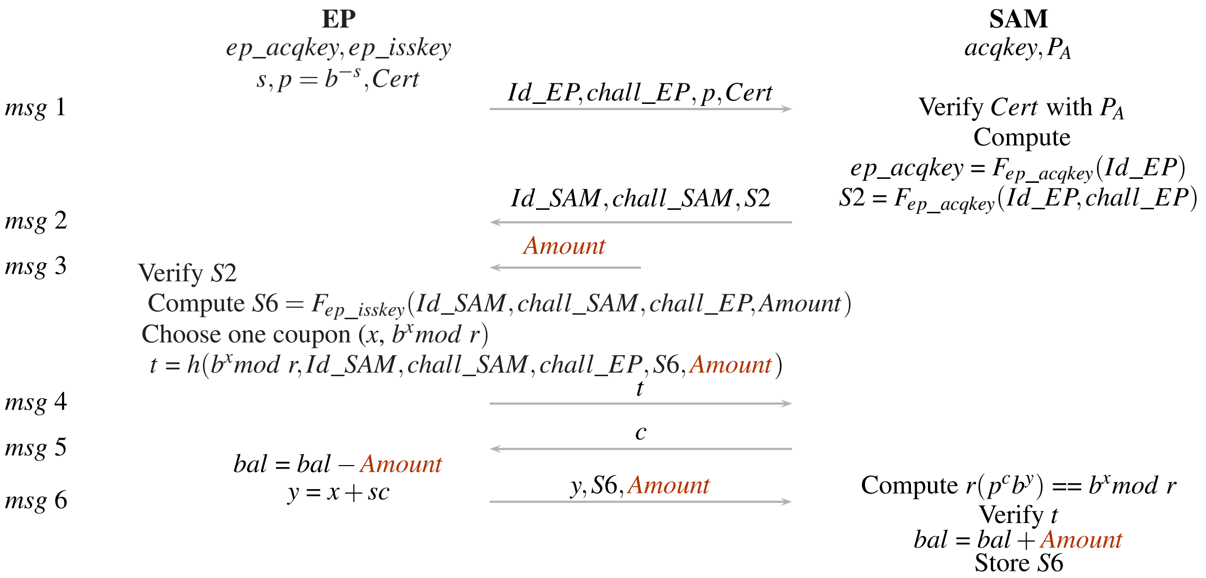

FIG. 1 – Description du Protocole de Porte-Monnaie Electronique

2.2 Vers la Modélisation du Protocole

En général, la majeure partie des spécifications sont en langue naturelle, et la première partie du travail consiste à lever les ambiguïtés. La modélisation de ce texte est une tâche délicate et laborieuse qui ne peut se faire qu'en partenariat avec les concepteurs du protocole pour ne pas biaiser leur vision initiale du protocole. En effet, une vérification sans adhésion à ce niveau perd tout son intérêt. Un certain nombre de points, parmi lesquels ceux détaillés ci-dessous, ont ainsi pu être éclaircis :

- Les fonctions F_{ep_acqkey} et F_{ep_isskey} ne doivent pas être vues comme des fonctions de chiffrement symétrique : elles ne sont pas inversibles. Autrement dit, il est impossible de retrouver les composantes du message $F_{ep\_isskey}(Id\_SAM, chall\_SAM, chall\_EP, Amount)$, même si on a connaissance de la fonction F_{ep_isskey}. Ces deux fonctions, F_{ep_acqkey} et F_{ep_isskey}, doivent être vues comme des fonctions de hachage. Pour vérifier les messages tels que $S2$, t et $S6$, la seule façon est de les reconstruire pour effectuer une comparaison.
- Le canal sur lequel circule le montant de la transaction est un canal sécurisé, l'intrus peut écouter sur ce canal, mais en aucun cas intercepter et/ou modifier les messages qui y circulent.
- Ce protocole a été concu pour réaliser du paiement local. L'objectif n'était donc pas d'obtenir un protocole Internet. Les sessions ont donc lieu les unes après les autres sans s'entrelacer (ni côté *EP*, ni côté *SAM*).

Tous les points cités plus haut sont implicites dans la description en langue naturelle et sont pourtant indispensables lorsque l'on souhaite formaliser le protocole et vérifier un certain nombre de propriétés.

Malheureusement, l'état actuel de l'art ne permet pas de prendre en considération toutes les caractéristiques de ce protocole et aucun des langages d'entrée des outils actuels ne permet de décrire le protocole en tenant compte de toutes ses subtilités. Nous proposons, comme point de départ, une formalisation raisonnablement convaincante du protocole dans le but de pouvoir le donner en entrée aux outils existants actuellement. Cette formalisation s'améliorera au fil du projet mettant ainsi en évidence l´évolution des outils.

3 Formalisation du Protocole

La modélisation du protocole proposée Figure 2 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.

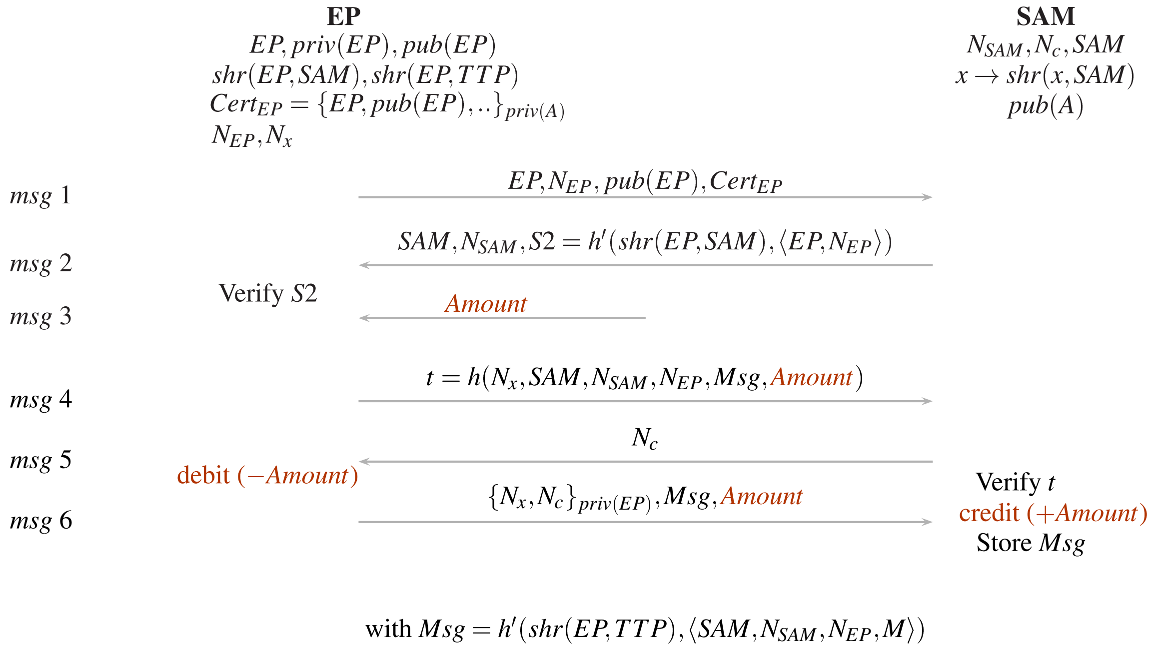

FIG. 2 – Formalisation du Protocole de Porte-Monnaie Electronique

Bien que la modélisation faite des 3 derniers messages soit assez différente de la spécification initiale du protocole, elle reflète bien 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, 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.

Expérimentation

Cette formalisation biaisée du protocole a permis de tester différents outils tels que SECURIFY et HERMÈS implémentés dans le cadre du projet RNTL EVA [1], CASRUL [3] et PROVERIF [2]. Le lecteur trouvera ces spécifications dans les annexes A, B et C.

Les langages d'entrées utilisés par ces outils sont dans l'ensemble suffisamment riches pour exprimer l'essentiel du protocole modulo un codage. Cependant notre cas d'étude, qui est un protocole plutôt dur, a permis de mettre en évidence certaines faiblesses.

- Absence de propriétés algébriques : elles sont nécessaires pour modéliser l'exponentielle.
- Absence de notion de canal sûr : il est utilisé pour transférer le montant de la transaction à EP.
- Absence de conditionnelles qui sont nécessaires pour vérifier $S2$ et t après réception des messages 2 et 6.
- Absence de constructions pour la description des scénarios qui permettraient ici de spécifier que les agents du protocole (serveur, porte-monnaie) ne peuvent pas être impliqués dans plusieurs communications en même temps.
- Absence de constructions pour la description de variables incluant des notions telles que la persistance, la portée, l'initialisation et les droits. Dans notre cas d'étude, elles permettraient de modéliser la balance se trouvant dans chacun des serveurs et des porte-monnaie. Chacune de ces variables est accessible en lecture/écriture par l'instance du rôle qui l'héberge et perdure durant les sessions.
- Absence de définition explicite du pouvoir déductif des rôles, ce qui implique qu'une même spécification peut avoir un comportement différent selon l'outil. De façon plus explicite, supposons par

exemple qu'une instance rôle doivent envoyer une constante a. Elle ne pourra exécuter cette opération que si elle peut déduire a à partir de ses connaissances initiales et des messages précédemment reçus. En conséquence, l'exécution de cet envoi dépend du pouvoir déductif du rôle.

Le langage de spécification issu de la tâche 1 améliore très largement la situation et le lecteur trouvera dans l'annexe D la formalisation du porte-monnaie électronique dans ce nouveau formalisme.

Des difficultés sont également apparues lors de la formalisation d'une des propriétés à vérifier : la non création de fausse monnaie que l'on traduit ici par un accord entre *SAM* et *EP* sur le montant de la transaction. En effet, la seule approche qui permette de traiter raisonnablement une partie des propriétés du protocole de porte-monnaie électronique semble être de se ramener à une propriété d'authentification avec accord sur la valeur du montant *Amount*. Mais les outils tels que SECURIFY et HERMÈS ne permettent d'exprimer que des propriétés de secret ce qui n'a que peu ou pas d'intérêt compte tenu de la propriété à laquelle on s'intéresse. En revanche des outils tels que PROVERIF et CASRUL, permettant d'exprimer des propriétés d'authentification, sont beaucoup plus intéressants.

Faute de primitives adaptées, des propriétés telles que la non répudiation et la non duplication n'ont pas été traitées. Pour ce qui concerne les propriétés, le point qui ressort est que les formalismes actuels sont bien trop restreints au vu des propriétés que doivent satisfaire les protocoles modernes. Il s'agit là d'un point difficile et qui, de par son importance, sera à aborder rapidement dans la tâche 1. L'aspect spécification de propriétés sera approfondi lors du travail de vérification et au cours de la prochaine étude de cas. Celle-ci sera un protocole de vote ou un protocole d'enchère électronique, les propriétés devant être satisfaites par ce type de protocoles sont bien plus complexes que celles du porte-monnaie électronique.

4 Conclusion

Nous avons pour l'instant, par un effort de modélisation et d'abstraction, exprimé le protocole de porte-monnaie électronique dans les langages utilisés à l'heure actuelle. Cette étude de cas montre qu'aucun de ces outils et de ces langages n'est réellement adapté, et constitue un *guide* pour définir le langage d'entrée qui sera ensuite utilisé tout au long du projet.

Références

[1] Projet RNTL EVA – Explication et Vérification Automatique de protocoles cryptographiques, `http ://www-eva.imag.fr/`

[2] ProVerif - Protocol Verifier, `http ://www.di.ens.fr/ blanchet/crypto.html`

[3] Casrul, `http ://www.loria.fr/equipes/cassis/softwares/casrul/`

[4] J. Clark and J. Jacob. A survey of authentication protocol literature, 1997.

[5] M. Girault and J.C. Paillès. Contactless : a public key solution with good performances, 2001.

A Spécification en LAEVA

Le langage LAEVA est un langage d'entrée commun aux outils SECURIFY et HERMÈS.

```
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* {Ns} 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))
```

Commentaires :
La modélisation du protocole de porte-monnaie électronique dans le langage LAEVA a nécessité quelques codages. Les fonctions de hachage sont modélisées par un chiffrement avec une clé publique dont l'inverse est une inconnue pour tous, c'est le rôle des paires de clés `(Kpriv, K)` et `(Hpriv, H)`. D'autre part, un échange de messages supplémentaire a été ajouté à la fin du protocole pour faire intervenir le principal `TTP` dans les règles du protocole. Celui-ci étant déclaré en tant que principal au début du protocole, il doit nécessairement intervenir au cours des échanges de messages, le non respect de cette règle entraîne une erreur à la compilation du protocole.

Aucune propriété n'a été spécifiée, car en LAEVA aucune construction ne permet d'exprimer une propriété d'authentification.

B Spécification pour CASRUL

```
Protocol Electronic_Purse_based_on_asymmetric_algorithms

Identifiers
  EP,SAM : user;
  fepacqkey, hash: function;
  Nep, Nsam, M, Nx, Nc: number;
  Kep: public_key;


(****************************************************************
Initial Knowledge:
 - Kep is the public-key of EP and Kep' its private key
 - hash is a hash function
 - fepacqkey the master key
****************************************************************)
Knowledge
  EP : EP, SAM, hash, fepacqkey(EP), Kep, Kep';
  SAM : SAM, hash, fepacqkey, Kep;


(****************************************************************
Rules of message exchanges:
****************************************************************)
Messages
 1. EP -> SAM : EP, Nep
 2. SAM -> EP : SAM, Nsam, hash( fepacqkey(EP), (EP, Nep))
(*********** 3. -> EP : M, montant de la transaction *************)
 4. EP -> SAM: hash((Nx, SAM, Nsam, Nep, M))
 5. SAM -> EP: Nc
 6. EP ->SAM: {Nx, Nc}Kep', M


(****************************************************************
Instrances Sessions:
- une session entre ep and sam
****************************************************************)
Session_instances
[EP:ep ; SAM:sam; hash:h; fepacqkey: fepacq, Kep: kep];
```

```
(***************************************************************
Capabilities and Initial Knowledge of the Intruder
***************************************************************)
Intruder divert, impersonate, Eaves_dropping;
Intruder_knowledge ep, sam, kep;

(***************************************************************
Property
***************************************************************)
Goal SAM authenticate EP on M;
```

Commentaires :
Le message `S6` et le certificat `Cert` n'ont pas été modélisés pour des soucis de lisibilité de la spécification, mais ces deux points ne posent pas de problèmes particuliers dans la syntaxe d'entrée utilisée par CASRUL. Une propriété d'authentification a aussi pu être modélisée dans ce langage.

C Spécification pour PROVERIF

```
(***************************************************************
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;                                              (* generates Nep *)
    out(c, (pkX, Nep));                                   (* sends msg 1 *)
    in(c, m1);                                            (* receives msg 2 *)
    let (IdY, NY, S2)= m1 in                              (* verifies msg 2 *)
    if S2 = hash(facqkey(pkX),(pkX, Nep))
    then new M;                                           (* generates M, the amount *)
    let S6 = hash(fisskey(pkX), (IdY, NY, Nep, M)) in
    new Nx;                                               (* generates Nx *)
    out(c, h((Nx, IdY, NY, Nep, S6, M)));                 (* sends msg 4 *)
    in(c,nc);                                             (* receives msg 5 *)
    begin agreement_amount(M);
    let y = encrypt((Nx, nc), skEP) in
    out(c, (y,S6,M)).                                     (* sends msg 6 *)

(***************************************************************
Process SAM:
***************************************************************)

    let processSAM =
    in(c, IdY);
    in(c, m3);                                            (* receives msg 1 *)
    let (pkX, NX) = m3 in
    let fepacqkey = facqkey(pkX) in
    new Nsam;                                             (* generates Nsam *)
    let s2 =  hash(fepacqkey, (pkX, NX)) in
    out(c, (IdY, Nsam, s2));                              (* sends msg 2 *)
    in(c, m4);                                            (* receives msg 4 *)
    new Nc;                                               (* generates Nc *)
    out(c, Nc);                                           (* sends msg 5 *)
    in(c, m5);                                            (* receives msg 6 *)
    let (Y, S6X, MX) = m5 in                              (* verifies msg 4 *)
    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)
```

Commentaires :
Le certificat `Cert` n'a pas été modélisé pour ne pas alourdir la spécification, mais ce point ne pose pas de problème particulier. Bien que ce langage d'entrée soit le plus riche des langages que nous avons étudiés

jusqu'à présent tant du point de vue des constructions (fonctions de hachages, destructeurs et test explicites, ...) que des propriétés (secret, authentification, secret fort, ...), il ne permet pas d'exprimer toutes les caractéristiques de notre protocole. Il manque en particulier des structures de données pour stocker des messages, des variables persistantes pour modéliser les balances des différents agents du protocole,...

D Spécification dans le formalisme PROUVÉ

```
# Electronic Purse version with asymmetric keys

signature
  # the function F produces a key, which is to be shared between a purse
  # and a module, from the master key of the module and the name of the purse
  F: (message,message) -> message;

  # the function FS takes a symmetric sign key k and list messages l, and
  # returns l signed with k.
  FS: (message,list(message)) -> message;

  # the hash function
  h: list(messages) -> message;

  b: integer; # the generator of the group
end

axioms
  x + y = y + x;
  x + (y + z) = (x + y) + z;
  minus(x) + x = 0;
  x + 0 = x;
  exp(exp(x,y),z) = exp(x,mult(y,z));
  mult(exp(x,y),exp(x,z)) = exp(x,y+z);
end

# the role of a purse during one transaction
role purse(
            ep_acqkey, ep_isskey;      # the two keys
            myname: principal;         # the name of the purse
            balance: mutable integer;  # the balance of the purse
            certificat: message;       # thew certificate of the purse
            s: integer                 # the secret s
          )
declare
  ep_challenge,sam_challenge: nonce;
  sam_name: message;
  amount: integer;
  s2,t,s6: message;
  p: integer; # the public key of the purse
  x: integer;
  c: integer;
begin
  p := exp(b,minus(s));
  ep_challenge := new;
  send([myname,ep_challenge,p,certificat]);
```

```
  recv([sam_name,sam_challenge,s2]);
  if s2 != FS(ep_acqkey,[myname,ep_challenge]) then fail; fi;
  recv(amount);
  s6 := FS(ep_isskey,[sam_name,sam_challenge,ep_challenge,amount]);
  x := new;
  coupon := exp(b,x);
  t := h([coupon,sam_name,sam_challenge,ep_challenge,s6,amount]);
  send(t);
  recv(c);
  balance := balance - amount;
  send([x + mult(s,c),s6,amount]);
end


# the role of an application module during one transaction
role secure_application_module(
             acqkey;                   # the key
             myname: principal;        # the name of the module
             balance: mutable integer; # the balance of the module
             signcheckkey: pubkey;     # the key to check a signature
     trace: mutable list(message) # audit
          )
declare
  ep_acqkey: message;
  id_ep, ep_challenge: message;
  s2,t,s6: message;
  sam_challenge: nonce;
  amount: integer;           # the amount to charge
  p: integer;                # public key of the purse
  c: integer;
begin
  recv([id_ep,ep_challenge,p,
        sign(rsa,inv(signcheckkey),[id_ep,p])]);
  ep_acqkey := F(acqkey,id_ep);
  s2 := FS(ep_acqkey,[id_ep,ep_challenge]);
  send([myname,sam_challenge,s2]);
  recv(t);
  c := new;
  recv([s3,s6,amount]);
  u := mult(exp(p,c),exp(b,s3));
  if
    t = h([u,myname,sam_challenge,ep_challenge,s6,amount])
  then
    balance := balance + amount;
    trace := cons([s6, sam_challenge, ep_challenge, amount],trace);
  else
    fail;
  fi;
end

# the keyboard where the user types the sum
role keyboard ()
declare amount: integer;
begin
```

```
  amount := new;
  send(amount);
end

variables
  ep_isskey(principal): integer;            # iss key of a purse
  acqkey: integer;                          # master key
  ep_balance(principal) : mutable integer;  # balance of a purse
  sam_balance(principal): mutable integer;  # balance of a module
  trace(principal) : mutable list(message); # audit trail of a module

  # the following variables are only used to implement mutual exclusion
  ep_lock(principal) : mutable bool;  # lock of a purse
  sam_lock(principal): mutable bool;  # lock of a sam
end

scenario
  begin
    # set intially all the locks to false
    forall purse,module: principal . begin
      ep_lock(purse) := false;
      sam_lock(module) := false;
    end;

    # compute the master key
    acqkey := new;

    # compute the purse keys
    forall purse: principal .
      ep_isskey(purse) := new;

    # initialize the traces
    forall module: principal .
      trace(module) := [];

    # infinitely start parallel sessions between purses and sams, such that
    # at every moment a purse or an agent is engaged in at most one
    # transaction
    forever
      forall purse,module: principal.
        if not(ep_lock(purse)) && not(sam_lock(module))
        then
          ep_lock(purse) := true;
          sam_lock(module) := true;
          parallel
            purse(F(acqkey,purse),
              ep_isskey(purse), purse, ep_balance(purse),
              sign(certkey,[purse,exp(b,minus(s))]),
              s)
          | secure_application_module(acqkey(module),module,
                                    sam_balance(module),
        trace)
  | keyboard()
          end;
```

```
  ep_lock(purse) := false;
  sam_lock(module) := false;
        fi;
    end;  # forever
  end     # begin
end       # scenario
```

Commentaires :
Ce nouveau langage d'entrée permet de prendre en compte toutes les caractéristiques de notre étude de cas. En particulier, le stockage des données est possible grâce à l'utilisation de variables mutables et de structures de données telles que des listes.