RAPPORT TECHNIQUE PROUVÉ

Spécification du Protocole de Vote Électronique

Auteur : Stéphanie Delaune, France Télécom Division R&D, LSV
Francis Klay, France Télécom Division R&D
Steve Kremer, LSV

Date : 18 Novembre 2005

Rapport PROUVÉ numéro : 6

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 : Cette nouvelle étude de cas a pour but de tester les limites du langage PROUVÉ. En effet, le protocole que nous avons choisi d'étudier est volontairement complexe tant au niveau de la modélisation des propriétés de sécurité que de la description du protocole lui-même en raison de la manipulation de structures de données telles que les listes.

Notre étude de cas est un protocole de vote qui a été mis au point par J. Traoré, ingénieur de recherche chez France Télécom. Ce protocole est basé sur le mécanisme de signature en aveugle et peut être considéré comme un dérivé du protocole de Fujioka, Okamoto et Ohta. Ce document introduit dans un premier temps le problème du vote électronique en général avant de décrire le protocole en lui-même et sa formalisation dans le langage PROUVÉ.

Spécification du Protocole de Vote Électronique

Stéphanie Delaune, France Télécom Division R&D, LSV
Francis Klay, France Télécom Division R&D
Steve Kremer, LSV

18 Novembre 2005

1 Introduction

Le vote à *bulletins secrets* est un moyen d'expression qui possède la caractéristique de ne pas divulguer les opinions individuelles. Le but du vote électronique est de se rapprocher du vote réel (et pourquoi pas, de faire mieux) en transposant dans un environnement informatique cette manière de s'exprimer anonymement. Suivant que l'on souhaite ou non garder les outils traditionnels que sont les bureaux de vote et les isoloirs, on est amené à considérer deux types de vote électronique : le vote *hors-ligne* et le vote *en-ligne*.

– Le vote *hors-ligne* consiste à conserver les outils du vote traditionnel que sont les bureaux de vote et les isoloirs et à ajouter une « machine à voter ». L'avantage de cette méthode est de permettre une comptabilisation rapide et efficace des bulletins, mais aussi de permettre aux électeurs de voter dans n'importe quel bureau de vote. Les bulletins seront ensuite acheminés par le réseau informatique vers leur véritable destination.
– Le vote *en-ligne* permet à un électeur de voter de chez lui en utilisant une simple connexion internet. L'inconvénient de cette méthode est le manque de confidentialité : Comment s'assurer qu'un électeur n'a pas voté sous la menace ?

Les protocoles de vote possèdent des caractéristiques spécifiques. Ils font intervenir différentes catégories de participants (autorités électorales, scrutateurs, électeurs), et surtout un nombre d'électeurs non connu *a priori*, mais faisant parti d'une liste prédéfinie. En fait, un protocole de vote peut être vu comme un assemblage de sous-protocoles servant chacun à la réalisation d'une tâche spécifique. Citons par exemple, le protocole permettant l'initialisation de l'élection, le protocole de vote proprement dit au cours duquel l'électeur s'engage sur une valeur et le protocole effectuant le comptage des bulletins et annonçant les résultats. Certains de ces sous-protocoles sont exécutés un nombre arbitraire de fois au cours d'une même élection, c'est le cas du protocole faisant intervenir l'électeur. Un protocole de vote électronique est en cela très différent des protocoles que l'on considérait jusqu'à présent servant, par exemple, à l'établissement d'une clef de session ou à la réalisation d'une transaction bancaire. Ces derniers ne faisaient intervenir qu'un nombre fixé de participants (en général 2 ou 3).

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. En effet, un protocole de vote, pour être utilisable doit vérifier de nombreuses propriétés de sécurité dont certaines semblent contradictoires (cf. Section 2.2). Pour assurer ces propriétés, il a fallu mettre en place de nouveaux mécanismes de base, plus complexes que les primitives cryptographiques classiques que sont le chiffrement (symétrique / asymétrique) et les fonctions à sens unique. Ainsi, un tel protocole est intéressant aussi bien du point de vue de la modélisation des propriétés algébriques que de la modélisation des propriétés de sécurité. D'autre part, ces protocoles ont un besoin crucial d'être vérifié : la moindre faille pourrait permettre la réalisation d'une fraude à grande échelle.

Dans un premier temps, ce document introduit plus en détail le problème du vote électronique ainsi que les propriétés de sécurité qu'un protocole de vote doit vérifier. En Section 3, nous présentons quelques éléments de cryptographie servant de « composants de base » à la conception des protocoles de vote électronique. Enfin, une description informelle du protocole [10] est proposée en Section 4 ainsi que sa formalisation dans le langage PROUVÉ (Section 5).

2 Présentation du Vote Électronique

Il existe de nombreuses solutions de vote électronique et il serait difficile de toutes les exposer. Cependant, parmi toutes ces solutions, trois grandes stratégies se détachent. Elles sont basées sur :

- le chiffrement homomorphique,
- les schémas de signature aveugle,
- les réseaux de mélangeurs.

2.1 Différentes Approches

Chiffrement Homomorphique. Le chiffrement homomorphique est une propriété de certains cryptosystèmes permettant de chiffrer un certain nombre de données les unes après les autres et de déchiffrer l'ensemble sans compromettre la confidentialité de chacun des éléments initiaux (cf. Section 3, pour plus de détails). La solution du vote électronique utilisant un schéma de chiffrement homomorphique nécessite la mise en place d'autres mécanismes tels que la signature, l'utilisation de preuves de connaissance à divulgation nulle de connaissance. Nous n'en dirons pas plus ici, puisque notre étude de cas n'est pas basé sur ce principe.

Signature en Aveugle. Les schémas de signature en aveugle ont été introduits par D. Chaum [2]. Ils permettent à une entité d'obtenir d'une autre la signature d'un message sans que le signataire ne le connaisse. Ainsi, chaque électeur va pouvoir obtenir une signature de son vote par une autorité qui vérifiera avant de signer que l'électeur est bien inscrit sur les listes électorales et qu'il n'a pas déjà voté pour cette élection. Commence ensuite la phase de vote proprement dite au cours de laquelle chaque électeur envoie à l'urne son vote signé. Bien entendu, seuls les votes signés par l'autorité seront comptabilisés.

Notre étude de cas est basé sur ce principe. Plus précisément, un schéma de *signature en aveugle à anonymat révocable* est utilisé afin de pouvoir, à l'aide d'une autorité compétente (appelée juge), retrouver l'identité du votant fraudeur et le couple (message, signature) en cas de litige. En plus du protocole de signature entre le signataire et l'utilisateur, il faut ajouter un protocole, appelé protocole de révocation, entre le signataire et le juge.

Réseaux de Mélangeurs. Introduit par D. Chaum [1], un mélangeur est une boîte noire prenant en entrée un nombre quelconque de données et qui a pour but de cacher la correspondance entre ces données et celles produites en sortie. L'utilisation de plusieurs mélangeurs en série, on parle alors de réseaux de mélangeurs, permet d'être sûr du résultat final dès lors qu'un des mélangeurs a réellement brassé les données.

L'utilisation d'un réseau de mélangeurs permet de réaliser un système de vote électronique de manière très simple. Mais, il est alors important de s'assurer que chacun des mélangeurs a bien fait son travail, *i.e.* que le mélange a bien été effectué, que des valeurs n'ont pas été modifiées, rajoutées ou enlevées. Lorsque chacun des mélangeurs est à même de fournir une preuve que ce travail a bien été effectué, on parle alors de *réseaux de mélangeurs universellement vérifiables*.

2.2 Propriétés de Sécurité

Un protocole de vote, pour être utilisable, doit vérifier un certain nombre de propriétés. Nous allons en dresser la liste et donner une brève description informelle de chacune d'entre elles. Exprimer ces différentes propriétés dans un langage de spécification avec une sémantique définie rigoureusement paraît, à première vue, difficile.

Secret des Votes (Anonymat). Personne ne doit être capable de faire le rapprochement entre un électeur et son vote. Il ne s'agit pas du secret au sens habituel du terme. En effet, supposons qu'il s'agisse d'un simple référendum, les valeurs *oui* et *non* ne sont pas secrètes, mais bien connues de l'agent malhonnête.

Éligibilité - Double Vote. Seules les personnes autorisées à voter le peuvent, et aucun électeur ne doit pouvoir voter deux fois lors d'une même élection. La première propriété est vérifiée si l'intrus ne peut pas obtenir au cours de la première phase du vote la signature ou le certificat lui permettant de continuer le protocole. La deuxième propriété (pas de double vote) assure le fait qu'un électeur ne puisse pas faire en sorte que son vote soit comptabilisé deux fois. Il faut donc que le scrutateur dispose d'un mécanisme lui permettant de rejeter les messages similaires. Mais attention, il ne faudrait pas non plus rejeter des votes valides.

Vérifiabilité (Individuellement / Universellement). Chaque électeur peut vérifier que son vote a été comptabilisé. Toute personne doit pouvoir se convaincre que tous les votes valides ont été comptabilisés sans avoir été modifiés.

Pas de Résultat Partiel. Personne ne doit être capable d'obtenir des résultats partiels, la connaissance de ces résultats pourrait influencer les électeurs n'ayant pas encore voté.

Sans Reçu. Aucun électeur ne doit être capable de prouver la manière dont il a voté. Obtenir ou être capable de construire un reçu de son vote, c'est à dire un document prouvant la manière dont on a voté, permettrait l'achat de vote ou la coercition (forcer quelqu'un à voter d'une certaine manière et s'en assurer ensuite).

Ces propriétés ne semblent pas facile à exprimer rigoureusement, et certaines d'entre elles semblent même contradictoires. En effet, chaque électeur doit pouvoir vérifier que son vote a été pris en compte (individuellement vérifiable), et pourtant il ne doit pas pouvoir prouver à un tiers comment il a voté !

Il est à noter que le vote traditionnel est loin d'être parfait. En effet, un attaquant pourrait forcer un électeur à ne pas aller voter ou faire en sorte que son vote soit considérer comme un vote nul. Il lui suffit pour cela de surveiller l'électeur pendant 24 heures, ou simplement de consulter les registres pour voir si celui-ci a apposé sa signature. L'attaquant peut également remettre à l'électeur un bulletin signé et vérifier que celui-ci se retrouve bien dans l'urne en assistant au dépouillement. Bien sûr, l'électeur pourra profiter de son passage dans l'isoloir pour échanger le bulletin mais si l'attaquant ne retrouve pas son bulletin dans l'urne, il pourrait y avoir des représailles.

D'autre part, la vérifiabilité est loin d'être une chose aisée dans le cadre du vote traditionnel. Un électeur ne souhaitant pas faire confiance à une tierce personne doit alors assister à l'élection et au dépouillement.

3 Cryptographie de Base

Nous commençons par rappeler les mécanismes de base que sont le chiffrement et la signature, puis nous nous intéressons à des schémas plus complexes souvent utilisés dans les protocoles de vote électronique.

Schéma de Chiffrement Pour assurer la non utilisation de certaines données confidentielles, des mécanismes de camouflage sont mis en place : c'est ce que l'on appelle communément le *chiffrement*. Il existe des algorithmes de chiffrement symétriques et asymétriques, déterministes ou probabilistes. Rappelons qu'un schéma de chiffrement consiste en un triplet d'algorithmes comprenant un algorithme de génération de clefs ainsi que deux algorithmes : un de chiffrement et un de déchiffrement. Les différents mécanismes de chiffrement permettent de garantir certaines propriétés. Il est par exemple très improbable qu'un individu obtienne de l'information sur un chiffré s'il ne connaît pas la clef de déchiffrement. Il est parfois intéressant d'utiliser des schémas de chiffrement disposant de propriétés supplémentaires. C'est le cas du chiffrement homomorphique, vérifiant l'égalité :

$$Enc(m_1,k) \times Enc(m_2,k) = Enc(m_1+m_2,k),$$

Ils trouvent de nombreuses applications, en particulier dans les protocoles de vote électronique. Le chiffrement est le mécanisme de base de la cryptographie et peut donc être modélisé (au moins dans sa version parfaite) dans tous les outils de vérification automatique existants à l'heure actuelle.

Dans le cadre de notre étude de cas, nous avons besoin du chiffrement asymétrique probabiliste.

```
(* Schema de chiffrement asymétrique et probabiliste *)
fun pencrypt/3.
fun decrypt/2.
fun pk/1.
equation decrypt(pencrypt(x,pk(z),y),z) = x.
```

Schéma de Signature La signature est une manière de prouver que la personne qui a produit le document est bien celle qu'elle prétend être. C'est un mécanisme de base, couramment utilisé en cryptographie. Il comprend un algorithme de génération de clefs, un algorithme de signature et un algorithme de vérification.

Ce schéma est relativement proche du schéma de chiffrement. Comme certains outils ne définissent pas cette primitive, un codage simple consiste alors à modéliser la signature par un chiffrement asymétrique avec la clef privée de l'agent. Sinon, le schéma est le suivant :

```
(* Schema de signature *)
fun pk/1.
fun sign/2.
fun checksign/2.
equation checksign(sign(x,y),pk(y)) = x.
```

Schéma d'Engagement Un schéma d'engagement permet à une personne de mettre en gage une valeur sans la dévoiler : la valeur gagée pourra être définitivement ou temporairement cachée. Un tel schéma comprend deux protocoles :

– un protocole, commit, au cours duquel la personne s'engage sur une valeur,
– un protocole, decommit, au cours duquel la personne révèle la donnée sur laquelle elle s'est précédemment engagée.

Contrairement aux fonctions de chiffrement et de signature, souvent prédéfinies dans les outils consacrés à la vérification de protocoles cryptographiques, les primitives commit et decommit d'un schéma d'engagement ne sont en général pas prédéfinies. Le schéma est le suivant :

```
(* Schema d'engagement *)
fun commit/2.
fun decommit/2.
equation decommit(commit(x,r),r) = x.
```

Pour les outils qui ne permettent pas de définir de nouvelles primitives, il est possible de coder ce mécanisme en utilisant le mécanisme de chiffrement. S'engager sur une valeur v consiste à générer un nombre aléatoire r et à envoyer le message $\{v\}_r$. Pour révéler la donnée, il suffit alors de révéler r.

Signature en Aveugle Le principe est simple : une personne va faire signer un message à quelqu'un en faisant en sorte que le signataire du message n'apprenne rien sur son contenu. C'est donc un protocole interactif faisant intervenir deux entités. Le schéma est le suivant :

```
(* Signatures en Aveugle *)
fun blind/2.
fun sign/2.
fun unblind/2.
equation unblind(sign(blind(x,y),z),y) = sign(x,z).
equation unblind(blind(x,y),y) = x.
```

Il semble également possible de coder la signature en aveugle si l'on dispose d'une primitive de chiffrement commutative et que l'on code la signature comme un chiffrement asymétrique avec la clef privée.

$$\begin{array}{ll} A \rightarrow S: & \{m\}_{\mathsf{pub}(A)} \\ S \rightarrow A: & \{\{m\}_{\mathsf{pub}(A)}\}_{\mathsf{prv}(S)} \end{array}$$

L'agent A peut alors déchiffrer le message reçu avec sa clef privée et obtenir m signé par S : $\{m\}_{\mathsf{prv}(S)}$. Ce codage est intéressant car des résultats théoriques récents étudient la vérification de protocoles cryptographiques en présence d'une primitive de chiffrement commutative [3]. On peut donc, dans un futur proche, espérer traiter cette primitive.

Signature en Aveugle à Anonymat Révocable Une variante des schémas de signature en aveugle consiste à rendre cet anonymat révocable. Pour un tel schéma, en plus du signataire et de l'utilisateur, une troisième entité peut intervenir, c'est l'autorité (encore appelée juge). Il faut également ajouter un protocole (appelé protocole de révocation) entre le signataire et l'autorité. Il existe deux types de levée d'anonymat, suivant l'information que l'autorité reçoit du signataire :

1. L'autorité reçoit la partie du protocole de signature venant du signataire et donne une information permettant à n'importe qui de retrouver le message et la signature.
2. À l'aide du message et de la signature, l'autorité permet au signataire de retrouver l'utilisateur ou la partie du protocole correspondant à la signature.

Nous proposons le schéma suivant :

```
(* Signature en Aveugle à Anonymat Révocable de type I*)
fun fairblind/2.
fun sign/2.
fun unblind/2.
fun revmsg/2.
fun revsign/2.

equation unblind(sign(fairblind(x,y),z),y) = sign(x,z).
equation revmsg(fairblind(x,y), sign(fairblind(x,y),z)) = x.
equation revsign(fairblind(x,y), sign(fairblind(x,y),z)) = sign(x,z).
```

Les fonctions revmsg et revsign sont ici considérées comme des fonctions privées, *i.e.* non connues de l'intrus. On aurait pu choisir de considérer ces symboles de fonctions comme des symboles de fonctions publiques et ajouter un troisième argument qui aurait été la clé privée du juge.

Preuve de Connaissance à Divulgation Nulle de Connaissance La preuve de connaissance à divulgation nulle de connaissance (de l'anglais *zero-knowledge*) consiste à prouver à une personne, lors d'un protocole interactif, sa connaissance d'un secret, sans rien révéler sur celui-ci. La plupart des protocoles à divulgation nulle de connaissance sont des itérations de protocoles en 3 phases (engagement, défi, réponse), mais ce n'est pas nécessairement le cas.

Dans notre étude de cas, nous avons besoin de modéliser le fait que le votant réalise le chiffrement de b avec la clef publique du réseau de mélangeurs $\mathsf{pub}(\mathcal{M})$ et il doit fournir une preuve P à divulgation nulle de connaissance accompagnant le message chiffré $\{b\}_{\mathsf{pub}(\mathcal{M})}$ pour prouver qu'il connaît le message en clair correspondant, *i.e.* b.

Nous proposons le schéma suivant :

```
fun proof/3.
fun checkproof/3.
fun ok/0.

equation checkproof(proof(pencrypt(m,pubk,r),m,pubk),
                    pencrypt(m,pubk,r),
                    pubk) = ok.
```

Canal Anonyme - Réseaux de Mélangeurs Un mélangeur est une boîte noire qui a pour but de cacher la correspondance entre les entrées et les sorties : l'expéditeur reste ainsi anonyme. Cette méthode est très utilisée dans les protocoles de vote électronique pour modéliser l'utilisation de canaux anonymes.

4 Description

Dans cette section, nous présentons les différentes phases du protocole de vote électronique faisant l'objet de notre étude. Certaines phases (trop complexes) ne sont pas détaillées (cf. [10], pour plus de détails).

4.1 Un Peu d'Histoire

Notre étude de cas est basée sur le mécanisme de signature en aveugle et peut être considérée comme un dérivé du protocole de Fujioka, Okamoto, Ohta (FOO92) [6]. Ce dernier nécessite l'intervention de l'électeur à plusieurs reprises. Il n'est donc pas « vote and go ». En effet, l'électeur pour ne pas révéler son vote va simplement envoyer un engagement. Il doit donc lors d'une première phase obtenir la signature de cet engagement auprès d'une autorité et envoyer cet engagement. Ensuite, une fois cette première phase terminée, il doit faire parvenir une donnée permettant d'ouvrir son engagement.

Or, pour être utilisable en pratique, il est important que l'électeur n'ait pas à intervenir plusieurs fois au cours de la procédure de vote. Pour parer à ce défaut, Ohkubo *et al.* [9] ont modifié le protocole de FOO92. Ils proposent de ne plus utiliser un schéma d'engagement qui aboutit nécessairement à un protocole de vote en deux phases, mais un schéma de chiffrement classique associé à un réseau de mélangeurs. Une implémentation de ce schéma a été réalisée, il s'agit de VOTOPIA [7].

Puis, J. Traoré a mis en évidence une faille (concernant la vérifiabilité) sur VOTOPIA et propose pour « réparer » le protocole, l'utilisation d'un schéma de signature en aveugle à anonymat révocable [10]. C'est ce protocole que nous allons étudier.

4.2 Description du Protocole

Nous allons décrire brièvement les différentes phases du protocole (plus de détails dans [10]). Les différents intervenants sont :

- l'administrateur $\mathcal{AS}$,
- le votant $\mathcal{V}_i$,
- les réseaux de mélangeurs $\mathcal{M}$ et $\mathcal{TM}$,
- le bulletin board $\mathcal{BB}$,
- l'autorité de confiance I.

Phase d'Enregistrement Dans un premier temps, le votant $\mathcal{V}_i$ s'enregistre auprès de l'administrateur $\mathcal{AS}$ pour obtenir un certificat C_i et avoir le droit de participer aux futures élections.

Phase de Vote

1. $\mathcal{V}_i$ contacte $\mathcal{AS}$ qui vérifie si $\mathcal{V}_i$ a le droit de vote et s'il n'a pas déjà voté.
2. $\mathcal{V}_i$ chiffre son vote v_i avec la clef $\mathsf{pub}(\mathcal{TM})$ du réseau de mélangeurs $\mathcal{TM}$. Il obtient $x_i = \{v_i\}_{\mathsf{pub}(\mathcal{TM})}$. Ensuite $\mathcal{V}_i$ cache x_i en calculant $e_i = \mathsf{fairblind}(x_i, r_i)$ où r_i est un nombre aléatoire. Enfin, $\mathcal{V}_i$ signe e_i pour obtenir $s_i = sign(e_i, \mathsf{priv}(\mathcal{V}_i))$. Il envoie $(\mathcal{V}_i, C_i, e_i, s_i)$ à l'administrateur.
3. $\mathcal{AS}$ vérifie que s_i est une signature valide et envoie $d_i = sign(e_i, \mathsf{priv}(\mathcal{AS}))$ à $\mathcal{V}_i$.
4. $\mathcal{V}_i$ obtient la signature y_i de son bulletin x_i en « retirant » son nombre aléatoire r_i, $y_i = \mathsf{unblind}(d_i, r_i)$.
5. $\mathcal{V}_i$ chiffre $b_i = (x_i, y_i)$ avec la clef du réseau de mélangeurs $\mathcal{M}$, $c_i = \{b_i\}_{\mathsf{pub}(\mathcal{M})}$. Soit P_i une preuve de connaissance à divulgation nulle de connaissance du message en clair caché dans c_i (*i.e.* b_i). $\mathcal{V}_i$ signe c_i, $\sigma_i = sign(c_i, \mathsf{priv}(\mathcal{V}_i))$ et envoie $(\mathcal{V}_i, C_i, c_i, \sigma_i, P_i)$ au bulletin board $\mathcal{BB}$ qui vérifie la validité de la signature σ_i et de la preuve P_i.
6. Lors de la clôture des élections, $\mathcal{AS}$ annonce le nombre de participants ayant reçu une signature de l'administrateur et publie la liste finale $L_{\mathcal{AS}}$ des n-ulpets $(\mathcal{V}_i, C_i, e_i, s_i)$. De même $\mathcal{BB}$ publie la liste $L_{\mathcal{BB}}$ des messages postés $(\mathcal{V}_i, C_i, c_i, \sigma_i, P_i)$. Les deux listes sont comparées. Si un votant a obtenu une signature de la part de l'administrateur $\mathcal{AS}$ mais n'a pas posté son bulletin sur le bulletin board, alors I révoque l'anonymat de e_i. Le couple ($\mathsf{revmsg}(e_i$, $sign(e_i, \mathsf{priv}(\mathcal{AS}))$), $\mathsf{revsign}(e_i$,

$sign(e_i, \mathsf{priv}(\mathcal{AS}))))$ est mémorisé sur une liste noire pour que tout le monde soit en mesure de reconnaître le couple message-signature (x_i, y_i) à l'origine de la fraude plus tard. Inversement, si $\mathcal{V}_i$ apparaît dans $L_{\mathcal{BB}}$ mais pas dans $L_{\mathcal{AS}}$, alors $(\mathcal{V}_i, C_i, c_i, \sigma_i, P_i)$ est supprimé de la liste $L_{\mathcal{BB}}$.

Phase de Comptage $\mathcal{BB}$ envoie au réseau de mélangeurs $\mathcal{M}$, la liste L_0 des c_i extraite à partir de $L_{\mathcal{BB}}$. $\mathcal{M}$ déchiffre la liste des c_i, permute aléatoirement la liste des (x_i, y_i) ainsi obtenue et envoie cette liste L_k au réseau de mélangeurs $\mathcal{TM}$.

1. $\mathcal{TM}$ teste si des paires (x_i, y_i) apparaissent deux fois dans la liste L_k.
   – Si de telles paires n'existent pas, on continue au point 2.
   – Sinon, pour chaque paire $(\tilde{x}_g, \tilde{y}_g)$, il faut lancer la procédure de *back tracing*. Les mélangeurs doivent alors fournir une preuve montrant qu'il se sont comportés correctement, sous peine d'être disqualifiés. Si tout les mélangeurs fournissent une telle preuve, alors la procédure de *back tracing* identifie le votant $\mathcal{V}_f$ malhonnête, révèle son identité ainsi que la paire $(\tilde{c}_f, \tilde{\sigma}_f)$ étant à l'origine de $(\tilde{x}_g, \tilde{y}_g)$ dans L_k. L'anonymat est révoqué, le bulletin $(\tilde{x}, \tilde{y})$ correspondant à $(\tilde{c}_f, \tilde{\sigma}_f)$ est obtenu et est inséré dans la liste noire. La paire $(\tilde{x}_g, \tilde{y}_g)$ est supprimée de L_k.
2. $\mathcal{TM}$ teste la validité de la signature y_i pour chacune des paires (x_i, y_i) dans L_k.
   – Si toutes les signatures sont valides, la procédure continue au point 3.
   – Autrement, pour chaque paire (x_i, y_i) incorrecte, il faut déterminer si l'anomalie provient d'un des mélangeurs ou si la fraude provient du votant, à l'aide de la procédure de *back tracing*. Si l'anomalie est due au votant, l'utilisation du mécanisme de signature en aveugle à anonymat révocable permettra de retrouver l'identité du fraudeur.
3. $\mathcal{TM}$ compare la liste L_k avec la liste noire.
   – Si ces deux listes n'ont aucun élément en commun alors $\mathsf{priv}(\mathcal{TM})$ est révélée. Les x_i sont déchiffrés et $\mathcal{TM}$ publie le résultat de l'élection.
   – Sinon, pour chaque paire $(\tilde{x}_g, \tilde{y}_g)$, la procédure de *back tracing* est lancée pour déterminer le fraudeur. La procédure continue au point 3.

La procédure de *back tracing* permet de retrouver le mélangeur à l'origine de l'anomalie ou de retrouver le votant à l'origine de la fraude. Compte tenu du fait que dans notre modélisation, le réseau de mélangeurs est abstrait par un processus unique, nous avons choisi de considérer que les mélangeurs ne pouvaient pas être à l'origine d'une anomalie. En revanche, nous recherchons le votant à l'origine de la fraude.

5 Formalisation dans le Langage PROUVÉ

5.1 La Théorie Équationelle

La théorie équationelle est décrite (en partie) en Figure 1. Les équations permettant de manipuler les clefs publiques, le chiffrement et le mécanisme de signature sont très classiques. Nous avons besoin de modéliser le mécanisme de signature en aveugle à anonymat révocable ainsi que le mécanisme de preuve à divulgation nulle de connaissance (cf. Section 3).

Nous avons également dû ajouter dans la théorie équationelle, les propriétés algébriques d'un certain nombre de fonctions pour permettre la manipulation des listes. Nous pensons que ces fonctionnalités de base (suppression d'un élément d'une liste, ...) devraient être fournies par le langage PROUVÉ.

5.2 Le Protocole

Processus Principal (Process 2). Nous modélisons ici la phase de génération des clefs privées et des clefs publiques correspondantes (makekey). Les différentes listes (liste_of_voters, list_AS, list_BB, listBallot, listVote, blacklist) sont initialisées. Ensuite, on distingue trois phases dans le protocole. Chacune d'entre elles correspond à l'exécution en parallèle de plusieurs processus. Notez bien que même si le protocole se déroule en trois phases, le votant n'intervient que dans la première : ce protocole est donc bien « vote and go ».

```
signature
    # host name and public-key
    host: pubkey → principal;
    getpubkey: principal → pubkey;
    # probabilisitic encryption
    pencrypt:(message,pubkey,nonce) → message;
    decrypt:(message,privkey) → message;
    # blind signature
    fairblind: (message,nonce) → message;
    unblind: (message,nonce) → message;
    revmsg: message → message;
    revsign: (message,pubkey) → message;
    # zero-knowledge proof
    proof: (message, message, pubkey) → message;
    checkproof: (message, message, pubkey) → message;
    ok: message;
end

axioms
    declare h:principal; m:message; r:nonce; privk: privkey; pubk: pubkey;
    begin
        # host name and public-key
        host(getpubkey(h)) = h;
        getpubkey(host(pubk)) = pubk;
        # probabilistic encryption
        decrypt(pencrypt(m,inv(privk),r),privk) = m;
        # blind signature
        unblind(sign(asym,privk,fairblind(m,r)),r)=sign(asym,privk,m);
        revmsg(fairblind(m,r)) = m;
        revsign(fairblind(m,r),pubk) = sign(asym, inv(pubk),m);
        # zero-knowledge proof
        checkproof(proof(pencrypt(m,pubk,r),m,pubk),
        pencrypt(m,pubk,r),
        pubk) = ok;
    end
```

Process 1 – Théorie Équationelle

Au cours de la première phase, le votant, l'administrateur et le Bulletin Board communiquent. A l'issue de cette phase, le votant doit avoir récupéré la signature de son vote auprès de l'administrateur et avoir posté son bulletin sur le Bulletin Board. La deuxième phase est destiné à publier les listes $L_{\mathcal{AS}}$ et $L_{\mathcal{BB}}$. Enfin, au cours d'une troisième phase, ces listes sont comparées et le processus de révocation enclenché. La procédure de *back tracing* est alors exécutée chaque fois que nécessaire pour retrouver l'identité d'un fraudeur.

Votant (Process 3). La phase d'obtention du certificat n'étant pas décrite dans la spécification du protocole, nous supposerons donc que le votant a déjà un certificat en sa possession. Bien évidemment, dans la réalité, ce n'est pas au votant lui-même de fabriquer son propre certificat. La première étape consiste pour le votant à obtenir la signature de son bulletin en interagissant avec l'administrateur. La deuxième étape consiste à poster son bulletin signé accompagné de quelques informations supplémentaires sur le bulletin board.

Administrateur (Process 4). Un des paramètres du rôle administrateur est une liste contenant les clefs publiques des votants légitimes. Lorsque l'administrateur reçoit une demande d'un votant pour l'obtention d'une signature, il vérifie tout d'abord que le votant est légitime, *i.e.* que sa clef publique apparaît bien dans la liste des votants légitimes. Si c'est le cas, l'administrateur fournie la signature au votant, et met à jour la liste $L_{\mathcal{AS}}$ des signatures délivrées par l'administrateur.

Ensuite, lors de la deuxième phase, l'administrateur publiera la liste $L_{\mathcal{AS}}$ des votants auxquels il a fourni une signature.

Bulletin Board (Process 5). Le bulletin board reçoit le bulletin signé accompagné de quelques informations en provenance du votant. Il procède alors à un certain nombre de vérifications (validité de la signature, du certificat, de la preuve à divulgation nulle de connaissance, ...). Si tout se passe bien, le message est alors ajouté à la liste $L_{\mathcal{BB}}$.

Lors de la deuxième phase, le Bulletin Board publiera la liste $L_{\mathcal{BB}}$ des messages ayant été postés sur le bulletin board.

Processus de Révocation (Process 6). Le processus de révocation est chargé de retrouver les votants ayant obtenu une signature auprès de l'administrateur et n'ayant pas poursuivi la procédure de vote. Il doit donc comparer (compare1, compare2) le contenu des listes $L_{\mathcal{AS}}$ et $L_{\mathcal{BB}}$. Si un votant a effectué la première étape du protocole, *i.e.* il se retrouve dans la liste $L_{\mathcal{AS}}$ et n'a pas effectué la deuxième étape, *i.e.* on ne le retrouve pas dans $L_{\mathcal{BB}}$, on doit alors retrouver le bulletin correspondant et le mettre sur une liste noire. Inversement, si un votant a posté un bulletin sur le bulletin board et n'a pas demandé de signature à l'administrateur, le message correspondant doit être supprimé (removelist) de la liste $L_{\mathcal{BB}}$.

Ensuite, la liste L_0 des c_i extraite de $L_{\mathcal{BB}}$ est envoyée au réseau de mélangeurs $\mathcal{M}$. La fonction map3 permet d'extraire le troisième élément de chacun des 5-ulpets de la liste passée en argument.

Réseaux de Mélangeurs (Process 7 et 8). Le premier réseau de mélangeurs permet de déchiffrer et d'obtenir la liste des bulletins. Un bulletin est une paire constituée d'un vote chiffré avec la clef publique du deuxième réseau de mélangeur (c'est le message xt), et de la signature de xt par l'administrateur. La fonction mapDecrypt permet d'appliquer cette opération de déchiffrement à chacun des éléments de la liste passée en paramètre. Ces bulletins sont ensuite publiés dans un ordre aléatoire (permute).

Ensuite, le deuxième réseau de mélangeurs $\mathcal{TM}$ s'occupe des doublons apparaissant dans la liste publiée par le premier réseau de mélangeurs. La procédure de *back tracing* (BTProcedure) est lancée si nécessaire : les listes blacklist et listBallot sont mises à jour. Ensuite, $\mathcal{TM}$ vérifie si les signatures sont valides et lance de nouveau la procédure de *back tracing* si ce n'est pas le cas. Enfin, tant que des éléments de la liste noire se retrouvent dans listBallot, il faut continuer à lancer la procédure de *back tracing* pour retrouver tous les fraudeurs. À l'issue de cette étape, l'intersection des deux listes listBallot et blacklist est vide, les bulletins restants dans listBallot proviennent de votants honnêtes. On peut procéder au dépouillement et à la comptabilisation des votes.

Pour modéliser ce deuxième réseau de mélangeurs, quelques fonctionnalités nous ont manquées :

- l'instruction while ... do ...,
- le fait de pouvoir parcourir une liste et d'exécuter une procédure sur chacun des éléments de cette liste. La plupart du temps, on peut s'en sortir en créant une fonction de type « map ». Mais, il serait plus approprié de disposer d'une instruction du type : for each x in L do ....

La procédure de Back Tracing Il s'agit d'une procédure relativement complexe. Étant donné un couple (xt, yt), elle doit permettre de retrouver la provenance de ce bulletin. Pour cela, il faut interroger le réseau de mélangeurs pour savoir comment il a effectué son mélange et retrouver la trace du bulletin avant son entrée dans le mélangeur. C'est le rôle de la fonction assoc. Ensuite, on retrouve en parcourant $L_{\mathcal{BB}}$, l'identité du votant fraudeur ainsi que le tuple correspondant dans $L_{\mathcal{AS}}$, on peut alors procéder à la révocation. On obtient un couple message-signature que l'on ajoute à la blacklist et on supprime (xt, yt) de la liste listBallot.

5.3 Les Propriétés

Comme nous l'avons vu en Section 2.2, les propriétés demandées pour un protocole de vote sont très particulières et difficile à exprimer.

Anonymat. L'anonymat est généralement exprimé en terme d'équivalences observationelles entre processus. Dans [8], l'anonymat d'un protocole est exprimé de la façon suivante : l'intrus ne peut pas différencier le processus où deux votants votent v_1 et v_2 du processus où ils inversent leur vote. Le langage de propriété de PROUVÉ n'offre malheureusement pas la possibilité d'exprimer des équivalences observationelles entre différents processus.

Secret des votes et résultats préliminaires. Une propriété importante est que les votes restent secrets jusqu'au moment de la publication du résultat. Des résultats préliminaires pourraient sinon influencer la décision des votants n'ayant pas encore voté. Cette propriété s'exprime de façon élégante dans le langage de propriété de PROUVÉ :

$$\boxminus(scenario@start \rightarrow secret(vote)) \wedge mixnet@before\_published \rightarrow secret(vote)$$

Cette propriété dit que si le vote est secret au début du protocole, il le sera aussi au moment qui précède la publication des résultats. Le label *before_published* définit effectivement l'endroit précédant la publication des votes dans le processus 8.

Par contre, dans un protocole de vote, la valeur du vote n'a rien d'un secret. C'est généralement une constante telle que oui ou non. En revanche, les protocoles de vote sont particulièrement vulnérable aux attaques par dictionnaire de part le fait que la valeur du vote est en général une constante à choisir dans un domaine restreint. Intuitivement, une telle attaque consiste pour l'intrus à deviner la valeur choisie par le votant et à utiliser le protocole pour vérifier la véracité d'un tel choix. Pour l'instant PROUVÉ ne permet pas de spécifier ce genre de propriété.

Éligibilité. Cette propriété peut se voir comme une propriété d'atteignabilité et peut se modéliser au moins partiellement en PROUVÉ. En effet, il suffit de donner à l'intrus un *vote challenge*, et de voir si ce vote peut se retrouver dans le résultat final. Malheureusement, le codage de cette propriété entraîne des modifications dans le codage du protocole.

Sans Reçu. Ce type de propriété est relativement complexe à définir. Une première définition formelle est proposée dans [5]. PROUVÉ ne permet pas de modéliser ce type de propriété.

6 Synthèse

Comme nous l'avons déjà remarqué lors de notre première étude de cas, le langage PROUVÉ est relativement complet en particulier en ce qui concerne la description des scénarios. Cette souplesse dans la description du scénario nous a permis de coder le mécanisme de phases en permettant une synchronisation globale entre différents processus. Ce type de synchronisation est primordial pour la modélisation de protocoles de vote électronique. En effet, pour garantir l'anonymat des votes, il faut par exemple assurer que la procédure de publication ne commence pas avant que les votants aient fini leur procédure de vote.

Le principal problème rencontré lors de la modélisation du protocole est lié à la manipulation des listes. Dans le langage PROUVÉ, le type `List` existe mais très peu de fonctionnalités sont offertes. Il est uniquement possible d'ajouter un élément dans une liste (opérateur : :) et de tester si un élément donné est dans la liste (opérateur `element`). Si l'on souhaite utiliser d'autres fonctionnalités telles que la suppression d'un élément d'une liste, le parcours d'une liste en vue de répéter une même opération sur chacun des éléments de la liste,... on doit définir de nouveaux symboles de fonctions et ajouter des équations à la théorie équationnelle. Ceci est possible, mais devient très rapidement extrêmement lourd. Les fonctionnalités les plus intéressantes à la vue de cette étude de cas (et manquant dans le langage) sont une instruction `for each`

x in L do permettant de réaliser un même traitement sur chacun des éléments d'une liste, et l'opération remove pour permettre la suppression d'un élément d'une liste.

Du point de vue de la modélisation des propriétés, il est possible d'en modéliser quelques-unes. Il s'agit des propriétés dites de trace telles que le secret des votes, l'éligibilité. En revanche, le langage PROUVÉ ne permet pas de modéliser les propriétés s'exprimant en terme d'équivalences observationelles (anonymat, sans-reçu).

7 Conclusion

Pour conclure, le langage PROUVÉ s'est révélé être très riche pour certains aspects, mais il n'a pas permis de modéliser la totalité de cette étude de cas. Certaines fonctionnalités concernant la manipulation des listes ne sont pas disponibles dans le langage PROUVÉ et pourraient être ajouter au langage.

En ce qui concerne l'absence de l'instruction while ... do ..., il s'agit d'un choix délibéré. Une telle instruction est en effet complexe à traiter du point de vue de la vérification. Concernant le langage de propriétés, le fait que le langage PROUVÉ ne permette pas d'exprimer l'équivalence observationelle est également un choix délibéré. En effet, nous avons choisi de nous concentrer et de définir un langage pour exprimer les propriétés dites de traces.

Références

[1] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communication of ACM*, 24(2) :84–88, 1981.

[2] D. Chaum. Blind signature system. In P. Press, editor, *Proc. of CRYPTO '83*, page 153, New York (USA), 1984.

[3] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the security of protocols with commuting public key encryption. *ENTCS*, 125(1) :55–66, 2005.

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

[5] S. Delaune, S. Kremer, and M. D. Ryan. Receipt-freeness : Formal definition and fault attacks (extended abstract). In *Proc. Workshop Frontiers in Electronic Elections (FEE'05)*, Milan, Italy, 2005.

[6] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology (AUSCRYPT'92)*, volume 718 of *LNCS*, pages 244–251. Springer, 1992.

[7] K. Kim, J. Kim, B. Lee, and G. Ahn. Experimental design of worldwide internet voting system using PKI. In *SSGRR'01*, L' Aquila (Italy), 2001.

[8] S. Kremer and M. D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 186–200, Edinburgh, U.K., 2005. Springer.

[9] M. Ohkubo, F. Miura, M. Abe, A. Fujioka, and T. Okamoto. An improvement on a practical secret voting scheme. In *Proc. 2nd International Workshop on Information Security (ISW'99)*, volume 1729 of *LNCS*, pages 225–234. Springer, 1999.

[10] J. Traoré. Are blind signatures suitable for on-line voting ? (extended abstract). In *Proc. of Workshop Frontiers in Electronic Elections (FEE'05)*, Milan, Italy, 2005.

```
variables
   voter_key(principal): privkey;
   admin_key, mix_key, tmix_key: privkey;
   list_of_voters: mutable list(pubkey);
   vote(principal): message;
   listAS, listBB, listBallot, listVote: mutable list(message);
   blacklist: mutable list(message);
end


scenario
   begin
        makekey(asym, voter_key);
        makekey(asym, admin_key);
        makekey(asym, mix_key);
        makekey(asym, tmix_key);
        list_of_voters ← [[inv(voter_key)]];
        listAS ← [[]];
        listBB ← [[]];
        listBallot ← [[]];
        listVote ← [[]];
        blacklist ← [[]];
        # phase 1
        parallel # eventuellement plusieurs
            voter(host(inv(voter_key)), voter_key, inv(admin_key),
            inv(mix_key), inv(tmix_key), vote) |
            administrator(admin_key, list_of_voters, listAS) |
            bulletinBoard(inv(admin_key), inv(mix_key), listBB)
        end;
        # phase 2;
        parallel # un seul process de chaque
            publisherListAS(listAS) |
            publisherListBB(listBB)
        end;
        # phase 3; un seul process de chaque
        revocation(blacklist, listAS, listBB);
        publisherList0(listBB);
        mixnetM(mix_key, listBallot);
        mixnetTM(tmix_key, inv(admin_key), listVote, listBallot, blacklist);
    end
end
```

Process 2 – Processus Principal

```
role voter(hostv: principal;skv: privkey; pka,pkm,pktm:pubkey; v: message)
    declare
    r1,r2,r3: nonce;
    pkv: pubkey;
    certv: message;
    x,e,s: message;
    y,c,sigma,p,d: message;
    b: (message, message);
    begin
        # registration stage
        pkv ← inv(skv);
        certv ← sign(asym, inv(pka), [hostv, pkv]);
        # voting stage
        new(r1);new(r2);new(r3);
        x ← pencrypt(v, pktm, r1);
        e ← fairblind(x,r2);
        s ← sign(asym,skv,e);
        send([hostv,certv,e,s]);
        recv(d);
        y ← unblind(d,r2);
        b ← [x, y];
        c ← pencrypt(b,pkm,r3);
        sigma ← sign(asym,skv,c);
        p ← proof(c,b,pkm);
        send([hostv,certv,c,sigma,p]);
    end
```

Process 3 – Processus Votant

```
role administrator(ska: privkey; voters: mutable list(pubkey);
    listAS: mutable list(message))
    declare
    hv: principal;
    ev,dv: message;
    pubkeyv: pubkey;
    begin
        # voting stage
        recv([hv, sign(asym,ska,[hv,pubkeyv]), ev, sign(asym,inv(pubkeyv),ev)]);
        dv ← sign(asym,ska,ev);
        # test if it is a legitimate voter
        if element(pubkeyv, voters)
        then
            voters ← remove(pubkeyv, voters);
            dv ← sign(asym,ska,ev);
            send(dv);
            # add to List_AS and publish List_AS
            listAS ← [hv,
                      sign(asym,ska,[hv,pubkeyv]),
                      ev,
                      sign(asym,inv(pubkeyv),ev)] :: listAS;
        else
            fail;
        fi;
    end


role publisherListAS(listAS: mutable list(message))
    begin
        send(listAS);
    end
```
Process 4 – Processus Administrateur

```
role bulletinBoard(pkadmin,pkmix:pubkey; listBB: mutable list(message))
    declare
    hvoter: principal;
    cb,pb:message;
    pkvoter:pubkey;
    begin
        # voting stage
        recv([hvoter,
           sign(asym,inv(pkadmin),[hvoter,pkvoter]),
           cb,
           sign(asym,inv(pkvoter),cb),
           pb]);
        if checkproof(pb,cb,pkmix) = ok
        then
            # add to List_BB and publish List_BB
            listBB ← [hvoter,
                     sign(asym,inv(pkadmin),[hvoter,pkvoter]),
                     cb,
                     sign(asym,inv(pkvoter),cb),
                     pb] :: listBB;
        else
            fail;
        fi;
    end

role publisherListBB(listBB: mutable list(message))
    begin
        # publish ListBB
        send(listBB);
    end
```

Process 5 – Bulletin Board

```
role revocation(blacklist, listAS, listBB: mutable list(message))
    declare
    list1, list_f, list2: list(message);
    begin
        # compare ListAS et ListBB

        # si element dans AS, et pas d' element correspondant dans BB
        list1 ← compare1(listAS, listBB);
        if list1 = [[]]
        then
            # skip;
        else
            # revocation
            list_f ← revocationlist(list1);
            blacklist ← addlist(list_f, blacklist);
        fi;

        # si element dans BB et pas d' element correspondant dans AS
        list2 ← compare2(listAS, listBB);
        if list2 = [[]]
        then
            # skip
        else
            # remove
            listBB ← removelist(list2, listBB);
        fi;
    end

role publisherList0(listBB: mutable list(message))
    declare
    list0: list(message);
    begin
        # extract L0 (c) from ListBB (id, C, c,sigma,P)
        list0 ← map3(listBB);
        send(list0);
    end
```

Process 6 – Processus de Révocation

```
role mixnetM(skm:privkey; listBallot: mutable list(message))
    declare
    r: nonce;
    list0 , list_b: list(message);
    begin
        # Step 1: counting stage
        recv(list0);
        # decrypt each element of a list
        list_b ← mapDecrypt(list0 ,skm);
        # permute the elements of list_b to obtain list_k
        # Given an element [x,y] of list_k, we can retrieve the
        # corresponding element c of list0 by assoc([x,y],list_k,r)
        new(r);
        listBallot ← permute(list_b ,r);
        # publish ListBallot
        send(listBallot);
end
```
Process 7 – Réseaux de Mélangeurs 1

```
role mixnetTM(sktm:privkey;pka:pubkey;
    listVote, listBallot, blacklist: mutable list(message))
    declare
    z: int;
    x,y:message;
    list_s: list(message);
    list_doublons, list_nonsign, list_inter : list(message);
    begin
        # counting stage
        # Step 2: tests whether there are some duplicated pairs
        list_doublons ← doublons(listBallot);
        if list_doublons = [[]]
        then
            # skip;
        else
            # back tracing procedure for each elements in list_doublons
            # BTP procedure updates blacklist and listBallot
            # for each [x,y] in list_doublons
            z ← BTProcedure(x,y);
        fi;

        # Step 3: tests whether the signatures yi of elements (xi,yi)
        # in ListBallot are valid
        list_nonsign ← mapChecksign(listBallot,pka);
        if list_nonsign = [[]]
        then
            # skip;
        else
            # back tracing process for each elements in list_nonsign
            # updating of black_list and list_k
            # for each [x,y] in list_nonsign
            z ← BTProcedure(x,y);
    fi;

    # Step 4: tests whether or ¬ (black_list \cap listBallot) is empty
    list_inter ← intersection(listBallot,blacklist);

    #while (list_inter ≠ [[]])
    #do {
    # back tracing process for each element on list_inter
    # updating of black_list and list_k
    #for each [x,y] in list_inter
    z ← BTProcedure(x,y);
    #}

    # reveal sktm
    before_published : send(sktm);
    list_s ← map1(listBallot);
    listVote ← mapDecrypt(list_s,sktm);
    send(listVote);
end
```

Process 8 – Réseaux de Mélangeurs 2

```
role BTProcedure(x,y: message)
    declare
    c: message;
    begin
    # retrieve the c corresponding to [x,y] in List0
    # Il faut interroger le MixNet pour obtenir cette information
    c ← assoc([xt, yt], listBallot, r);

    # retrieve the corresponding tuple in listBB
    [id, cert, c, sigma, P] ← assoc([_,_,c,_,_], listBB);
   Listvoterblack ← id :: Lvoterblacklist;

    # retrieve the correponding tuple in ListAS
    [id, cert, e, s] ← assoc([id, cert, _, _], listAS);
    # with  e and thanks to unfairblind signature mechanism,
    # retrieve the "real" ballot
    x ← revmsg(e, fairblind(e, inv(pka)));
    y ← revsign(e, fairblind(e, inv(pka)));
    # add [x,y] in blacklist
    blacklist ← [x,y]:: blacklist;
    # remove [xt,yt] from listBallot
    remove([xt, yt], listBallot);
end
```

Process 9 – Back Tracing Procedure