







RAPPORT TECHNIQUE PROUVÉ

Retour d'Expérience sur la Validation du Vote Électronique

Auteurs : Francis Klay, France Télécom
Liana Bozga, Yassine Lakhnech, Laurent Mazaré, Verimag
Stéphanie Delaune, Steve Kremer, LSV

Date : 16 Novembre 2006

Rapport numéro : 7

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 : Dans ce rapport, nous présentons le travail de vérification qui a été réalisé sur le protocole de vote électronique que nous avons introduit et formalisé dans le rapport [6]. Ce protocole a été mis au point par J. Traoré, ingénieur de recherche chez France Télécom. Il est basé sur le mécanisme de signature en aveugle et peut être considéré comme un dérivé du protocole de Fujioka, Okamoto et Ohta [8].

La formalisation de ce protocole à mis en évidence une grande complexité due en particulier aux structures de données et aux primitives cryptographiques manipulées. D'un autre côté ce travail a également révélé que les propriétés de sûreté à garantir sont particulièrement subtiles. Ce document présente les résultats qui ont été obtenus lors de la vérification de ce protocole. En particulier nous montrons que certaines propriétés de sûreté ont pu être prouvées automatiquement alors que pour d'autres une preuve manuelle s'est avérée nécessaire.

1 Introduction

Dans cette section nous allons dans un premier temps rappeler le protocole de vote dont il est question et les propriétés de sûreté a garantir. Ensuite nous donnerons un aperçu des résultats présentés dans les autres sections. Pour une présentation plus générale sur les protocoles de vote électronique, le lecteur pourra consulter la présentation qui en est faite dans [6]. Dans ce même document le lecteur trouvera également la spécification des notions qui sont présentées de façon informelle dans cette introduction.

Le protocole de J. Traoré fournit par France Télécom est une évolution du protocole de Fujioka, Okamoto, Ohta (FOO92) [8]. 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.* [12] 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 [9].

J. Traoré a mis en évidence une faille (concernant la vérifiabilité) sur VOTOPIA et propose l'utilisation d'un schéma de signature en aveugle à anonymat révocable [13] pour contourner le problème. C'est ce dernier protocole qui est notre cas d'étude.

Outre les schémas classiques de la cryptographie notre cas d'étude met en oeuvre deux notions moins connues que nous rappelons ci-dessous.

Le schéma de signature en aveugle introduit par D. Chaum [4]. Ce schéma permet à une entité d'obtenir d'une autre entité la signature d'un message sans que le signataire ne connaisse son contenu. 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 cas d'étude utilise une évolution de ce schéma appelée *signature en aveugle à anonymat révocable*. Ce schéma permet à l'aide d'une autorité compétente (appelée juge) de 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.

Les réseaux de mélangeurs introduit par D. Chaum [3]. Un mélangeur est une boîte noire qui simule une permutation aléatoire. Prenant en entrée des données son but est de cacher la correspondance entre ces données et celles produites en sortie. Lors de l'utilisation de plusieurs mélangeurs en série, on parle de réseaux de mélangeurs. Un réseau de mélangeur permet de réaliser un canal anonyme et c'est ce point qui est à la base de son utilisation dans certains protocoles de vote.

1.1 Propriétés de sûreté des protocoles de vote

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.

Un caractéristiques des protocoles de vote est que les propriétés de sûreté a garantir sont subtiles, parfois elle peuvent même sembler contradictoire. 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 !

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).

1.2 Description du Protocole

Nous allons décrire brièvement les différentes phases du protocole (plus de détails dans [13]). 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 $\mathcal{I}$.

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 $\mathcal{I}$ 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.

1.3 Résultats obtenus

Lors de la création du projet Prouvéle but était la prise en compte de protocoles dans la veine de ceux que l'on peut trouver dans [5], c'est à dire des protocoles dont les propriétés de sûreté peuvent en général s'exprimer comme des invariants de trace ou d'état. Notre premier cas d'étude qui était un porte-monnaie électronique appartenait à cette classe de protocoles et il a pu être traité à l'aide des outils du projet. Plutôt que de répéter une expérimentation de même nature, les motivations du second cas d'étude étaient très différentes. L'objectif était d'apporter des réponses aux questions ci-dessous pour le vote électronique qui est l'un des protocoles les plus complexes existant à ce jour.

1. Est-il possible de formaliser un protocole de cette nature et ses propriétés de sûreté dans le langage Prouvé ?

2. Est-ce que les outils automatiques développées dans le cadre du projet sont une aide pour l'analyse de propriétés de sûreté de ce protocole ?
3. Est-ce que des propriétés de sûreté aussi subtiles que celles de ce protocole sont formellement vérifiables ?
4. Pour appréhender des protocoles aussi complexes quelles sont les carences des outils du projet et plus généralement des outils existants ?
5. Comment les techniques et les outils existants doivent ils évoluer pour mécaniser le traitement de tels protocoles.

Pourvoir répondre à ces questions est crucial car la moindre faille pourrait permettre la réalisation d'une fraude à grande échelle, d'un autre côté les polémiques survenues lors des récentes élections ont montré qu'il s'agit d'un véritable sujet de société.

Le rapport [6] a répondu à la question 1 et le rapport de synthèse des expérimentations répondra aux questions 4 et 5. Dans la section 2 de ce rapport nous répondons par l'affirmative à la question 2 en montant que le secret faible du vote peut être vérifié automatiquement avec un outil du projet. Dans la section 3 nous répondons à la question 3 en montrant que des propriétés de sûreté autres que des invariants de trace ou d'état peuvent être prouvées formellement dans le modèle symbolique (i.e. le modèle de Dolev-Yao [7]). Dans la section 4 nous apportons également une réponse à la question 3 en montrant que pour des objets aussi complexes que des mélangeurs, il est possible de prouver formellement des propriétés telles que l'anonymat au niveau du modèle calculatoire. Finalement dans la conclusion nous donnons des éléments de réponse à la question 4, éléments qui seront développés et mis en relief dans le rapport final de synthèse sur les expérimentations.

2 Analyse automatique du secret faible du vote

Dans cette section nous présentons une preuve automatique du secret faible du vote à la fin de la première phase du protocole. Le problème a été formalisé dans le langage Prouvé et la preuve a été réalisée avec l'outil Hermès 2 du projet. Hermès 2 et une refonte de Hermès qui entre autres permet la prise en compte des propriétés algébriques des primitives cryptographiques. Cette preuve a pu être réalisée pour un nombre non borné de sessions et d'acteurs grâce au mécanisme d'abstraction inclu dans l'outil.

Le lecteur pourra trouver dans l'annexe C la spécification complète du protocole qui a été fourni à l'outil. La plus grande partie de cette spécification est naturelle et n'appelle pas de commentaire. Cependant dans ce qui suit nous détaillons certains points qui sont délicat ou intéressant. Par rapport au protocole original la principale divergence concerne les listes qui n'ont pas été modélisées. 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 et de tester si un élément donné est dans la liste. De plus, d'un point de vue vérification les outils du projets ne permettent pas la prise en compte des fonctionnalités qui seraient nécessaires pour traiter les listes.

2.1 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 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.

Le shéma proposé dans le rapport [6] était le 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. Le listing 1 montre le codage de la signature en aveugle à anonymat révocable que nous avons utilisé pour cette expérimentation. Cette modélisation diffère quelque peu de la spécification ci-dessus suite à des restrictions de l'outil sur les théories équationnelles cependant la sémantique reste comparable :

- `h(x,y)` représente le contenu caché qui doit être signé par l'administrateur et `h` est un symbole libre qui code l'opération de masquage. `h(x,y)` correspond à l'expression `fairblind(x,y)` de la spécification originale.
- `sign(asym, inv(PK(AS)), h(x,y))` correspond au contenu caché, signé avec la clé privée `inv(PK(AS))` de l'administrateur.
- `symcrypt(sym, y, sign(asym, inv(PK(AS)), x))` code la notion de révocation : si on dispose du contenu signé et du secret `y` (le secret du juge) alors il est possible de déduire le contenu en clair. Ce qui est dit ici correspond à l'équation `equation revmsg(fairblind(x,y), sign(fairblind(x,y),z)) = x` de la spéficification originale.
- L'opérateur `revsign` n'a pas pu être codé mais son intérêt est mineur pour cette étude.

```
axioms
    declare
    x: message;
    y: symkey;
    begin
        sign(asym, inv(PK(AS)), h(x,y)) =
        symcrypt(sym, y, sign(asym, inv(PK(AS)), x));
    end
```

Listing 1 – Signature en aveugle à anonymat révocable

2.2 Scénario

Le scénario est la partie de la spécification qui initialise les différents rôles ou en d'autres termes qui définit pour quel schéma d'exécution la propriété de sûreté est vérifiée. Le listing 2 indique qu'un nombre arbitraire de rôles de chaque type sont lancés en parallèle. De plus :

- `administrator (inv(PK(AS)))` signifie que tous les administrateurs partagent le même identifiant et la même clé privée.
- `exists v : principal` signifie que le votant peut être n'importe quel rôle, ceci rend possible des cas où par exemple le votant est un administrateur.
- `voter (v,inv(PK(v)),AS,TM,M, sign(asym,inv(PK(AS)),[v,PK(v)]))` indique que chaque votant reçoit lors de son initialisation : son identifiant, sa clé privée, l'identifiant de l'administrateur ainsi que ceux des mélangeurs et son certificat.

```
scenario
        parallel
            forall i : int . administrator (inv(PK(AS)))
            |forall i : int .   bulletinBoard (AS)
            |forall i : int .
             exists v : principal .
             voter (v,inv(PK(v)),AS,TM,M,
                       sign(asym,inv(PK(AS)),[v,PK(v)]))
        end
end
```

Listing 2 – Scénario

2.3 Propriété de sûreté

La propriété qui a été vérifiée est le secret faible du vote, elle est décrite dans le listing 3.

```
public
 V1, TM, AS, M, intruder,  PK

initial
  x_M = [[inv(PK(intruder))]]

always

forall P:voter . ( P.myname ≠ intruder ) → issecret(P.vote)
```

Listing 3 – Secret faible

- `x_M = [[inv(PK(intruder))]]` indique que l'intrus dispose d'une clé privée reconnue par le protocole de vote. Cette clé lui permet de jouer le rôle de votant.
- `forall P :voter.(P.myname != intruder) -> issecret(P.vote)` est la propriété elle même. Préfixée par `always` elle stipule qu'a tout moment si le votant n'est pas l'intrus alors son vote est secret.

3 Analyse du protocole de vote de Fujioka, Okamoto et Ohta

Dans la section 2 nous avons présenté une preuve automatique du secret faible du vote. Une telle preuve ne permet pas d'exclure des attaques par dictionnaire ou une divulgation partielle du secret. Le lecteur trouvera dans l'annexe A un article [10] publié dans le cadre du projet. Ce document montre sur un protocole plus simple que celui de notre cas d'étude que le secret fort peut être vérifié automatiquement. Le protocole traité est celui de Fujioka, Okamoto, Ohta [8] et l'outil utilisé est ProVerif [2]. Outre le secret fort, l'éligibilité à également pu être prouvée automatiquement par contre pour l'anonymat une preuve manuelle a été nécessaire.

4 Étude calculatoire du schéma de vote de Chaum

Le réseaux de mélangeurs est une pièce importante de notre cas d'étude car c'est l'élément qui est à la base de l'anonymat. Le problème est que sa complexité est telle qu'actuellement et à notre connaissance il n'a jamais été traité dans les preuves formelles des protocoles de vote.

Les mélangeurs de notre cas d'étude sont les plus difficiles a prendre en compte car on peut les voir comme des mélangeurs optimistes. Ceci signifie qu'une procédure de détection de fraude potentielle (le backtracing) n'est lancée que s'il y a une suspicion de fraude. Dans le cadre de ce projet un travail a été réalisé pour débroussailler le terrain dans ce sens. Le lecteur pourra le trouver dans le rapport [11] de l'annexe B une étude du schéma de vote de Chaum qui est en fait un réseau de mélangeurs. Ce document est intéressant à plusieurs titres, d'un côté il prouve dans le modèle symbolique l'anonymat de vote pour ce schéma mais en plus il donne des conditions sur les primitives cryptographiques pour que ce résultat reste valide dans le modèle caculatoire.

5 Conclusions

Le protocole de vote de notre cas d'étude n'a pas pu être traité d'un seul tenant et dans un cadre uniforme. Ceci n'a rien d'étonnant et c'était prévu dès le départ du projet. Contrairement au premier cas d'étude (porte-monnaie électronique) qui était un problème adapté aux outils du projet ici le but était très différent : évaluer les carences et les évolutions potentielles des techniques développées au sein du projet sur un des protocoles les plus complexes.

Globalement les résultats sont très positifs. Ils montrent que même sur un protocole complexe les outils de preuve automatique apportent une aide considérable sur certains aspects comme la preuve de secret du vote ou de l'éligibilité. D'un autre côté la plupart des propriétés de sûreté qui sortent du spectres des outils existants ont pu être prouvées manuellement mais formellement. Ce dernier point est important car ce travail manuel a permis de mieux apprécier les méthodes qui devront être mises en oeuvre demain pour mécaniser les preuves qui ont été réalisées.

Un premier grand enseignement de ce travail est que la notion d'équivalence observationnelle est une pièce maîtresse pour appréhender des protocoles aussi complexes. Intuitivement cette notion signifie que quoi que fasse un intrus il sera incapable de distinguer deux exécutions. Il s'agit d'une propriété de base fondamentale car elle permet de coder la plupart des propriétés de notre protocole. Aujourd'hui le seul outil d'analyse de protocoles cryptographiques capable de prendre en compte cette notion est ProVerif [2]. Cependant la technique utilisée impose des limitations qui ne permettent par de traiter des propriétés comme l'anonymat dans notre cas.

Le second grand enseignement concerne l'importance du modèle calculatoire. Ce point est mis en relief dans le travail réalisé autour des réseaux de mélangeurs. Intuitivement et dans un cas extrême le problème est le suivant : dans le modèle symbolique il peut-être impossible d'invalider une propriété de sûreté alors que dans le modèle calculatoire elle peut-être invalidée avec une probabilité de 99%. Cette remarque ne signifie pas que le modèle symbolique soit inutile bien au contraire, c'est dans ce modèle que les preuves ont le plus de chance de pouvoir être automatisées. Ce que signifie cette remarque c'est qu'il n'est pas possible de travailler dans le modèle symbolique en laissant de côté le modèle calculatoire. Quand un cryptologue rédige une preuve manuelle il passe perpétuellement d'un modèle à l'autre selon les aspects qu'il doit traiter. Ceci signifie que les outils a venir devrons être capable de faire de même, pour obtenir ce résultat le principe le plus prometteur a vu le jour en 2000 dans [1]. L'idée est de considérer le modèle symbolique comme une abstraction du modèle calculatoire et de prouver que cette abstraction est sûre. Depuis cet article fondateur ce domaine a donné lieu à une recherche intense dont font partie beaucoup travaux réalisés dans ce projet.

Actuellement une réflexion est en cours au sein du projet suite à cette expérimentation. Son but est de définir qu'elles sont les voies le plus prometteuses pour traiter des protocoles aussi complexes que celui du vote. Le résultat de ce travail sera présenté dans le rapport de synthèse sur les expérimentations.

Références

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness o f formal encryption). In *IFIP International Conference on Theoretical Computer*

Science (IFIP TCS2000), Sendai, Japan, 2000. Springer-Verlag, Berlin Germany.

[2] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In S. Schneider, editor, *14th IEEE Computer Security Foundations Workshop*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society Press.

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

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

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

[6] S. Delaune, F. Klay, and S. Kremer. Spécification du protocole de vote électronique. Rapport Technique 6 du projet RNTL PROUVÉ, Nov. 2005. 19 pages.

[7] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2) :198–208, 1983.

[8] 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.

[9] 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.

[10] 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.

[11] Y. Lakhnech and L. Mazaré. Probabilistic opacity for a passive adversary and its application to chaumś voting scheme. Technical Report 4, Verimag, 2005.

[12] 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.

[13] 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.

A Analyse du protocole de vote de Fujioka, Okamoto et Ohta

Analysis of an Electronic Voting Protocol in the Applied Pi Calculus

Steve Kremer[1] and Mark Ryan[2]

[1] Laboratoire Spécification et Vérification
CNRS UMR 8643 & INRIA Futurs projet SECSI & ENS Cachan, France
kremer@lsv.ens-cachan.fr
[2] School of Computer Science
University of Birmingham, UK
M.D.Ryan@cs.bham.ac.uk

Abstract. Electronic voting promises the possibility of a convenient, efficient and secure facility for recording and tallying votes in an election. Recently highlighted inadequacies of implemented systems have demonstrated the importance of formally verifying the underlying voting protocols. The applied pi calculus is a formalism for modelling such protocols, and allows us to verify properties by using automatic tools, and to rely on manual proof techniques for cases that automatic tools are unable to handle. We model a known protocol for elections known as FOO 92 in the applied pi calculus, and we formalise three of its expected properties, namely fairness, eligibility, and privacy. We use the ProVerif tool to prove that the first two properties are satisfied. In the case of the third property, ProVerif is unable to prove it directly, because its ability to prove observational equivalence between processes is not complete. We provide a manual proof of the required equivalence.

1 Introduction

Electronic voting promises the possibility of a convenient, efficient and secure facility for recording and tallying votes. It can be used for a variety of types of elections, from small committees or on-line communities through to full-scale national elections. However, the electronic voting machines used in recent US elections have been fraught with problems. Recent work [13] has analysed the source code of the machines sold by the second largest and fastest-growing vendor, which are in use in 37 US states. This analysis has produced a catalogue of vulnerabilities and possible attacks.

A potentially much more secure system could be implemented, based on formal protocols that specify the messages sent between the voters and administrators. Such protocols have been studied for several decades. They offer the possibility of abstract analysis of the protocol against formally-stated properties. There are two main kinds of protocol proposed for electronic voting [16]. In blind signature schemes, the voter first obtains a token, which is a message blindly signed by the administrator and known only to the voter herself. She later sends her vote anonymously, with this token as proof of eligibility. In schemes using homomorphic encryption, the voter cooperates with the administrator in order to construct an encryption of her vote. The administrator then

exploits homomorphic properties of the encryption algorithm to compute the encrypted tally directly from the encrypted votes.

Among the properties which electronic voting protocols may satisfy are the following:

Fairness: no early results can be obtained which could influence the remaining voters.
Eligibility: only legitimate voters can vote, and only once.
Privacy: the fact that a particular voted in a particular way is not revealed to anyone.
Individual verifiability: a voter can verify that her vote was really counted.
Universal verifiability: the published outcome really is the sum of all the votes.
Receipt-freeness: a voter cannot prove that she voted in a certain way (this is important to protect voters from coercion).

In this paper, we study a protocol commonly known as the FOO 92 scheme [12], which works with blind signatures. By informal analysis (e.g., [16]), it has been concluded that FOO 92 satisfies the first four properties in the list above.

Because security protocols are notoriously difficult to design and analyse, formal verification techniques are particularly important. In several cases, protocols which were thought to be correct for several years have, by means of formal verification techniques, been discovered to have major flaws [14, 6]. Our aim in this paper is to use verification techniques to analyse the FOO 92 protocol. We model it in the applied pi calculus [3], which has the advantages of being based on well-understood concepts. The applied pi calculus has a family of proof techniques which we can use, is supported by the ProVerif tool [4], and has been used to analyse a variety of security protocols [1, 11].

2 The FOO 92 protocol

The protocol involves voters, an administrator, verifying that only eligible voters can cast votes, and a collector, collecting and publishing the votes. In comparison with authentication protocols, the protocol also uses some unusual cryptographic primitives, such as secure bit-commitment and blind signatures. Moreover, it relies on anonymous channels.

In a first phase, the voter gets a signature on a commitment to his vote from the administrator. To ensure privacy, blind signatures [7] are used, i.e. the administrator does not learn the commitment of the vote.

- Voter V selects a vote v and computes the commitment $x = \xi(v,r)$ using the commitment scheme ξ and a random key r;
- V computes the message $e = \chi(x, b)$ using a blinding function χ and a random blinding factor b;
- V digitally signs e and sends his signature $\sigma_V(e)$ to the administrator A together with his identity;
- A verifies that V has the right to vote, has not voted yet and that the signature is valid; if all these tests hold, A digitally signs e and sends his signature $\sigma_A(e)$ to V;
- V now *unblinds* $\sigma_A(e)$ and obtains $y = \sigma_A(x)$, i.e. a signed commitment to V's vote.

The second phase of the protocol is the actual voting phase.

- V sends y, A's signature on the commitment to V's vote, to the collector C using an anonymous channel;
- C checks correctness of the signature y and, if the test succeeds, enters (ℓ, x, y) onto a list as an ℓ-th item.

The last phase of the voting protocol starts, once the collector decides that he received all votes, e. g. after a fixed deadline. In this phase the voters reveal the random key r which allows C to open the votes and publish them.

- C publishes the list (ℓ_i, x_i, y_i) of commitments he obtained;
- V verifies that his commitment is in the list and sends ℓ, r to C via an anonymous channel;
- C opens the ℓ-th ballot using the random r and publishes the vote v.

Note that we need to separate the voting phase into a commitment phase and an opening phase to avoid releasing partial results of the election.

3 The applied pi calculus

The applied pi calculus [3] is a language for describing concurrent processes and their interactions. It is based on the pi calculus, but is intended to be less pure and therefore more convenient to use. Properties of processes described in the applied pi calculus can be proved by employing manual techniques [3], or by automated tools such as ProVerif [4]. As well as reachability properties which are typical of model checking tools, ProVerif can in some cases prove that processes are observationally equivalent [5]. This capability is important for privacy-type properties such as those we study here. The applied pi calculus has been used to study a variety of security protocols, such as those for private authentication [11] and for fast key establishment [1].

To describe processes in the applied pi calculus, one starts with a set of *names* (which are used to name communication channels or other constants), a set of *variables*, and a *signature* Σ which consists of the function symbols which will be used to define terms.

In the applied pi calculus, one has (plain) processes and extended processes. Plain processes are built up in a similar way to processes in the pi calculus, except that messages can contain terms (rather than just names). Extended processes can also be *active substitutions*: $\{^M/_x\}$ is the substitution that replaces the variable x with the term M. Active substitutions generalise "let". The process $\nu x.(\{^M/_x\} \mid P)$ corresponds exactly to "let $x = M$ in P".

Active substitutions are useful because they allow us to map an extended process A to its *frame* $\phi(A)$ by replacing every plain processes in A with 0. A frame is an extended process built up from 0 and active substitutions by parallel composition and restriction. The frame $\phi(A)$ can be viewed as an approximation of A that accounts for the static knowledge A exposes to its environment, but not A's dynamic behaviour.

The operational semantics of processes in the applied pi calculus is defined by structural rules defining two relations: *structural equivalence*, noted $\equiv$, and *internal reduction*, noted $\rightarrow$. A context $C[\cdot]$ is a process with a hole; an evaluation context is a context

whose hole is not under a replication, a conditional, an input, or an output. Structural equivalence is is the smallest equivalence relation on extended processes that is closed under α-conversion on names and variables, by application of evaluation contexts, and satisfying some further basic structural rules such as $A \mid 0 \equiv A$, associativity and commutativity of $|$, binding-operator-like behaviour of ν, and when $\Sigma \vdash M = N$ the equivalences:

$$\nu x.\{^M/_x\} \equiv 0 \qquad \{^M/_x\} \mid A \equiv \{^M/_x\} \mid A\{^M/_x\} \qquad \{^M/_x\} \equiv \{^N/_x\}$$

Internal reduction $\rightarrow$ is the smallest relation on extended processes closed under structural equivalence such that $\bar{a}\langle x\rangle.P \mid a(x).Q \rightarrow P \mid Q$ and whenever $\Sigma \nvdash M = N$,

$$\text{if } M = M \text{ then } P \text{ else } Q \rightarrow P \qquad \text{if } M = N \text{ then } P \text{ else } Q \rightarrow Q.$$

Many properties of security protocols (including some of the properties we study in this paper) are formalised in terms of *observational equivalence* between processes. To define this, we write $A \Downarrow a$ when A can send a message on a, that is, when $A \rightarrow^* C[\bar{a}\langle M\rangle.P]$ for some evaluation context C that does not bind a.

Definition 1. *Observational equivalence ($\approx$) is the largest symmetric relation R between closed extended processes with the same domain such that $A\ R\ B$ implies:*

1. *if $A \Downarrow a$ then $B \Downarrow a$.*
2. *if $A \rightarrow^* A'$ then $B \rightarrow^* B'$ and $A'\ R\ B'$ for some B'.*
3. *$C[A]\ R\ C[B]$ for closing evaluation contexts C.*

In cases in which the two processes differ only by the terms they contain, if they are also observationally equivalent then ProVerif may be able to prove it directly. However, ProVerif's ability to prove observational equivalence is incomplete, and therefore sometimes one has to resort to manual methods, whose justifications are contained in [3].

The method we use in this paper relies on two further notions: *static equivalence* ($\approx_s$), and *labelled bisimilarity* ($\approx_l$). Static equivalence just compares the static knowledge processes expose to their environment. Two frames are statically equivalent if, when considered as substitutions, they agree on the distinguishability of terms. For frames, static equivalence agrees with observational equivalence, while for general extended processes, observational equivalence is finer.

The definition of *labelled* bisimilarity is like the usual definition of bisimilarity, except that at each step in the unravelled definition one additionally requires that the processes are statically equivalent. Labelled bisimilarity and observational equivalence coincide [3]. Therefore, to prove observational equivalence, it is sufficient to prove bisimilarity and static equivalence at each step. This is what we do to prove the privacy property.

4 Modelling FOO 92 in the applied pi calculus

4.1 Model

We use the applied pi calculus to model the FOO 92 protocol. The advantage is that we can combine powerful (hand) proof techniques from the applied pi calculus with automated proofs provided by Blanchet's ProVerif tool. Moreover, the verification is not

```
(* Signature *)
fun commit/2     (* bit commitment *)
fun open/2       (* open bit commitment *)
fun sign/2       (* digital signature *)
fun checksign/2  (* open digital signature *)
fun pk/1         (* get public key from private key *)
fun host/1       (* get host from public key *)
fun getpk/1      (* get public key from host *)
fun blind/2      (* blinding *)
fun unblind/2    (* undo blinding *)

(* Equational theory *)
equation open(commit(m,r),r) = m
equation getpk(host(pubkey))=pubkey
equation checksign(sign(m,sk),pk(sk)) = m
equation unblind(blind(m,r),r) = m
equation unblind(sign(blind(m,r),sk),r) = sign(m,sk)
```

Process 1. signature and equational theory

restricted to a bounded number of sessions and we do not need to explicitly define the adversary. We only give the equational theory describing the intruder theory. Generally, the intruder has access to any message sent on a public, i.e. unrestricted, channel. These public channels model the network. Note that all channels are anonymous in the applied pi calculus. Unless the identity or something like the IP address is specified explicitly in the conveyed message, the origin of a message is unknown. This abstraction of a real network is very appealing, as it avoids having us to model explicitly an anonymiser service. However, we stress that a real implementation needs to treat anonymous channels with care.

Most of our proofs rely directly on Blanchet's ProVerif tool. The input for the tool is given in an ascii version of the applied pi calculus. To be as precise as possible, the processes described below are directly extracted out of the input files and are given in a pretty-printed version of the ascii input. The minor changes with the usual applied pi calculus notation should be clear.

4.2 Signature and equational theory

The signature and equational theory are represented in Process 1. We model cryptography in a Dolev-Yao style as being perfect. In this model we can note that bit commitment (modeled by the functions commit and open) is identical to classical symmetric-key encryption. The functions and equations that handle public keys and hostnames should be clear. Digital signatures are modeled as being signatures with message recovery, i.e. the signature itself contains the signed message which can be extracted using the checksign function. To model blind signatures we add a pair of functions blind and unblind. These functions are again similar to perfect symmetric key encryption and bit commitment. However, we add a second equation which permits us to extract a signature out of

```
process
    ν ska. ν skv. (* private keys *)
    ν privCh. (* channel for registering legimitate voters *)
    let pka=pk(ska) in
    let hosta = host(pka) in
    let pkv=pk(skv) in
    let hostv=host(pkv) in
    (* publish host names and public keys *)
    out(ch,pka). out(ch,hosta).
    out(ch,pkv). out(ch,hostv).
    (* register legimitate voters *)
    ((out(privCh,pkv). out(privCh,pk(ski))) |
    (!processV)|(!processA)|(!processC))
```

Process 2. environment process

a blinded signature, when the blinding factor is known. The ProVerif tool also implicitly handles pairing: pair(x,y) is abbreviated as (x,y). We also consider the functions fst and snd to extract the first, respectively second element of a pair. Note that because of the property that unblind(sign(blind(m,r),sk),r) = sign(unblind(blind(m,r),r),sk) = sign(m,sk), our theory is not a subterm theory. Therefore the results for deciding static equivalence from [2] do not apply. However, an extension of [2] presents new results that seem to cover a more general family of theories, including the one considered here [9].

4.3 The environment process

The main process is specified in Process 2. Here we model the environment and specify how the other processes (detailed below) are combined. First, fresh secret keys for the voters and the administrator are generated using the restriction operator. For simplicity, all legitimate voters share the same secret key in our model (and therefore the same public key). The public keys and hostnames corresponding to the secret keys are then sent on a public channels, i.e. they are made available to the intruder. The list of legitimate voters is modeled by sending the public key of the voters to the administrator on a private communication channel. We also register the intruder as being a legitimate voter by sending his public key pk(ski) where ski is a free variable: this enables the intruder to introduce votes of his choice and models that some voters may be corrupted. Then we combine an unbounded number of each of the processes (voter, administrator and collector). An unbounded number of administrators and collectors models that these processes are servers, creating a separate instance of the server process (e.g. by "forking") for each client.

4.4 The voter process

The voter process given in Process 3 models the role of a voter. At the beginning two fresh random numbers are generated for blinding, respectively bit commitment of the

```
let processV =
    ν blinder. ν r.
    let blindedcommitedvote=blind(commit(v,r),blinder) in
    out(ch,(hostv,sign(blindedcommitedvote,skv))).
    in(ch,m2).
    let blindedcommitedvote0=checksign(m2,pka) in
    if blindedcommitedvote0=blindedcommitedvote then
    let signedcommitedvote=unblind(m2,blinder) in
    phase 1.
    out(ch,signedcommitedvote).
    in(ch,(l,=signedcommitedvote)).
    phase 2.
    out(ch,(l,r))
```

Process 3. voter process

```
let processA =
    in(privCh,pubkv). (* register legimitate voters *)
    in(ch,m1).
    let (hv,sig)=m1 in
    let pubkeyv=getpk(hv) in
    if pubkeyv = pubkv then
    out(ch,sign(checksign(sig,pubkeyv),ska))
```

Process 4. administrator process

vote. Note that the vote is not modeled as a fresh nonce. This is because generally the domain of values of the votes are known. For instance this domain could be $\{yes, no\}$, a finite number of candidates, etc. Hence, vulnerability to guessing attacks is an important topic. We will discuss this issue in more detail in section 5. The remainder of the specification follows directly the informal description given in section 2. The command in(ch,(l,=s)) means the process inputs not any pair but a pair whose second argument is s. Note that we use phase separation commands, introduced by the ProVerif tool as global synchronization commands. The process first executes all instructions of a given phase before moving to the next phase. The separation of the protocol in phases is useful when analyzing fairness and the synchronization is even crucial for privacy to hold.

4.5 The administrator process

The administrator is modeled by the process represented in Process 4. In order to verify that a voter is a legitimate voter, the administrator first receives a public key on a private channel. Legitimate voters have been registered on this private channel in the environment process described above. The received public key has to match the voter who is trying to get a signed ballot from the administrator. If the public key indeed matches, then the administrator signs the received message which he supposes to be a blinded ballot.

```
let processC =
    phase 1.
    in(ch,m3).
    ν l. out(ch,(l,m3)).
    phase 2.
    in (ch,(=l,rand)).
    let voteV=open(checksign(m3,pka),rand) in
    out(ch,voteV)
```

Process 5. collector process

4.6 The collector process

In Process 5 we model the collector. When the collector receives a committed vote, he associates a fresh label 'l' with this vote. Publishing the list of votes and labels is modeled by sending those values on a public channel. Then the voter can send back the random number which served as a key in the commitment scheme together with the label. The collector receives the key matching the label and opens the vote which he then publishes. Note that in this model the collector immediately publishes the vote without waiting that all voters have committed to their vote. In order to verify in section 5 that no early votes can be revealed we simply omit the last steps in the voter and collector process corresponding to the opening and publishing of the results.

5 Analysis

We have analysed three major properties of electronic voting protocols: fairness, eligibility and privacy. Most of the properties can be directly verified using ProVerif. The tool allows us to verify standard secrecy properties as well as resistance against guessing attacks, defined in terms of equivalences. For all but one property, privacy, the tool directly succeeds its proofs. When analysing privacy, we need to rely on the proof techniques introduced in [3]. Although the results are positive results, we believe that the way we verify the properties increases the understanding of the properties themselves and also the way to model them.

5.1 Fairness

Fairness is the property that ensures that no early results can be obtained and influence the vote. Of course, when we state that no early results can be obtained, we mean that the protocol does not leak any votes before the opening phase. It is impossible to prevent "exit polls", i.e. people revealing their vote when asked.

We model fairness as a secrecy property: it should be impossible for an attacker to learn a vote before the opening phase, i.e. before the beginning of phase 2.

Standard secrecy. Checking *standard* secrecy, i.e. secrecy based on reachability, is the most basic property ProVerif can check. We request ProVerif to check that the private free variable v representing the vote cannot be deduced by the attacker. ProVerif directly succeeds to prove this result.

Resistance against guessing attacks. In the previous paragraph we deduce that a *standard* attacker cannot learn a legitimate voter's vote. However, voting protocols are particularly vulnerable to *guessing attacks* because the values of the votes are taken from a small domain of possible values. Intuitively, in a guessing attack, an attacker *guesses* a possible value for the secret vote and then tries to verify his guess. A trivial example of a guessing attack is when the voter encrypts his vote with the collector's public key (using deterministic encryption). Then the attacker just needs to encrypt his guess and compare the result with the observed encrypted vote. Guessing attacks have been formalized by Lowe [15] and later by Delaune and Jacquemard [10]. A definition in terms of equivalences has been proposed by Corin et al. in [8]:

Definition 2. *Let ϕ be a frame in which v is free. Then we say that ϕ verifies a guess of v if $\phi \not\approx_s \nu v.\phi$. Conversely, we say that ϕ is secure wrt v if $\phi \approx_s \nu v.\phi$.*

Intuitively, if ϕ and $\nu v.\phi$ can be distinguished then an adversary can verify his guess using ϕ. This is also the definition checked by ProVerif. ProVerif succeeds in proving this stronger version of secrecy for the commitment phase of the FOO 92 protocol. Note that verification of guessing attacks does not support considering the protocol up to a given phase. Therefore, we slightly change the processes presented in section 4: we omit the last sending of the voter process which allows the opening of the commitment.

Strong secrecy. We also verified *strong secrecy* in the sense of [5]. Intuitively, strong secrecy is verified if the intruder cannot distinguish between two processes where the secret changes. For the precise definition, we refer the reader to [5]. The main difference with guessing attacks is that strong secrecy relies on observational equivalence rather than static equivalence. ProVerif directly succeeds to prove strong secrecy.

Corrupt administrator. We have also verified standard secrecy, resistance against guessing attacks and strong secrecy in the presence of a corrupt administrator. A corrupt administrator is modeled by outputting the administrator's secret key on a public channel. Hence, the intruder can perform any actions the administrator could have done. Again, the result is positive: the administrator cannot learn the votes of a honest voter, before the committed votes are opened. Note that we do not need to model a corrupt collector, as the collector never uses his secret key, i.e. the collector could anyway be replaced by the attacker.

5.2 Eligibility

Eligibility is the property verifying that only legitimate voters can vote, and only once. The way we verify the first part of this property is by giving the attacker a *challenge vote*. We modify the processes in two ways: (i) the attacker is not registered as a legitimate voter; (ii) the collector tests whether the received vote is the challenge vote and

```
let processC =
    phase 1.
    in(ch,m3).
    ν l. out(ch,(l,m3)).
    phase 2.
    in (ch,(=l,rand)).
    let voteV=open(checksign(m3,pka),rand) in
    ν attack.
        if voteV=challengeVote then
                out(ch, attack)
        else
                out(ch, voteV)
```

Process 6. modified collector process for checking the eligibility properties

outputs the restricted name attack if the test succeeds. The modified collector process is given in Process 6. Verifying eligibility is now reduced to secrecy of the name attack. ProVerif succeeds in proving that attack cannot be deduced by the attacker.

If we register the attacker as a legitimate voter, the tool finds the trivial attack, where the intruder votes *challenge vote*. Similarly, if a corrupt administrator is modeled then the intruder can generate a signed commitment to the challenge vote and insert it.

The second part of the eligibility property (that a voter can vote only once) cannot be verified in our model, because of our simplifying assumption that all voters share the same key.

5.3 Privacy

The privacy property aims to guarantee that the link between a given voter V and his vote v remains hidden. Anonymity and privacy properties have been successfully studied using equivalences. However, the definition of privacy in the context of voting protocols is rather subtle. While generally most security properties should hold against an arbitrary number of dishonest participants, arbitrary coalitions do not make sense here. Consider for instance the case where all but one voter are dishonest: as the results of the vote are published at the end, the dishonest voter can collude and determine the vote of the honest voter. A classical trick for modeling anonymity is to ask whether two processes, one in which V_1 votes and one in which V_2 votes, are equivalent. However, such an equivalence does not hold here as the voters' identities are revealed (and they need to be revealed at least to the administrator to verify eligibility). In a similar way, an equivalence of two processes where only the vote is changed does not hold, because the votes are published at the end of the protocol. To ensure privacy we need to hide the *link* between the voter and the vote and not the voter or the vote itself.

In order to give a reasonable definition of privacy, we need to suppose that at least two voters are honest. We denote the voters V_1 and V_2 and their votes $vote_1$, respectively $vote_2$. We say that a voting protocol respects privacy whenever a process where V_1 votes $vote_1$ and V_2 votes $vote_2$ is observationally equivalent to a process where V_1 votes $vote_2$ and V_2 votes $vote_1$.

```
process
    let x=choice[v1,v2] in
    let y=choice[v2,v1] in
    ( (out(ch,x)) | (out(ch,y)) )
```

Process 7. limitation of the ProVerif tool to prove observational equivalence

With respect to the modeling given in section 4 we explicitly add a second voter. However, the equivalence that is checked by ProVerif is strictly finer than observational equivalence. Therefore the tool does not succeed in proving the above given privacy property. In Process 7, we illustrate a simple process that is observationally equivalent (it is actually structurally equivalent), but cannot be proven so by ProVerif. This example also illustrates ProVerif's choice operator used to define two processes that should be proven observationally equivalent. The choice operator is a binary operator that defines two processes P_1 and P_2 such that choice(x_1,x_2) evaluates to x_1 in P_1 and to x_2 in P_2. Although the two processes are structurally equivalent, the current version of ProVerif does not succeed in proving observational equivalence.

As ProVerif takes as input processes in the applied pi calculus, we can rely on hand proof techniques to show privacy. The processes modeling the two voters are shown in Process 8. The main process is adapted accordingly to publish public keys and host names.

Proposition 1. *The FOO 92 protocol respects privacy, i.e.* $P[vote_1/v_1, vote_2/v_2] \approx P[vote_2/v_1, vote_1/v_2]$*, where* P *is given in Process 9.*

The proof can be sketched as follows. First note that the only difference between $P[vote_1/v_1, vote_2/v_2]$ and $P[vote_2/v_1, vote_1/v_2]$ lies in the two voter processes. We therefore first show that

$$\begin{gathered}(processV1|processV2)[vote_1/v_1, vote_2/v_2] \\ \approx \\ (processV1|processV2)[vote_2/v_1, vote_1/v_2].\end{gathered}$$

To prove this we show labelled bisimilarity. We denote the left-hand process as P_1 and the right-hand process as P_2. The labelled transition of P_1

$$\begin{aligned}P_1 \overset{\nu x1.\bar{ch}\langle x1\rangle}{\longrightarrow}\ & \nu blinder1.\nu r1.\nu blinder2.\nu r2. \\ & (P_1'|\{^{(hostv1,sign(blind(commit(v1,r1),blinder1),skv1)}/_{x1}\}) \\ \overset{\nu x2.\bar{ch}\langle x2\rangle}{\longrightarrow}\ & \nu blinder1.\nu r1.\nu blinder2.\nu r2. \\ & (P_1''|\{^{(hostv1,sign(blind(commit(v1,r1),blinder1),skv1)}/_{x1}\} \\ & |\{^{(hostv2,sign(blind(commit(v2,r2),blinder1),skv2)}/_{x2}\})\end{aligned}$$

```
(* Voter1 *)
let processV1 =
    ν blinder1. ν r1.
    let blindedcommitedvote1=blind(commit(v1,r1),blinder1) in
    out(ch,(hostv1,sign(blindedcommitedvote1,skv1))).
    in(ch,m21).
    let blindedcommitedvote01=checksign(m21,pka) in
    if blindedcommitedvote01=blindedcommitedvote1 then
    let signedcommitedvote1=unblind(m21,blinder1) in
    phase 1.
    out(ch,signedcommitedvote1).
    in(ch,(l1,=signedcommitedvote1)).
    phase 2.
    out(ch,(l1,r1))

(* Voter2 *)
let processV2 =
    ν blinder2. ν r2.
    let blindedcommitedvote2=blind(commit(v2,r2),blinder2) in
    out(ch,(hostv2,sign(blindedcommitedvote2,skv2))).
    in(ch,m22).
    let blindedcommitedvote02=checksign(m22,pka) in
    if blindedcommitedvote02=blindedcommitedvote2 then
    let signedcommitedvote2=unblind(m22,blinder2) in
    phase 1.
    out(ch,signedcommitedvote2).
    in(ch,(l2,=signedcommitedvote2)).
    phase 2.
    out(ch,(l2,r2))
```

Process 8. two voters for checking the privacy property

can be simulated by P_2 as

$$
\begin{aligned}
P_2 \xrightarrow{\nu x1.\bar{ch}\langle x1\rangle} & \nu blinder1.\nu r1.\nu blinder2.\nu r2. \\
& (P_2' | \{^{(hostv1, sign(blind(commit(v2,r1),blinder1),skv1)}/_{x1}\}) \\
\xrightarrow{\nu x2.\bar{ch}\langle x2\rangle} & \nu blinder1.\nu r1.\nu blinder2.\nu r2. \\
& (P_2'' | \{^{(hostv1, sign(blind(commit(v2,r1),blinder1),skv1)}/_{x1}\} \\
& | \{^{(hostv2, sign(blind(commit(v1,r2),blinder1),skv2)}/_{x2}\})
\end{aligned}
$$

For the first input of both voters, we need to consider two cases: either the input of both voters corresponds to the expected messages from the administrator or any other message has been introduced by the attacker. In the first case, both voters synchronize

```
process
    ν ska. ν skv1. ν skv2. (* private keys *)
    ν privCh. (* channel for registrating legimitate voters *)
    let pka=pk(ska) in
    let hosta = host(pka) in
    let pkv1=pk(skv1) in
    let hostv1=host(pkv1) in
    let pkv2=pk(skv2) in
    let hostv2=host(pkv2) in
    (* publish host names and public keys *)
    out(ch,pka). out(ch,hosta).
    out(ch,pkv1). out(ch,hostv1).
    out(ch,pkv2). out(ch,hostv2).
    let v1=choice[vote1,vote2] in
    let v2=choice[vote2,vote1] in
    ((out(privCh,pkv1). out(privCh,pkv2). out(privCh,pk(ski))) |
    (processV1)|(processV2)|(!processA)|(!processC))
```

Process 9. main process with two voters

at phase 1 and the frames of P_1, respectively P_2 are

$$\phi_1 = \nu blinder1.\nu r1.\nu blinder2.\nu r2.\{^{(hostv1,sign(blind(commit(v1,r1),blinder1),v1))}/_{x1},\ ^{(hostv2,sign(blind(commit(v2,r2),blinder2),v2))}/_{x2},\ ^{sign(blind(commit(v1,r1),blinder1),skva)}/_{x3},\ ^{sign(blind(commit(v2,r2),blinder2),skva)}/_{x4}\}$$

$$\phi_2 = \nu blinder1.\nu r1.\nu blinder2.\nu r2.\{^{(hostv1,sign(blind(commit(v2,r1),blinder1),v1))}/_{x1},\ ^{(hostv2,sign(blind(commit(v1,r2),blinder2),v2))}/_{x2},\ ^{sign(blind(commit(v2,r1),blinder1),skva)}/_{x3},\ ^{sign(blind(commit(v1,r2),blinder2),skva)}/_{x4}\}$$

Given our equational theory and the fact that the blinding factors are restricted, these frames are statically equivalent. In the second case, if at least one input does not correspond to the correct administrator's signature, both voter processes will block, as testing correctness of the message fails and hence they cannot synchronize.

After the synchronization at phase 1, the remaining of the voter processes are structurally equivalent: the remaining of the first voter's process of P_1 is equivalent to the remaining of the second voter's process of P_2 and vice-versa. Due to this structural equivalence, P_2 can always simulate P_1 (and vice-versa). Moreover static equivalence will be ensured: with respect to frames ϕ_1 and ϕ_2 no other difference will be introduced and the blinding factors are never divulged.

Given observational equivalence of the voter processes, we can conclude observational equivalence of the the whole process, as observational equivalence is closed under application of closed evaluation contexts.

Note also that the use of phases is crucial for privacy to be respected. Surprisingly, when we omit the synchronization after the registration phase with the administrator, privacy is violated. Consider the following scenario. Voter 1 contacts the administrator. As no synchronization is considered, voter 1 can send his commited vote to the collector before voter 2 contacts the administrator. As voter 2 could not have submitted the commited vote, the attacker can link this commitment to the first voter's identity. This problem was found during a first attempt to prove the protocol where the phase instructions were omitted. The original paper divides the protocol into three phases but does not explain the crucial importance of the synchronization after the first phase. Our analysis emphasizes this need and we believe that it increases the understanding of some subtle details of the privacy property in this protocol.

6 Conclusion

We have modelled the FOO 92 electronic voting scheme in the applied pi calculus, and proved three kinds of property. Each property is checked either by reachability analysis or by checking observational equivalence:

Fairness. F1: the vote of a particular voter is not leaked to an attacker (reachability). F2: a guess of a vote cannot be verified by the attacker and strong secrecy is guaranteed (observational equivalence). These properties are also proved in the presence of a corrupt administrator.

Eligibility. E1: an attacker cannot trick the system into accepting his vote (reachability).

Privacy. P1: the attacker cannot distinguish the actual situation from one in which two voters have swapped their votes (observational equivalence).

The reachability properties (F1, E1) and the first observational equivalence property (F2) can be proved by ProVerif. The other observational equivalence property (P1) is more delecate, both in the way it is formulated and in the way that it is proved. ProVerif cannot prove this observational equivalence automatically. Therefore we proved it manually, by showing that the two processes are labelled-bisimilar.

In proving P1 manually, we noticed a feature of the protocol which is not much stressed in the descriptions (e.g. [12, 16]) but is vital for the proof: every participant must finish the registration stage before proceeding to the voting stage, and every participant must finish the voting stage before the collector can begin opening the votes. Otherwise, some attacks are possible. For example, if voting could begin before everyone has registered, the attacker could break privacy by temporarily blocking all registrations but V's. If V then votes, the attacker can establish a link between V and V's vote. We used the phase construct of ProVerif to prevent this.

Acknowledgments. Many thanks to Bruno Blanchet for suggestions about using ProVerif, as well as to Mathieu Baudet and Stéphanie Delaune for interesting discussions and comments. Thanks also to anonymous reviewers for helpful comments. Steve Kremer's work was partially supported by the RNTL project PROUVÉ and the ACI-SI Rossignol.

References

1. Martín Abadi, Bruno Blanchet, and Cedric Fournet. Just fast keying in the pi calculus. In David Schmidt, editor, *13th European Symposium on Programming (ESOP'04)*, volume 2986 of *Lecture Notes in Computer Science*, pages 340–354, Barcelona, Spain, March 2004. Springer.
2. Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. In Josep Diaz, Juhani Karhumäki, Arto Lepistö, and Don Sannella, editors, *31st Int. Coll. Automata, Languages, and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 46–58, Turku, Finland, July 2004. Springer.
3. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In Hanne Riis Nielson, editor, *Proceedings of the 28th ACM Symposium on Principles of Programming Languages*, pages 104–115, London, UK, January 2001. ACM.
4. Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In Steve Schneider, editor, *14th IEEE Computer Security Foundations Workshop*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society Press.
5. Bruno Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, Oakland, California, May 2004.
6. Rohit Chadha, Steve Kremer, and Andre Scedrov. Formal analysis of multi-party contract signing. In Riccardo Focardi, editor, *17th IEEE Computer Security Foundations Workshop*, pages 266–279, Asilomar, CA, USA, June 2004. IEEE Computer Society Press.
7. David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology, Proceedings of CRYPTO'82*, pages 199–203. Plenum Press, 1983.
8. Ricardo Corin, Jeroen Doumen, and Sandro Etalle. Analysing password protocol security against off-line dictionary attacks. In *2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP'04)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2004. To appear.
9. Véronique Cortier. Personal communication, 2004.
10. Stéphanie Delaune and Florent Jacquemard. A theory of dictionary attacks and its complexity. In Riccardo Focardi, editor, *17th IEEE Computer Security Foundations Workshop*, pages 2–15, Asilomar, Pacific Grove, CA, USA, June 2004. IEEE Computer Society Press.
11. Cedric Fournet and Martin Abadi. Hiding names: Private authentication in the applied pi calculus. In *International Symposium on Software Security (ISSS'02)*, pages 317–338. Springer, 2003.
12. Atsushi Fujioka, Tatsuaki Okamoto, and Kazui 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 *Lecture Notes in Computer Science*, pages 244–251. Springer, 1992.
13. Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2004.
14. Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.
15. Gavin Lowe. Analysing protocols subject to guessing attacks. In Joshua Guttman, editor, *In Proceedings of the Workshop on Issues in the Theory of Security (WITS'02)*, Portland, Oregon, USA, January 2002.
16. Zuzana Rjaskova. Electronic voting schemes. Master's thesis, Comenius University, 2002. www.tcs.hut.fi/ helger/crypto/link/protocols/voting.html.

B Étude calculatoire du schéma de vote de Chaum

Probabilistic Opacity for a Passive Adversary and its Application to Chaum's Voting Scheme

Yassine Lakhnech and Laurent Mazaré

VERIMAG - 2, av. de Vignates, 38610 Gières - FRANCE
yassine.lakhnech@imag.fr,laurent.mazare@imag.fr

Abstract. A predicate is opaque for a given system, if an adversary will never be able to establish truth or falsehood of the predicate for any observed computation. This notion has been essentially introduced and studied in the context of transition systems whether describing the semantics of programs, security protocols or other systems. In this paper, we are interested in studying opacity in the probabilistic computational world. Indeed, in other settings, as in the Dolev-Yao model for instance, even if an adversary is 99% sure of the truth of the predicate, it remains opaque as the adversary cannot conclude for sure. In this paper, we introduce a computational version of opacity in the case of passive adversaries called cryptographic opacity. Our main result is a composition theorem: if a system is secure in an abstract formalism and the cryptographic primitives used to implement it are secure, then this system is secure in a computational formalism. Security of the abstract system is the usual opacity and security of the cryptographic primitives is IND-CPA security. To illustrate our result, we give two applications: a short and elegant proof of the classical Abadi-Rogaway result and the first computational proof of Chaum's visual electronic voting scheme.
Keywords: Opacity, Non-Interference, Chaum's Voting Scheme, Computational Model, Probabilistic Encryption.

Introduction

Roughly speaking, a predicate is opaque for a given system, if an adversary will never be able to establish truth or falsehood of the predicate, for any observed execution of the system. It is clear that this notion only makes sense when the adversary does not have access to the complete state of the system but rather accesses its execution through an observation function. Typically, the predicate of interest concerns some non-determinism that is resolved initially such as the votes in a voting scheme. This notion has been essentially introduced and studied in the context of transition systems whether describing the semantics of programs, security protocols or other systems. It generalizes well-known security properties such as anonymity and non-interference (See [7] for a discussion).

Opacity has been essentially studied in the so-called formal [1] world of security, where cryptography is assumed perfect. A typical formal model, for security protocols for instance, is the Dolev and Yao model [11] where messages are described by algebraic terms and there is one single adversary that subsumes all possible attacks. In this paper, we are interested in studying opacity in the probabilistic computational world. Indeed, in the formal world, even if an adversary is 99% sure of the truth of the predicate, it remains opaque as the adversary cannot conclude for sure. Such a definition is clearly not useful in the probabilistic computational setting. Therefor, we introduce a computational version of opacity. We restrict ourselves to the case of passive adversaries and call our security notion *cryptographic opacity*. More generally, we introduce probabilistic opacity that includes : strict opacity, plausible deniability and cryptographic opacity. All three notions are defined by experiments and in terms of the advantage of the adversaries. Strict opacity requires that the advantage is null; plausible deniability requires that the probability to win the experiment is different from 1 and cryptographic opacity requires that the advantage is negligible.

[1] One might prefer the word symbolic here since formal is not used in the sense of rigorous.

Then, the question of how to prove probabilistic cryptographic opacity rises. For strict opacity, we show a decidability result for finite systems. The main core of the paper, however, deals with cryptographic opacity. Our answer to this question is inspired by a recent trend started by [2] and pushed further in [4, 18, 15, 10] in bridging the gap that separates the Dolev-Yao model and its perfect cryptography assumption on one hand and the computational model on the other hand. Indeed, we prove our main result that states the following: if a predicate is opaque in the formal model for an abstraction of the considered system and if the cryptographic primitives are IND-CPA, then cryptographic opacity of the predicate holds in the computation model. The previously mentioned results on the relationship between the formal and the computational models do not immediately apply in our case as we have to carefully deal with random coins used for encryption. Indeed, in the case of Chaum's voting scheme, for instance, some of these coins appear as plain-text. On the other hand, for some IND-CPA schemes, e.g. [6], knowledge of the random coins induces knowledge about the encrypted message. To deal with this problem, we introduce a new security criterion, called n-RPAT-CPA. We then show that any IND-CPA secure cryptographic scheme is also n-RPAT-CPA secure.

An other important contribution of our paper is the proof of opacity for Chaum's visual voting scheme [8]. This is done by applying our main result. We also give a sort proof of Abadi and Rogaway's result as an application of our main theorem.

Related work. The initial work of Abadi and Rogaway was pushed further in [1]. This last paper considers systems with cryptographic primitives and studies indistinguishability, but this property lacks the generality of opacity. Another interesting work is [3] which links a computational version of probabilistic non-interference [14] to the notion of simulatability. This work is very general as it considers active adversaries but as a consequence, their main theorem is more difficult to apply. In [17], a computational definition of indistinguishability (or strong secrecy) is given. This security notion is less general than opacity. Laud also formulates an analysis allowing verification of programs using cryptographic primitives but these primitives are still abstracted.

This paper is structured as follows. The first section recalls some necessary preliminaries. The second section introduces strict opacity, plausible deniability and cryptographic opacity . The following section proves a decidability result for strict opacity. Section 4 presents an approach to verification of cryptographic opacity. This approach is applied to Chaum's voting scheme in Section 5. The proof of main result and the security criterion n-RPAT-CPA are discussed in Section 6.

1 Preliminaries

In this section, we recall some basic definitions that are useful when considering probabilistic systems and introduce a general definition of security criteria along with a decomposition theorem that is used later in the paper. All these notions are detailed in [16].

1.1 Cryptographic Primitives

Let η be the security parameter of the system, it characterizes the strength of the cryptographic primitives as well as the length of nonces.

An *asymmetric encryption scheme* $\mathcal{AE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ is defined by three algorithms. The key generation algorithm $\mathcal{KG}$ is a randomized function which given a security parameter η outputs a pair of keys (pk, sk), where pk is a public key and sk the associated secret key. The encryption algorithm $\mathcal{E}$ is also a randomized function which given a message and a public key outputs the encryption of the message by the public key. The random part is explicitly represented as a nonce (bit-string of length η) bs which is given as argument to $\mathcal{E}$. Finally the decryption algorithm $\mathcal{D}$ takes as input a secret key and a cypher-text and outputs the corresponding plain-text, i.e., $\mathcal{D}(\mathcal{E}(m, pk, bs), sk) = m$ for any bs and any pair (pk, sk) produced by the key generation algorithm. The execution time of the three algorithms is assumed polynomially bounded by η.

A function $g : \mathbb{R} \to \mathbb{R}$ is *negligible*, if it is ultimately bounded by x^{-c}, for each positive $c \in \mathbb{N}$, i.e., for all $c > 0$ there exists N_c such that $|g(x)| < x^{-c}$, for all $x > N_c$.

1.2 Security Criteria

Security criteria define the correctness (or the expected properties) of an encryption scheme. They are defined as an experiment involving an adversary. Given access to a set of oracles, the adversary has to guess a randomly chosen data. [2] Roughly speaking, a scheme is safe w.r.t. a given criterion, if no adversary has a better probability to win than an adversary who does not have access to the oracles. Therefor, the strength of a criterion depends on the allowed adversaries and the offered oracles. In this paper, we consider adversaries that are terminating random Turing machines (RTM) or polynomial-time random TM (PRTM) when considering computational encryption. The time is bounded in the security parameter η. An RTM $\mathcal{B}$ is said to have a complexity *similar* to the complexity of $\mathcal{A}$ if the execution of $\mathcal{B}$ is polynomially bounded in the (maximum) execution duration of $\mathcal{A}$.

Let us now define formally *security criteria*. A criterion γ is a triple $(\Theta; F; V)$ where

- Θ is a (P)RTM that randomly generates some challenge θ (for example, a bit b and a pair of key (pk, sk)).
- F is a (P)RTM that takes as arguments a string of bits s and a challenge θ and outputs a new string of bits. F represents the oracles that an adversary can call to solve its challenge.
- V is a (P)RTM that takes as arguments a string of bits s and a challenge θ and outputs either true or false. It represents the verification made on the result computed by the adversary. The answer true (resp. false) means that the adversary solved (resp. did not solve) the challenge.

Note that Θ can generate an arbitrary number of parameters and F can represent an arbitrary number of oracles. Thus, it is possible to define criteria with multiples Θ and F. When no confusion may rise, we use the same notation for the challenge generator Θ and the generated challenge θ (both are denoted using θ).

A criterion $(\Theta; F; V)$ and adversary $\mathcal{A}$ produce the following experiment. First θ is generated randomly. The adversary can now make some computation using the oracle F, this is denoted by $\mathcal{A}/F$. The behavior of the oracle depends on θ. At the end of computation, the adversary has to return a string of bits which is verified by an algorithm V. Also V uses θ (e.g. θ includes a bit b and the adversary has to output the value of b). The aim of the adversary $\mathcal{A}$ is produce a bit string that is verified by V. More formally, the experiment $\mathbf{Exp}^{\gamma}_{\mathcal{A}}(\eta)$ involving $\mathcal{A}$ and γ is defined by the following Turing machine:

Experiment $\mathbf{Exp}^{\gamma}_{\mathcal{A}}(\eta)$:
  $\theta \leftarrow \Theta(\eta)$
  $d \leftarrow \mathcal{A}/\eta, \lambda s.F(s,\theta)$
  return $V(d,\theta)$

Experiment $\mathbf{Exp'}^{\gamma}_{\mathcal{A}}(\eta)$:
  $\theta \leftarrow \Theta(\eta)$
  $d \leftarrow \mathcal{A}/\eta$
  return $V(d,\theta)$

We can now define the advantage of $\mathcal{A}$ against γ as follows:

$$\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta) = 2.\big(pr(\mathbf{Exp}^{\gamma}_{\mathcal{A}}(\eta) = true) - PrRand^{\gamma}\big),$$

where $PrRand^{\gamma}$ is the best probability to solve the challenge that an adversary can have without using oracle F. Formally, $PrRand^{\gamma}$ is the maximum of $pr(\mathbf{Exp'}^{\gamma}_{\mathcal{A}}(\eta) = true)$ where $\mathcal{A}$ ranges over any possible adversaries and $\mathbf{Exp'}$ is similar to $\mathbf{Exp}$ except that F cannot be used by A.

1.3 Decomposition of Security Criteria

In this section, we recall the reduction theorem given in [16]. Let $\gamma = (\theta_1, \theta_2; F_1, F_2; V_2)$ be a criterion. Let γ_1 and γ_2 be two criteria such that:

- There exist two PRTM G and H such that:

$$G(H(s, \theta_2, \theta_2'), 1, \theta_1) = F_1(s, \theta_1, \theta_2)$$
$$G(H(s, \theta_2, \theta_2'), 0, \theta_1) = F_1(s, \theta_1, \theta_2')$$

[2] In some cases, as for symmetric encryption or signature, the adversary has other ways to win. This is not relevant for this paper.

Oracle G operates on a string of bits, thus it must receive two challenge informations, a bit b and θ_1.
- $\gamma_2 = (\theta_2; F_2; V_2)$ and $\gamma_1 = (b, \theta_1; G; verif_b)$ where b generates a random bit and $verif_b$ is the PRTM verifying that the output of the adversary is b: $verif_b(s, b, \theta_1) = (s \Leftrightarrow b)$.
- $F_2(s, \theta_1, \theta_2)$ and $V_2(s, \theta_1, \theta_2)$ do not depend on θ_1.

Then we say that (γ_1, γ_2) is a *valid simplified partition* of γ.

Theorem 1. *Let (γ_1, γ_2) be a valid simplified partition of γ. For any RTM $\mathcal{A}$, there exist two RTM $\mathcal{A}^o$ and $\mathcal{B}$ of similar complexity such that*

$$|\mathbf{Adv}^{\gamma}_{\mathcal{A}}(\eta)| \leq 2.|\mathbf{Adv}^{\gamma_1}_{\mathcal{B}}(\eta)| + |\mathbf{Adv}^{\gamma_2}_{\mathcal{A}^o}(\eta)|$$

2 Probabilistic Opacity

2.1 Systems and Observations

First, we give a general definition for random systems then we explain how their behaviors can be observed by an eavesdropper. As the results of this section do not depend on any particular model of systems, we simply consider randomized functions.

Let Σ be a finite alphabet representing actions made by a system. A trace is a finite sequence of actions, i.e., a word over Σ. Let Σ^* be the set of words over Σ and ϵ be the empty word.

A *system* is a random function Δ from a finite set S of initial states to sequences of actions. Random means that the system can perform some non-deterministic operations. For example it can pick up a random bit b. If $b = 0$, it performs action a else action b. The set of possible traces of a system Δ is denoted by $\Delta(S)$. In a similar way, $\Delta(\{s\})$ is the set of possible traces starting from s in S. This set can have more than one element as function Δ is random.

An *observation function* allows the eavesdropper to see only limited information about traces produced by the studied system. These functions are mappings from Σ to $\Sigma \cup \{\epsilon\}$. Hence, it is possible for an action to be totally invisible from the outside if the observation function replaces it with ϵ.

2.2 Opacity

Let us consider a system Δ with possible initial states S and an observation function *obs*. A property ψ is a predicate over S. A property ψ is *opaque* if given $s \in S$ and $t \in \Delta(\{s\})$, it is not possible, for an adversary that has access uniquely access to $obs(t)$ to know whether s verifies ψ. Here, not possible means that it should not be possible to achieve this with "reasonable" probability. This notion of opacity is introduced under the name of *initial opacity* in [7].

More than in opacity itself, we are here interested in the advantage that an adversary can get by having access to the observation of the trace. If a very vast majority of initial states in S verify ψ, the adversary can suppose that s verifies ψ even without looking at the trace. However, by looking at the trace, it is possible to get some new information and to deduce the result for sure. To define this advantage, we consider that the adversary $\mathcal{A}$ tries to win the following game/experiment:

1. An initial state s is chosen randomly in S;
2. The adversary $\mathcal{A}$ is given the observation of a trace in $\Delta(\{s\})$ and has to output a bit b;
3. $\mathcal{A}$ wins its challenge, when b is equivalent to the property "s satisfies ψ".

This game is represented by an experiment which is a random Turing machine. The experiment related to adversary $\mathcal{A}$ and to *obs* is the following RTM.

Experiment $\mathbf{Exp}^{obs}_{\mathcal{A}}$:
$s \leftarrow S$
$t \leftarrow \Delta(s)$
$b \leftarrow \mathcal{A}(obs(t))$
return $b \Leftrightarrow (s \in \psi)$

The advantage is the difference between the probability that $\mathcal{A}$ solves its challenge and the best probability that one can get without access to the observation. Hence, it is defined by the following formula.

$$\mathbf{Adv}^{obs}_{\mathcal{A}} = 2.\big(pr(\mathbf{Exp}^{obs}_{\mathcal{A}} \rightarrow true) - PrRand^{\psi}\big)$$

Where $PrRand^{\psi}$ is the greatest possible value for $pr(\mathbf{Exp}^{\epsilon}_{\mathcal{B}} \rightarrow true)$ for any $\mathcal{B}$ and ϵ represents the observation function that associates ϵ to any action in Σ.

Note that the above definitions for the experiment and the advantage can easily be defined in an equivalent way by using the general notion of security criterion.

- Θ randomly generates an initial parameter s and a trace t;
- F gives access to the trace observation $obs(t)$;
- V verifies that the output bit b correctly answers the question: does s verify ψ ?

Criterion $(\Theta; F; V)$ has exactly the same related experiment and advantage as those given above.

Using the definition of advantage, it is possible to tell if an observation function has any use in trying to solve the challenge.

Definition 1. *Let Δ be a system, S be the set of its initial states and ψ a property over S. An observation function obs is called*

- safe for **strict** opacity *of ψ , if for any RTM $\mathcal{A}$,* $\mathbf{Adv}^{obs}_{\mathcal{A}} = 0$
- safe for **cryptographic** opacity *of ψ, if for any PRTM $\mathcal{A}$,* $\mathbf{Adv}^{obs}_{\mathcal{A}}$ *is negligible*
- safe for **plausible deniability** *of ψ, if for any RTM $\mathcal{A}$,* $\mathbf{Adv}^{obs}_{\mathcal{A}} \neq 2 - 2.PrRand^{\psi}$ *i.e.* $pr(\mathbf{Exp}^{obs}_{\mathcal{A}} \rightarrow true)$ *is different from* 1.

Plausible deniability coincides with the opacity notion introduced in [7], which is itself closely related to anonymity [19] and non-interference [12, 20]. This link and some useful basic properties are detailed in appendix A.1.

For strict opacity, if an observation function is safe then an adversary gets no advantage at all by looking at the observation. This is for example the informations exchanged by cryptographers during the cryptographs diner (if we consider that one of them paid the diner for any element of S).

For cryptographic opacity, an observation function may return some relevant information that cannot be exploited in a reasonable (i.e. polynomial) time. For example, if we consider that all the actions made by a system are encrypted using a safe encryption scheme, then the observation is safe. In this context, a safe encryption scheme is an IND-CPA encryption scheme, this is detailed further in this document.

The idea is that with plausible deniability, if the adversary observes some trace t, then it cannot conclude for sure whether property ψ is verified or not. There exists at least one initial state satisfying ψ and one not satisfying ψ that both produce the observation $obs(t)$.

There is no clear hierarchy among strict opacity and plausible deniability as the first notion does not imply the second one (this implication is only true when $PrRand$ is different from one).

3 Decidability of Strict Opacity for Finite Systems

Let us consider the case where only a finite number of traces can occur. Thus, we suppose that both S and $\Delta(S)$ are finite. With this assumption, the greatest advantage for any adversary can be computed. Moreover, there exists an adversary that reaches this advantage.

Let O be the set of all possible observations, i.e. $O = obs(\Delta(S))$. We first define the interest of an observation function *obs*. This definition is rather intuitive as an observation function can bring some advantage if the probability for ψ to be true knowing the observation is different from the general probability of ψ. This explains why this definition uses the term $|pr(\psi) - pr(\psi|o)|$.

Definition 2. *The interest I_{obs} of an observation function obs is given by:*

$$I_{obs} = 2. \sum_{o \in O} pr(o).\big|pr(\psi) - pr(\psi|o)\big|$$

Then, the main result of this section is that the interest is the greatest possible advantage. For that reason, as it is possible to effectively compute the interest of a given observation, safety for strictly opacity of an observation function is a decidable problem. The following proposition states the main result. Its proof is given in Appendix A.

Proposition 1. *For any adversary $\mathcal{A}$ and observation function obs, $|\mathbf{Adv}^{obs}_{\mathcal{A}}| \leq I_{obs}$. Moreover, there exists an RTM $\mathcal{A}_{obs}$ whose advantage is exactly I_{obs}.*

It is important to notice that the second statement of the previous proposition asserts existence of an RTM $\mathcal{A}_{obs}$ with $\mathbf{Adv}^{obs}_{\mathcal{A}} = I_{obs}$. This RTM is not necessarily a legal adversary. Indeed, $\mathcal{A}_{obs}$ has an execution time which is linear in the number of possible observations. This is not a problem when considering strict opacity or plausible denying as adversaries are RTM. However, for cryptographic opacity, we only admit adversaries in PRTM. Worse, even if we assume the quite fair hypothesis (for an eavesdropper) that there is only a bounded number of messages which all have some bounded size, the number of possible observations may be exponential in the security parameter η, and hence, $\mathcal{A}_{obs}$ may not a PRTM.

Nevertheless, this result is interesting at least for strict opacity as shown now. Indeed, a consequence of this proposition is that no adversary has an advantage if and only if for any o in O, $pr(\psi) = pr(\psi|o)$. When one wants to verify strict opacity, it is possible to test that for any observation, the probability for ψ to be true assuming that observation is exactly the general probability for ψ to be true. Hence, we have

Proposition 2. *Let obs be an observation function and ψ a property. Then, obs is safe for strict opacity of ψ if and only if for any $o \in obs(\Delta(S))$, $pr(s \in \psi) = pr(s \in \psi | obs(s) = o)$*

4 An Approach to the Verification of Cryptographic Opacity

As we noted in the previous section, we make the finite behavior hypothesis for cryptographic systems (with passive adversaries). That is, we assume that S and $\Delta(S)$ are finite. The approach proposed for proving strict opacity using the interest of the observation function and the existence of an adversary matching this interest is not applicable for cryptographic opacity. Indeed, when considering cryptographic opacity, adversaries are restricted PRTM. Therefor, we present here a different approach. The main clue in this approach is to decompose the verification of cryptographic opacity into the verification of strict opacity for an abstracted system on one hand and the safety of the underlying encryption scheme on the other hand.

It is useful to notice that this approach is similar to the approach followed for proving secrecy properties of cryptographic protocols, where one proves an abstraction of the secrecy property while making the perfect cryptography hypothesis and relies on the fact that this verification is valid in the computational model, if the cryptographic primitives satisfy some well-defined properties. The formal justification of this approach is the result of recent research aiming at relating the formal and the computational models for security protocols [2, 4, 10]. These papers show that the Dolev-Yao [11] model is a safe abstraction of the computational model (where adversaries are poly-time Turing machines) as soon as the cryptographic primitives (e.g. the encryption scheme) verify some computational properties.

4.1 Specifications and Patterns

In the cryptographic setting, the alphabet Σ consists of the symbols, 0 and 1. Thus, an action of the alphabet is a bit-string. We consider systems that produce some finite size bit-string (usually, their size is polynomial in the security parameter η).

To define the abstract systems, we introduce patterns which are simply elements of the free algebra of terms almost as in the Dolev-Yao model. It is *almost* because in our setting and as we are interested in opacity, we have to be careful in handling the random coins[3] used in encryption. Let us

[3] Random coins are also nonces but some times we use rather random coins to insist on the fact they are used to randomize encryptions

explain. In the simplest Dolev-Yao model (also called the formal or symbolic model) an encryption of a message m with key pk is represented by the term $\{m\}_{pk}$. Thus, two message $\{m_1\}_{pk_1}$ and $\{m_2\}_{pk_2}$ are equal iff $m_1 = m_2$ and $pk_1 = pk_2$. Moreover, an adversary who does not know the inverse key of pk cannot get any information from $\{m\}_{pk}$. This means that the random nonce is completely abstracted away. In some refinements of this model, however, labels are introduced to distinguished encryptions made at different instants during a protocol execution [10]. Such labels are only an approximation of random coins as the latter may be equal even when two encryptions are performed at different instants. As we want to verify the Chaum voting scheme, we have to include explicitly random coins in our patterns. Therefor, we write $\{m; N\}_{pk}$ to represent the result of encrypting m with key pk using nonce N as random coins for the encryption algorithm.

Let $\mathcal{K}$ be an infinite set of keys (as explained above rather key names); k^{-1} represents the private key corresponding to a public key k. Moreover, let $\mathcal{N}$ be a set of nonces. Patterns are defined by the following grammar where k is a key, bs a bit-string and N is a nonce:

$$pat ::= bs|N|\langle pat, pat\rangle|\{pat; N\}_k|k \quad k \text{ may be a public or private key}$$

Without loss of generality, we consider abstract systems that only produce one pattern and not a list of patterns as it is possible to concatenate patterns using pairing. Thus, a *specification* Δ_s is a function from S to pat.

Obviously, given a pattern pat the information that can be extracted from pat depends on the set of private keys that can be computed from pat. The set of patterns that can be learned/computed from a pattern is defined as follows:

Definition 3. *Let p be a pattern, the set $dec(p)$ is inductively defined by the following inferences.*

- *p is in $dec(p)$.*
- *If $\langle p_1, p_2\rangle$ is in $dec(p)$, then p_1 and p_2 are in $dec(p)$.*
- *If $\{p_1; N\}_k$ and k^{-1} are in $dec(p)$, then p_1 and N are in $dec(p)$.*
- *If $\{p_1; N\}_k$ and N are in $dec(p)$, then p_1 is in $dec(p)$.*

Notice that since, we only consider atomic keys, we only have to consider decompositions. It is also useful to notice that the last clause is usually not considered in the Dolev-Yao model. This clause is motivated by the existence of IND-CPA algorithms such that the knowledge of the random used for encryption allows to decrypt the message. An example of such algorithm is presented in [6].

A pattern has also a denotation in the cryptographic setting. This depends on a context θ that associates keys and nonces to their corresponding bit-string values. Thus, the cryptographic (or computational) value of a term $\{pat; N\}_k$ is $\mathcal{E}(m, bs, bs')$, where m is the value of pat, bs the value of pk and bs' the value N. Let θ be a mapping associating bit-strings to nonces and keys. The value of a pattern in the context θ is defined recursively:

$$v(bs, \theta) = bs \qquad v(\langle p_1, p_2\rangle, \theta) = v(p_1, \theta).v(p_2, \theta)$$
$$v(N, \theta) = \theta(N) \qquad v(\{p; N\}_k, \theta) = \mathcal{E}(v(p, \theta), \theta(k), \theta(N)) \qquad v(k, \theta) = \theta(k)$$

Let us briefly summarize what we have introduced. We defined the systems we want to consider whose behavior in each initial state s is a set of patterns and we have associated to each pattern its value, a bit-string, in a given context.

We now turn our attention to the observations we can make about a pattern. We define two observations. The concrete observation of a pattern pat in a context θ is defined as follows: $obs_c(pat, \theta) = v(pat, \theta)$, that is, obs_c corresponds to the observations that can be made in the cryptographic setting. The abstract observation obs_a applied to a pattern replaces every subterms of the form $\{pat; N\}_k$ with $\Diamond^N$, that is, it simply replaces it by a black box. Formally, patterns are transformed in obfuscated patterns which are given by the following grammar:

$$opat ::= bs|N|\langle opat, opat\rangle|\{opat; N\}_k|\Diamond^N|k$$

And observation obs_a of a pattern pat is recursively defined by the following rules.

$$obs_a(bs) = bs \qquad obs_a(\langle p_1, p_2\rangle) = \langle obs_a(p_1), obs_a(p_2)\rangle$$
$$obs_a(N) = N \qquad obs_a(\{p; N\}_k) = \{obs_a(p); N\}_k \text{ if } k^{-1} \in dec(pat) \vee N \in dec(pat)$$
$$obs_a(k) = k \qquad obs_a(\{p; N\}_k) = \Diamond^N \text{ else}$$

As encryption cycles may lead to some vulnerabilities, we restrict ourselves to *well-formed* patterns. For that purpose, we define an ordering on pairs consisting of a key and a nonce. Let pat be a pattern and let $E_<$ be the set of pairs (k, N) such that there is a pattern of the form $\{pat'; N\}_k$ in $dec(pat)$ with k and N not in $dec(pat)$. Then, for $(k, N), (k', N') \in E_<$, $(k, N) < (k', N')$ iff there exist two patterns $\{pat_1; N\}_k$ and $\{pat_2; N'\}_{k'}$ in $dec(pat)$ verifying one of the following conditions:

1. N, k or k^{-1} is a sub-term of $pat_2; N'$;
2. $N = N'$ and $\{pat_1; N\}_k \neq \{pat_2; N\}_{k'}$.

A pattern pat is *well-formed*, if the projection of $<$ on keys is acyclic. Finally, we only consider *well-formed specifications*, i.e. specifications that output well-formed patterns.

The conditions above imply that if pat is well-formed, then for $(k, N) \in E_<$, there is only one encoding using each N (and a non-deducible key) in $dec(pat)$. Hence when obs_a transforms an encoding into $\diamondsuit^N$, this always denotes the exact same encoding (in particular, there is no randomness-reuse as described in [5]). Thus the N label can be seen as a constraint over encodings (specifying possible bit-to-bit equalities). This is why, equality between two *opat* is defined modulo renaming of the nonces. To illustrate this, let us consider two patterns $pat_0 = \langle \{m; N\}_k, \{m; N\}_k \rangle$ and $pat_1 = \langle \{m; N''\}_k, \{m; N'\}_k \rangle$. Then $obs_a(pat_0) = \langle \diamondsuit^N, \diamondsuit^N \rangle$ and $obs_a(pat_1) = \langle \diamondsuit^{N'}, \diamondsuit^{N''} \rangle$. As $obs_a(pat_1)$ and $obs_a(pat_2)$ are different, and hence, pat_0 and pat_1 are distinguishable. If we consider $pat_0 = \{m; N\}_k$ and $pat_1 = \{m; N''\}_k$. Then $obs_a(pat_0) = \diamondsuit^N$ and $obs_a(pat_1) = \diamondsuit^{N'}$. In this case, $obs_a(pat_1)$ and $obs_a(pat_2)$ are equal (modulo renaming), and hence, pat_0 and pat_1 are indistinguishable. And in fact, if the encryption scheme is IND-CPA, pat_0 and pat_1 are indistinguishable even in the computational setting.

Main result The main result of this paper, that we prove in Section 6, is that for each adversary $\mathcal{A}$, there exist two adversaries $\mathcal{A}^o$ and $\mathcal{B}$ such that

$$|\mathbf{Adv}_{\mathcal{A}}^{obs_c \times obs_a}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}| + 2.|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$$

Where n is the number of keys, n-RPAT-CPA is a security criterion verified by any IND-CPA algorithm, observable $obs_c \times obs_a$ gives access to both obs_c and obs_a (see appendix A.1 for details). This means that if the encryption scheme used is IND-CPA, opacity in the formal world implies opacity in the computational world.

4.2 Application: the Classical Abadi-Rogaway Result

Using our main theorem, it is possible to prove a slightly extended version of the seminal result of Abadi and Rogaway [2]. This result states that provided the used encryption scheme is IND-CPA indistinguishability in the formal (Dolev-Yao) model implies indistinguishability in the computational model. In fact, obfuscated patterns are close to patterns as introduced in [2]. The main difference is that our patterns explicitly represent random coins. However, it is still possible to get exactly Abadi and Rogaway's result by assuming fresh distinct nonces for every encryption. If we consider messages with no encryption cycles, then the corresponding patterns (using fresh nonces) are well-formed. Moreover as nonces used for encryption are fresh, each $\diamondsuit^N$ has a different label, thus to test equality these labels are not considered. The Abadi-Rogaway theorem can be stated as an opacity problem. Let m_0 and m_1 be two well-formed patterns. There are two initial states in S: 0 and 1. Specification $\Delta_S(s)$ outputs m_s. Then m_0 and m_1 are *indistinguishable* if for any adversary $\mathcal{A}$, $|\mathbf{Adv}_{\mathcal{A}}^{obs_c}|$ is negligible.

Proposition 3. *Let m_0 and m_1 be two well-formed patterns such that $obs_a(m_0) = obs_a(m_1)$. If the encryption scheme $\mathcal{AE}$ used in v is IND-CPA then m_0 and m_1 are indistinguishable.*

This result is immediate if we apply the above theorem: as obs_a returns the same result for the two patterns, $|\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}|$ is equal to zero. Hence as $|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$ is negligible, the advantage of $\mathcal{A}$ is also negligible and we get the desired result.

5 Application: Chaum's Visual Electronic Voting

To illustrate our results, we consider a slightly modified version of the electronic voting scheme proposed by Chaum [9]. The main advantage of this scheme is that it is verifiable using an audit procedure that preserves opacity of the votes [13], i.e., what did voter V vote?. However, this paper still makes the perfect cryptography hypothesis, encryptions are considered as black-box and are not taken into account. We give here a proof of security for Chaum's voting scheme in a computational setting. For that purpose, we assume that the encryption scheme is IND-CPA and prove that then, security results still hold (but we may have to add some negligible terms representing brute force attack against the encryption scheme).

5.1 System Description

Let us briefly recall how the Chaum's voting scheme works. We omit some important pieces (mostly the visual aspect) that are not relevant for this paper. The interested reader may consider reading [9] or [8] for details.

v_8^1	v_8^2	v_8^3	v_8^4	v_8^5
v_7^1	v_7^2	v_7^3	v_7^4	v_7^5
v_6^1	v_6^2	v_6^3	v_6^4	v_6^5
v_5^1	v_5^2	v_5^3	v_5^4	v_5^5
v_4^1	v_4^2	v_4^3	v_4^4	v_4^5
v_3^1	v_3^2	v_3^3	v_3^4	v_3^5
v_2^1	v_2^2	v_2^3	v_2^4	v_2^5
v_1^1	v_1^2	v_1^3	v_1^4	v_1^5

A vote session uses n trustees to guarantee the security of the procedure. Each trustee C_i has a public key pk_i and an associated secret key sk_i. The vote procedure works as follows: voters choose a vote value v, then the following bit-string is given to C_1 where each nonce N^i (unique for any voter) represents the random information used to compute the encryption layer using key pk_i:$\{...\{v; N^n\}_{pk_n}; N^1\}_{pk_1}$. Each trustee decodes its layer then makes a random permutation of all the votes and submits the resulting list to the next trustee. All the intermediate lists are made public and the last list allows anyone to compute the results of the vote.

After the decoding phase, an audit process allows to verify that trustees behave correctly with great probability. Hence each trustee C_i has to reveal the permutation it used for half the ballots. Thus it shows for these ballots the link between the input ballot encoded by pk_i and the output ballot encoded by pk_{i+1}, the trustee also shows nonce N^i to allow anyone to check that the link is valid (it is supposed that the encryption algorithm allows the trustee to get this nonce). Verified ballots are not chosen randomly but as described in figure above. The first set of verified ballot (for step 1) is chosen randomly. For step 2, verified ballots correspond to unconnected ballots w.r.t. step 1. For step 3, verified ballots are half unconnected ballots and half connected ones, the halves are chosen randomly. Finally, for step 4, verified ballots are unconnected ballots w.r.t. step 3. In the figure, v_j^i is the i^{th} ballots in the input of the j^{th} trustee and σ_j is the permutation chosen by this trustee. The set I_j consists of integers k such that the transition that reaches v_k^j is revealed.

5.2 Verification of the system

The property we are interested in is opacity of the vote. However, it should be possible to generalize our results to more complex properties like the bound over variation distance given in [13].

To simplify, let us suppose that there are two possible values for the vote: y and n. Then the set S of initial states contains all the vote distributions that give a fixed final result, i.e. for any element of S the number of voters that choose y is fixed, all the other variables are chosen at random (permutations, audit sets).

We study the opacity of property $\psi = (v_1^1 = y)$: are we able to deduce that the vote chosen by voter 1 is y ? We want to prove that the audit information cannot bring any advantage to an attacker. This requires that for any observation o, $pr(\psi|o) = pr(o)$. Then, as $PrRand$ is given by the vote result, it is clear that it is impossible to guess the value of v_1 with better efficiency than when answering the most probable vote with respect to the result. The specification of the system

is pretty straightforward: Δ_s outputs the revealed permutations, the ballots lists and the nonces used to check the link for any $k \in I_j$. The output pattern is well-formed (there are no cycles for $<$, the form of the ballots gives $pk_n < pk_{n-1} < ... < pk_1$).

After applying obs_a, the abstract system gives information on the permutations and the final ballot line, indeed there are two cases for remaining encrypted ballots: they can be abstracted to $\diamondsuit^N$ or they can be linked by a permutation to a vote in the final (unencrypted) line and so these ballots are useless to the description because it would be possible from the rest of the description to rebuild them using the final vote and the revealed nonce. Let o be an observation in the formal world. Let p_y be the percentage of voter that choose y. Then a quick calculus detailed in appendix A.3 gives us that $pr(v_1^1 = y|o) = p_y$. This proves opacity of ψ in the abstract world.

Security in the computational world is easy to obtain by applying our composition theorem: let $\mathcal{A}$ be a PRTM, then there exist $\mathcal{A}^o$ and $\mathcal{B}$ two PRTMs such that:

$$|\mathbf{Adv}_{\mathcal{A}}^{obs_c \times obs_a}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}| + 2.|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$$

The advantage related to obs_a is zero. Moreover, if we consider that the encryption scheme used is IND-CPA, then $\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}$ is negligible. Thus the advantage of $\mathcal{A}$ is negligible and we can conclude that the observable obs_c is safe for the cryptographic opacity of ψ.

6 Formal Description of the Main Result

The aim of this section is to prove our main result. We proceed in two steps. We first define n-RPAT-CPA and relate it to IND-CPA. Then, we prove that the advantage of any adversary who accesses the observation functions obs_a and obs_c is bounded by a linear combination of the advantage of an adversary that has access to obs_a and the advantage of an adversary that has access to obs_c. In both steps, we apply Theorem 1. Although, the criterion we introduce is implied to IND-CPA, it is technically more appealing to use it to prove the main result. Besides this, our new criterion is of interest on its own as it clarifies and discloses some subtleties related to the treatment of random coins.

6.1 The RPAT Extension to IND-CPA

In IND-CPA, the experiment consists of generating a random bit b and a random public key pk. The adversary tries to guess the value of b. For that purpose, it accesses a *left-right oracle* submitting two bit-strings bs_0 and bs_1 and receives the encryption of bs_b using pk. The adversary also has access to the public key. An encryption scheme $\mathcal{AE}$ is said secure against IND-CPA if any PRTM has a negligible advantage in trying to find b (the advantage is two times the probability to answer correctly minus one). The criterion we introduce below allows the adversary to ask for encryption of patterns where challenged keys may be included and insisting on using the same random coins in different encryptions. Moreover, patterns may include encryption with the adversaries keys. As we show later these extensions do not give more power to an adversary, if he is deemed to produce well-formed patterns.

Let us now introduce n-RPAT-CPA. To do so, let n be a non-negative integer. We first define R-patterns:

$$rpat ::= bs|N|\langle rpat, rpat\rangle|\{rpat; N\}_k|\{rpat; N\}_{bs}|\{rpat; bs\}_k|\{rpat; bs\}_{bs'}|k$$

The only difference with respect to patterns introduced in Section 4 is the encryption with a non-challenge key or a non-challenge nonce. The evaluation function v is extended to R-patterns.

The experiment defining the criterion n-RPAT-CPA is as follows:

$$
\begin{array}{ll}
pat_0, pat_1, \sigma \leftarrow \mathcal{A}_1; & \\
b \leftarrow \{0,1\}; & \\
(bs_i, bs'_i) \leftarrow \mathcal{KG}(\eta); & \text{for } i = 1, \cdots, n \\
bs''_i \leftarrow \{0,1\}^\eta; & \text{for } i = 1, \cdots, l \\
\theta \leftarrow [b, (pk_1, sk_1) \mapsto (bs_1, bs'_1), \cdots, (pk_n, sk_n) \mapsto (bs_n, bs'_n), & \\
\quad N_1 \mapsto bs''_1, \cdots, N_l \mapsto bs''_l]; & \\
y \leftarrow v(pat_b, \theta); & \\
d \leftarrow \mathcal{A}_2(y, \sigma) & \\
V(d, \theta) \leftarrow b = d &
\end{array}
$$

The adversary is split up in two parts $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathcal{A}_1$ outputs two patterns pat_0 and pat_1. Pattern pat_b is computed (l is the number of nonces used by the pattern and n is the maximal number of keys that a pattern can use), it is given to $\mathcal{A}_2$ which has to answer the value of b. It is also possible to consider a single adversary $\mathcal{A}$ that access a left-right oracle F, giving it the two patterns. In this case, oracle F only answer its first call.

In the experiment of n-RPAT-CPA, it is mandated that $\langle pat_0, pat_1 \rangle$ is well-formed.

We show that algorithms secure w.r.t. IND-CPA are secure w.r.t. n-RPAT-CPA and as, there are algorithms strongly believed to verify IND-CPA, these algorithms also verify n-RPAT-CPA.

Proposition 4. *If an asymmetric encryption scheme is secure against IND-CPA, then it is secure against n-RPAT-CPA for any number of keys n.*

The proof is detailed in appendix A.4.This proposition can be generalized to a polynomial number of keys and nonces using the technique introduced in [16].

6.2 Composition Result

Our main result states that given a specification, the advantage of an adversary against the concrete system is lower than the advantage of the abstract system and the advantage of another adversary against n-RPAT-CPA.

Theorem 2. *For each adversary $\mathcal{A}$, there exist two adversaries $\mathcal{A}^o$ and $\mathcal{B}$ such that*

$$|\mathbf{Adv}_{\mathcal{A}}^{obs_c \times obs_a}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}| + 2.|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$$

The proof of the main theorem is given in appendix A.5

Using proposition 4, it is clear that if an encryption scheme is secure against IND-CPA, then it is secure against n-RPAT-CPA for any integer n. Therefor, we have

Corollary 1. *If the encryption scheme $\mathcal{AE}$ used in v is IND-CPA and obs_a brings negligible advantage to any adversary then obs_c brings negligible advantage to any adversary.*

Conclusion

Probabilistic opacity is far more realistic than classical opacity. However, our main result makes it simple to prove cryptographic opacity for systems involving cryptographic primitives by first proving opacity for the related abstract system and then using an IND-CPA cryptographic scheme. This composition result seems very general as it can be applied to get the classical Abadi-Rogaway result. Another interesting result is the implication from IND-CPA to the new criterion n-RPAT-CPA. This criterion allows us to consider systems where the random information used for public-key encryption is exchanged (usually, to allow checking of this encryption). This is necessary to deal with complex systems such as Chaum's vote protocol. Hence, our last result is the first (to our knowledge) proof of this voting scheme in a computational setting.

A natural extension of this work is to consider the case of active adversaries as in [3]. To do this, we need to consider simulation but modular proofs seems quite harder to obtain when using this relation. We also intend to extend our result to other cryptographic primitives such as digital signature, symmetric encryption or hashing as in [16]. Finally, it would be of interest to extend the computational security results for Chaum's voting scheme to properties given in [13].

References

1. M. Abadi and J. Jürgens. Formal eavesdropping and its computational interpretation. In *4th Intern. Symp. on Theoretical Aspects of Computer Software*, volume 2215 of *lncs*. springer, 2001.
2. Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness o f formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Sendai, Japan, 2000. Springer-Verlag, Berlin Germany.
3. M. Backes and B. Pfitzmann. Computational probabilistic non-interference, 2002.
4. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 220–230. ACM Press, 2003.
5. M. Bellare, A. Boldyreva, and J. Staddon. Randomness re-use in multi-recipient encryption schemes. In Y. Desmedt, editor, *Public Key Cryptography – PKC 2003*, volume 2567 of *lncs*. springer, 2003.
6. E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In C. S. Laih, editor, *Proc. of Asiacrypt'03*, volume 2894 of *LNCS*, pages 37–54, Taipei, TW, November-December 2003. IACR, Springer-Verlag.
7. Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y.A. Ryan. Opacity generalised to transition systems. Technical Report TR-2004-25, Verimag, Centre Équation, 38610 Gières, December 2004.
8. D. Chaum, P. Ryan, and S. Schneider. A practical, voter-verifiable election scheme. Technical Report 880, University of Newcastle upon Tyne, School of Computing Science, Dec 2004.
9. David Chaum. E-voting: Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, January/February 2004.
10. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proceedings of the 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, Edinburgh, U.K., April 2005. Springer. To Appear.
11. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
12. Riccardo Focardi and Roberto Gorrieri. A taxonomy of trace-based security properties for CCS. In *Proceedings of the Computer Security Foundations Workshop VII (CSFW '94)*, pages 126–137. IEEE, 1994.
13. Marcin Gomukiewicz, Marek Klonowski, and Mirosaw Kutyowski. Rapid mixing and security of chaum's visual electronic voting. In *Proceedings of ESORICS 2003*, October 2003.
14. J. W. Gray. Probabilistic interference. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 180–187, 1990.
15. Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proceedings of the 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, Edinburgh, U.K., April 2005. Springer. To Appear.
16. Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. (de)compositions of cryptographic schemes and their applications to protocols. Technical report, Verimag, Centre Équation, 38610 Gières, To Appear 2005.
17. Peeter Laud. Semantics and program analysis of computationally secure information flow. *Lecture Notes in Computer Science*, 2028:77+, 2001.
18. Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *IEEE Symposium on Security and Privacy*, pages 71–85, 2004.
19. Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *ESORICS*, pages 198–218, 1996.
20. Peter Y.A.Ryan. Mathematical models of computer security. In *Foundations of Security Analysis and Design*, number 2171 in LNCS, 2000.

A Appendix

A.1 Basic Properties

Using the definition of advantage given above, some rather obvious properties can be stated. They prove relations between different kind of advantages except the first one that details the value of $PrRand$. This proposition relates the value of $PrRand$ to the probability for an element of s to be in S.

Proposition 5. *Let $pr(\psi)$ be the probability for a random element of S to verify ψ. Then the greatest possible advantage is obtained by answering 1 if $pr(\psi) > \frac{1}{2}$ and 0 otherwise. Hence, the best probability is: $PrRand^{\psi} = \frac{1}{2} + |pr(\psi) - \frac{1}{2}|$*

Proof. Let $\mathcal{A}$ be an adversary that can only use the ϵ observation function. As $\mathcal{A}$ does not have access to any oracle. Its behavior can be represented by its probability p to answer 1. Of course, $\mathcal{A}$ also has probability $1 - p$ to answer 0. The probability that $\mathcal{A}$ answers correctly is:

$$\begin{aligned} pr(\mathbf{Exp}^{\epsilon}_{\mathcal{A}} \rightarrow true) &= pr(\psi).p + (1 - pr(\psi)).(1 - p) \\ &= 1 - pr(\psi) + p.(2.pr(\psi) - 1) \end{aligned}$$

Then, if $pr(\psi) \geq \frac{1}{2}$,

$$pr(\mathbf{Exp}^{\epsilon}_{\mathcal{A}} \rightarrow true) \leq pr(\psi)$$

In the other case,

$$pr(\mathbf{Exp}^{\epsilon}_{\mathcal{A}} \rightarrow true) \leq 1 - pr(\psi)$$

These two inequalities allow us to deduce the following.

$$\begin{aligned} pr(\mathbf{Exp}^{\epsilon}_{\mathcal{A}} \rightarrow true) &\leq max(pr(\psi), 1 - pr(\psi)) \\ &\leq \left|pr(\psi) - \frac{1}{2}\right| + \frac{1}{2} \end{aligned}$$

By looking at the $PrRand$ definition, an adversary cannot get a better probability to succeed with no observations. Thus it is clear that an adversary cannot get a positive advantage.

Proposition 6. *For any adversary $\mathcal{A}$, $\mathbf{Adv}^{\epsilon}_{\mathcal{A}} \leq 0$.*

If an adversary gets a negative advantage, it is possible to inverse its behavior (by inversing its output) and this may create a greater advantage.

Proposition 7. *For any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ of similar complexity such that $\mathbf{Adv}^{obs}_{\mathcal{B}} + 4.PrRand^{\psi} = 2 - \mathbf{Adv}^{obs}_{\mathcal{A}}$.*

The following properties state that some observation functions can bring no advantage compared to other observation functions. Basically, if an observation obs_2 has a "finer resolution" than obs_1, then the advantage related to obs_2 is greater than the one related to obs_1.

Proposition 8. *Let obs_1 and obs_2 be two observation functions such that for any pair of trace t, t' in $\Delta(S)$, $obs_2(t) = obs_2(t')$ implies $obs_1(t) = obs_1(t')$. Then, for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ of similar complexity such that $\mathbf{Adv}^{obs_2}_{\mathcal{B}} = \mathbf{Adv}^{obs_1}_{\mathcal{A}}$.*

If obs_1 and obs_2 are two observation functions, $obs_1 \times obs_2$ gives access to both simultaneously. Formally, let obs_1 and obs_2 be two observation functions from Σ to $\Sigma \cup \{\epsilon\}$. Then $obs_1 \times obs_2$ is an observation function from Σ to $\Sigma \cup \{\epsilon\} \times \Sigma \cup \{\epsilon\}$ such that the result on action a is $(obs_1(a), obs_2(a))$. It is clear that the sum of two observation functions gives a better advantage than the advantages related to one of the two functions.

Proposition 9. *Let obs_1 and obs_2 be two observation functions. For any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ of similar complexity such that $\mathbf{Adv}^{obs_1 \times obs_2}_{\mathcal{B}} = \mathbf{Adv}^{obs_1}_{\mathcal{A}}$.*

Relation with Classical Opacity

Probabilistic opacity is closely linked to the opacity notion introduced in [7]. We consider plausible deniability for probabilistic opacity and initial opacity with a static observation function.

Let Δ be a deterministic system and S be the set of initial states. Let Π be a labeled transition system whose set of initial states is S and that has the same behavior as Δ. Let *obs* be an observation function.

Proposition 10. *A predicate ψ over S is opaque with respect to obs iff for any adversary $\mathcal{A}$,*

$$\mathbf{Adv}_{\mathcal{A}}^{obs} < 2.\big(1 - PrRand^{\psi}\big)$$

Classical opacity can be linked to anonymity [19] and non-interference [12, 20] and the same thing can be done with probabilistic opacity.

A.2 Proof of proposition 1

Proposition *For any adversary $\mathcal{A}$ and observation function obs,*

$$|\mathbf{Adv}_{\mathcal{A}}^{obs}| \leq I_{obs}$$

Moreover, there exists an adversary $\mathcal{A}_{obs}$ which advantage is exactly I_{obs}.

Proof. This proof is achieved in three steps:

Step 1 First, consider the case where there is only one possible observation, $|O| = 1$. Then, the calculus defining $PrRand$ can be applied. Adversary $\mathcal{A}$ has a probability p (resp. $1-p$) to answer 1 (resp. 0).

$$\begin{aligned} pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true) &= pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true | s \in \psi).pr(\psi) + pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true | s \notin \psi).pr(\neg\psi) \\ &= p.pr(\psi) + (1-p).(1 - pr(\psi)) \\ &= (1 - pr(\psi)) + p.(2.pr(\psi) - 1) \end{aligned}$$

Then, if $pr(\psi) \geq \frac{1}{2}$,

$$pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true) \leq pr(\psi)$$

In the other case,

$$pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true) \leq 1 - pr(\psi)$$

These two inequalities allow us to deduce the following.

$$\begin{aligned} pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true) &\leq max(pr(\psi), 1 - pr(\psi)) \\ pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true) &\leq \big|pr(\psi) - \frac{1}{2}\big| + \frac{1}{2} \\ pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true) &\leq PrRand^{\psi} \end{aligned}$$

Hence, the advantage is negative.

Step 2 Now, it is possible to generalize the above result for any set O.

$$\begin{aligned} pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true) &= \sum_{o \in O} pr(o).pr(\mathbf{Exp}_{\mathcal{A}}^{obs} \to true | o) \\ &\leq \sum_{o \in O} pr(o).\big(\big|pr(\psi|o) - \frac{1}{2}\big| - \frac{1}{2}\big) \\ &\leq \frac{1}{2} + \sum_{o \in O} pr(o).\big|pr(\psi|o) - \frac{1}{2}\big| \end{aligned}$$

We introduce $PrRand^{\psi}$ in the former equation using its form $|pr(\psi) - \frac{1}{2}| + \frac{1}{2}$. Hence,

$$\begin{aligned}
\mathbf{Adv}^{obs}_{\mathcal{A}} &\leq \sum_{o \in O} pr(o).\big(\big|pr(\psi|o) - \frac{1}{2}\big| - \big|pr(\psi) - \frac{1}{2}\big|\big) \\
|\mathbf{Adv}^{obs}_{\mathcal{A}}| &\leq \sum_{o \in O} pr(o).\Big|\big|pr(\psi|o) - \frac{1}{2}\big| - \big|pr(\psi) - \frac{1}{2}\big|\Big| \\
&\leq \sum_{o \in O} pr(o).\big|pr(\psi|o) - pr(\psi)\big| \\
&\leq I_{obs}
\end{aligned}$$

Step 3 The machine $\mathcal{A}_{obs}$ which advantage is exactly I_{obs} is very simple:

Adversary $\mathcal{A}_{obs}(o)$:
$p \leftarrow pr(\psi|o)$
if $p \geq 0.5$, **return** $true$
else return $false$

Probability for the different *obs* equivalence classes are hardwired in the machine. As there are only a finite number of classes, $\mathcal{A}_{obs}$ works in polynomial time (w.r.t. the size of o). The advantage of this adversary can be computed in a similar way as step 1 and 2. □

A.3 Opacity of Chaum Voting Scheme

There are two cases to consider. First case, the link starting from v_1^1 is revealed.

$$\begin{aligned}
pr(v_1^1 = y|o) &= pr(v^2_{1\sigma_1} = y|o) \\
&= \frac{2}{n} \sum_{i \notin I_3} pr(v_i^3 = y|o) \\
&= \frac{1}{n} \sum_{i=1}^{n} pr(v_i^4 = y|o) \\
&= p_y
\end{aligned}$$

In the second case, a similar calculus can be done.

$$\begin{aligned}
pr(v_1^1 = y|o) &= \frac{2}{n} \sum_{i \notin I_2} pr(v_i^2 = y|o) \\
&= \frac{2}{n} \sum_{i \in I_3} pr(v_i^3 = y|o) \\
&= \frac{1}{n} \sum_{i=1}^{n} pr(v_i^4 = y|o) \\
&= p_y
\end{aligned}$$

A.4 Proof of proposition 4

In this section, we prove that IND-CPA implies n-RPAT-CPA. Henceforth, let $\mathcal{AE}$ be an encryption scheme. We proceed in three steps.

Let n-RPAT$_c$-CPA be the same criterion as n-RPAT-CPA except that adversaries can only output clean patterns, i.e. $\langle pat_0, pat_1 \rangle$ such that $dec(pat_0, pat_1)$ does not contain any nonce nor private key. Our first step consists in proving that IND-CPA implies 1-RPAT$_c$-CPA.

Lemma 1. *If $\mathcal{AE}$ is secure w.r.t. IND-CPA then it is secure w.r.t. 1-RPAT$_c$-CPA.*

Proof. Let us consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against 1-RPAT$_c$-CPA. We construct an adversary $\mathcal{B}$ against IND-CPA whose advantage the same as the advantage of $\mathcal{A}$.

Let k be the unique challenge key. As there are no cycles among keys, there does not exist any pair of nonces N, N' such that $(k, N) < (k, N')$. Hence relation $<$ is empty. For any nonce N such that $(k, N) \in E_<$, N appears in exactly one encoding (but this encoding can be used several times as in $\langle \{m; N\}_k, \{m; N\}_k \rangle$) and in this case it appears as a random coin.

The adversary $\mathcal{B}$ uses $\mathcal{A}_1$ and $\mathcal{A}_2$ as sub-machines. However, as $\mathcal{A}_1$ outputs patterns while $\mathcal{B}$ has to output messages, $\mathcal{B}$ has to simulate the evaluation function v using the IND-CPA left-right oracle. This is done using the function $vsim$ in the description of $\mathcal{B}$:

$$
\begin{array}{l}
pat_0, pat_1, \sigma \leftarrow \mathcal{A}_1; \\
y \leftarrow vsim(pat_0, pat_1); \\
d \leftarrow \mathcal{A}_2(y, \sigma); \\
\textbf{return } d
\end{array}
$$

We now have to describe the function $vsim$. First notice that nonces N that do not appear in $E_<$ appear encrypted in the patterns. Therefor, $vsim$ generates some random values for these nonces and creates the corresponding environment θ_{sim}. The context θ_{sim} is extended with public key k. Next, as pat_0 and pat_1 have the same obs_a (modulo renaming), the following recursive function $vrec_{\theta_{sim}}$ is applied to pat_0, pat_1:

$$
\begin{aligned}
vrec_{\theta_{sim}}(bs, bs) &= bs \\
vrec_{\theta_{sim}}(m_1.m_2, m_1'.m_2') &= vrec_{\theta_{sim}}(m_1, m_1').vrec_{\theta_{sim}}(m_2, m_2') \\
vrec_{\theta_{sim}}(\{m; N\}_k, \{m'; N\}_k) &= F\big(v(m, \theta_{sim}), v(m', \theta_{sim})\big)
\end{aligned}
$$

Note that for the last line, if $vrec_{\theta_{sim}}$ is called twice on the exact same patterns, then the same value has to be returned (so it is necessary to store the value although this is not done here to preserve simplicity). Finally, $vsim(pat_0, pat_1)$ returns $vrec_{\theta_{sim}}(pat_0, pat_1)$.

The experiments involving $\mathcal{B}$ and $\mathcal{A}$ are the same and as $PrRand$ is equal to $1/2$ for both criteria, the advantages of $\mathcal{B}$ and $\mathcal{A}$ are equal. □

The second step is to show that 1-RPAT$_c$-CPA implies n-RPAT$_c$-CPA for any n.

Lemma 2. *If $\mathcal{AE}$ is secure w.r.t. 1-RPAT$_c$-CPA then it is secure w.r.t. n-RPAT$_c$-CPA.*

Proof. Let us consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against n-RPAT$_c$-CPA.

Using the reduction Theorem 1, we split up the advantage between an advantage against $(n-1)$-RPAT$_c$-CPA and an advantage against 1-RPAT$_c$-CPA. We assume that adversary $\mathcal{A}$ accesses the left-right oracle F exactly once. Let k be a maximal key for $<$. The partition of θ is defined as follows: θ_1 contains key pairs k, k^{-1}, any nonce N such that $(k, N) \in E_<$. On the other hand, θ_2 contains the other informations from θ including the challenge bit. Oracle F_2 generates the encodings related to keys in θ_2 and F_1 those related to k.

As k is maximal, there are no keys k' different from k such that for a nonce N in θ_1 and any nonce N', $(k, N) < (k', N')$. This is why nonce N is only used as the random coins of an encryption using k.

F_1 can be separated into two layers G and H defined by:

$$H(\langle pat_0, pat_1 \rangle, \theta_2, \theta_2') = \big\langle v(pat_{b_2}, \theta_2), v(pat_{b_2'}, \theta_2') \big\rangle$$

$$G(\langle pat_0, pat_1 \rangle, b, \theta_1) = v(pat_b, \theta_1)$$

Where b_2 and b_2' are the challenge bits contained respectively in θ_2 and θ_2'.

Let pat_0 and pat_1 be two R-patterns such that $obs_a(pat_0) = obs_a(pat_1)$. Then both patterns are the concatenation of encodings and similar bit-strings. The call to F has to be simulated using F_1 and F_2. For that purpose, the valuation of their encodings is performed in a similar way as

in *vrec* except that F_1 and F_2 should only be called once. To achieve this, requests to F_1 and F_2 are stored in a single pattern as described for F_1 by function $vrec2$ which outputs a list of pair of patterns:

$$\begin{aligned} vrec2(bs, bs) &= [\,] \\ vrec2(m_1.m_2, m_1'.m_2') &= vrec2(m_1, m_1').vrec2(m_2, m_2') \\ vrec2(\{m; N\}_k, \{m'; N\}_k) &= \langle \{m; N\}_k, \{m'; N\}_k \rangle \end{aligned}$$

Then this list of pair $(\langle p_1, p_1' \rangle; ...; \langle p_n, p_n' \rangle)$ is transformed into the pair of list $\langle p_1; ...; p_n, p_1'; ...; p_n' \rangle$ which is the argument given to F_1. Another function should perform the same operation for keys different from k. After submitting the results to oracle F_1 and F_2, it it easy to rebuild the output of F.

Note that patterns submitted to F_1 and F_2 are pairs of encodings. F_2 receives two well-formed patterns that have the same obs_a and this is the same thing for G (both receives a concatenation of some $\diamondsuit^N$).

As F_2 only depends on θ_2, our partition is valid, criterion $(\theta_2; F_2; V_2)$ is $(n-1)$-RPAT-CPA and $(\theta_1, b; G; V_b)$ is 1-RPAT-CPA. The reduction theorem applies and gives that there exist two PRTM $\mathcal{A}^o$ and $\mathcal{B}$ such that

$$|\mathbf{Adv}_{\mathcal{A}}^{n-RPAT}(\eta)| \leq 2.|\mathbf{Adv}_{\mathcal{B}}^{1-RPAT}(\eta)| + |\mathbf{Adv}_{\mathcal{A}^o}^{(n-1)-RPAT}(\eta)|$$

A simple induction proves that as $\mathcal{AE}$ is secure against 1-RPAT$_c$-CPA, it is secure against n-RPAT$_c$-CPA for any integer n. □

Finally, we show that n-RPAT$_c$-CPA implies n-RPAT-CPA.

Lemma 3. *If $\mathcal{AE}$ is secure w.r.t. n-RPAT$_c$-CPA then it is secure w.r.t. n-RPAT-CPA.*

Proof. Let us consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against n-RPAT-CPA. As in step 1, an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is built such that $\mathcal{B}_1$ returns clean patterns. For that purpose, $\mathcal{B}_2$ is similar to $\mathcal{A}_2$. $\mathcal{B}_1$ executes $\mathcal{A}_1$ and computes $dec(pat_0, pat_1)$. Then it generates some keys and nonces and uses them for elements of *dec* on the answer of pat_0 and pat_1. The patterns remain well-formed and still have the same obs_a. $\mathcal{B}$ and $\mathcal{A}$ have the same advantage but $\mathcal{B}$ is an adversary against n-RPAT$_c$-CPA. As $\mathcal{AE}$ is secure against n-RPAT$_c$-CPA, it is also secure against n-RPAT-CPA. □

Proposition 4 is a simple consequence of the three above lemma.

Proposition *If an asymmetric encryption scheme is secure against IND-CPA, then it is secure against n-RPAT-CPA for any number of keys n.*

A.5 Proof of the Main Theorem

This theorem is an application of the reduction theorem 1. Let Δ_s be a specification. Let n be the maximal number of keys used by Δ_s. Then the experiment related to $obs_c \times obs_a$ can be reformulated as the following experiment:

- Θ is split up on two parts: Θ_1 generates the n pairs of keys (pk_i, sk_i) and l nonces n^i; Θ_2 generates the initial state s in S and the pattern $p = \Delta_s(s)$.
- We have two oracles: F_2 gives access to $obs_a(p)$, F_1 gives access to $v(p, \theta_1)$ which is $obs_c(p)$.
- V_2 verifies that the output b made by the adversary is equivalent to $s \in \phi$.

F_1 can be be cut in two layers. G corresponds to the left-right encryption algorithm for n-RPAT-CPA, $H(x, \theta_2, \theta_2')$ takes any argument as input x and outputs the pair $\langle p', p \rangle$ where p and p' are the patterns respectively contained in θ_2 and θ_2'.

It is now possible to apply the reduction theorem 1 to obtain that for each adversary $\mathcal{A}$, there exist two adversaries $\mathcal{A}^o$ and $\mathcal{B}$ such that

$$|\mathbf{Adv}_{\mathcal{A}}^{\gamma}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{\gamma_2}| + 2.|\mathbf{Adv}_{\mathcal{B}}^{\gamma_1}|$$

Moreover, γ is equivalent to the criterion related to $obs_c \times obs_a$, γ_2 is equivalent to the one related to obs_a. Finally, $\gamma_1 = (b, \theta_1; G; \lambda x.x = b)$, G is only the left-right oracle, hence this criterion is the n-RPAT-CPA criterion except that there is no oracle to view the public keys. As this criterion is weaker than n-RPAT-CPA, it is possible to conclude that with a different machine $\mathcal{B}$ (but still of comparable complexity),

$$|\mathbf{Adv}_{\mathcal{A}}^{obs_c \times obs_a}| \leq |\mathbf{Adv}_{\mathcal{A}^o}^{obs_a}| + 2.|\mathbf{Adv}_{\mathcal{B}}^{n-RPAT-CPA}|$$

C Spécification partielle du protocole de vote de J.Traoré

```
############## Spécification Vote Électronique

signature
 V1, TM, AS, M, intruder : principal;
 PK: principal -> pubkey;

hashfunctions
  h: message -> message;
end

# Hermes translates the axioms as a set of protocol rules that may be executed
# whenever and how often is necessary, however they are not used
# by the term unification algorithm

axioms
declare
x: message;
y: symkey;
begin
sign(asym, inv(PK(AS)), h(x,y)) =
symcrypt(sym, y, sign(asym, inv(PK(AS)), x));
end

role voter ( myname:principal;
           skv:privkey;
           as,tm,m: principal;
           cert:message)

declare
vote : nonce;
r, x, e, s : message;
y, b, c, sigma : message;


begin
new(r);
new(vote);
x := crypt (asym, PK(tm), vote);
e := h(x,r);
s := sign (asym, skv, e);
send ([myname, cert, e, s]);
recv (sign(asym, inv(PK(as)), e));
y := sign(asym, inv(PK(as)), e);
b := [x, y];
c := crypt (asym, PK(m), b);
sigma := sign (asym, skv, c);
send ([myname, cert, c, sigma]);
end

role administrator (ska:privkey)
declare
voter : principal ;
```

```
ev , dv : message ;

begin
recv ([voter, sign (asym, ska, [voter, PK(voter)]), ev, sign (asym, inv(PK(voter)), ev)]);
dv := sign (asym, ska , ev);
send (dv) ;
end

role bulletinBoard (as: principal)

declare
voter : principal;
cb: message;

begin
recv ([voter, sign(asym, inv(PK(as)),[voter, PK(voter)]),
      cb, sign(asym, inv(PK(voter)), cb)]);
end

scenario
parallel
forall i : int . administrator (inv(PK(AS)))
|forall i : int . bulletinBoard (AS)
|forall i : int .
 exists v : principal .
 voter (v,inv(PK(v)),AS,TM,M,
          sign(asym,inv(PK(AS)),[v,PK(v)]))
end
end


############## Spécification de la vérification du secret faible du vote
public
 V1, TM, AS, M, intruder,  PK

initial
  x_M = [[inv(PK(intruder))]]

always

forall P:voter . ( P.myname != intruder ) -> issecret(P.vote)
```