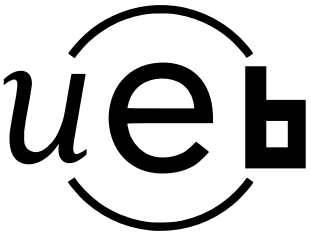
Tracing copies of multimedia documents with Tardos codes

Ana Charpentier

N° d'ordre : 4393 ANNÉE 2011



THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Ana CHARPENTIER

préparée à l'unité de recherche IRISA – UMR6074
Institut de Recherche en Informatique et Système Aléatoires IFSIC

Identification de copies de documents multimédia grâce aux codes de Tardos

Thèse à soutenir à Rennes le 21 octobre 2011

devant le jury composé de :

Sylvain DUQUESNE
IRMAR / *Président*

Philippe GABORIT
Professeur à l'université de Limoges, XLIM / *Rapporteur*

William PUECH
Professeur à l'université de Montpellier 2, LIRMM / *Rapporteur*

Gaétan LE GUELVOUIT
Orange Labs / *Examinateur*

Caroline FONTAINE
CR CNRS, Lab-STICC et Télécom Bretagne / *Directrice de thèse*

Teddy FURON
CR INRIA, Centre de recherche INRIA Rennes Bretagne Atlantique / *Co-directeur de thèse*

1

La propriété, c'est le vol
Proudhon

Table des matières

Chapitre 1

Introduction

Le développement de l'internet donne une part de plus en plus grande aux techniques de protection des contenus. Le *fingerprinting* est une technique destinée à lutter contre la redistribution incontrôlée de documents numériques. Dans cette thèse, nous nous intéresserons à la couche message du schéma de *fingerprinting*, c'est-à-dire au code anti-collusion. Nos travaux portent sur une famille particulière de codes, les codes de Tardos. Nous nous intéressons à l'optimisation des fonctions d'accusation de ces codes, ainsi qu'à la conception d'un protocole asymétrique permettant leur utilisation en pratique.

Le *fingerprinting*, tel que nous l'entendons dans ces travaux, est un procédé qui permet de lutter contre la distribution illégale de documents numériques en insérant un identifiant différent dans chaque copie distribuée. L'utilisation de ce terme est partagée avec une autre communauté de chercheurs pour lesquels le *fingerprinting* consiste à chercher les points remarquables d'un document numérique afin de procéder à son identification.

Dans les deux cas, il s'agit d'identifier un document, mais dans notre cas, nous cherchons à **identifier l'utilisateur** responsable de la distribution illégale en insérant une marque dans ce document (**le document est donc modifié**). Dans l'autre cas, on cherche à **identifier le document en lui-même** à partir de ses caractéristiques (**le document n'a pas été modifié**).

Un schéma de *fingerprinting* est composé d'un code de *fingerprinting* (dit code anti-collusion) et d'une technique de tatouage (technique de *watermarking*). Il peut être situé dans un environnement symétrique, asymétrique ou asymétrique anonyme, comme illustré par la figure 1.1. L'environnement concerne tous les échanges qui entourent le schéma proprement dit. Le code anti-collusion peut être étudié séparément, de même que la technique d'insertion, mais on voit aussi qu'il est pertinent de les considérer comme un ensemble.

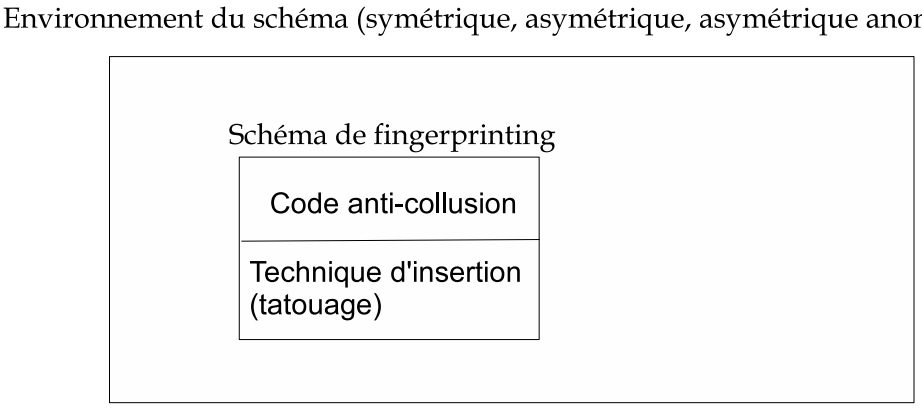


FIGURE 1.1 – Organisation d'un schéma de *fingerprinting*

1.1 Le contexte du traçage de documents

1.1.1 Présentation du problème

Nous sommes ici dans un contexte de marquage de documents numériques, avec un objectif de traçage des personnes qui diffuseraient le document de manière illicite. En l'absence d'une traduction française satisfaisante, nous conserverons le terme anglais *fingerprinting*.

Ce principe de personnalisation de document à des fins d'identification n'est pas récent, par exemple les tables de logarithmes de Néper étaient protégées de la façon suivante : un distributeur de ces tables avait décidé d'introduire des erreurs sur les décimales insignifiantes de $\ln x$ pour quelques de valeurs de x prises au hasard. Pour chaque copie de la table de logarithmes, le choix de ces valeurs x était différent. Si un possesseur d'une table essayait de distribuer des copies de sa table de façon illégale, ces petites différences permettaient à Néper de retrouver le distributeur fautif.
On peut aussi citer une anecdote concernant Margaret Thatcher qui ayant constaté des fuites d'informations distribua à chacun de ses collaborateurs des comptes-rendus légèrement modifiés afin de trouver le responsable.

Plus récemment, l'augmentation des échanges de documents sur internet rend cet outil d'identification très utile pour les distributeurs de contenus souhaitant en contrôler la diffusion.

Le contexte actuel est le suivant : un distributeur souhaite distribuer à plusieurs utilisateurs les copies d'un même document numérique. Seulement, l'usage de ces données est restreint et personne d'autre que ces personnes ne doit les posséder.
Si le possesseur du document original apprend qu'une copie supplémentaire a été dis-

tribuée à une personne non autorisée, il veut pouvoir retrouver l'origine de la fuite. Le *fingerprinting* est censé rendre possible la découverte d'un utilisateur coupable de distribution illicite. Pour cela, chaque copie contient une marque secrète d'identification, qui la rend unique : le *fingerprint*. Cette phase est schématisée par la figure 1.2.
La marque doit être insérée de manière à être invisible et faire corps avec le document, et un pirate ne doit pas pouvoir la détruire sans endommager le support. Elle doit aussi être résistante aux changements de format ou à la compression. Ces problèmes sont les mêmes que ceux liés au marquage d'image (*watermarking*).
Si un pirate seul distribue des copies de son exemplaire, elles porteront son identificateur. Si ces copies illégales sont découvertes, l'utilisateur malhonnête sera immédiatement identifié et pourra être poursuivi. Mais si les pirates sont plusieurs, ils vont pouvoir utiliser tous leurs exemplaires pour créer un faux, comme illustré par la figure 1.3. Ce type d'attaque est appelé attaque par collusion. Le procédé utilisé par les pirates pour créer ce faux a été considéré de différentes façons, en supposant que les pirates aient accès au message ou pas. Le but du distributeur est de trouver des familles d'identifiants (des codes) permettant de retrouver au moins un fraudeur ; les recherches sur ces familles d'identifiants sont très nombreuses, et les codes correcteurs d'erreurs ont été utilisés dans ce contexte. On peut trouver une présentation simple et assez complète du problème dans l'article [Fur09], qui est en français.

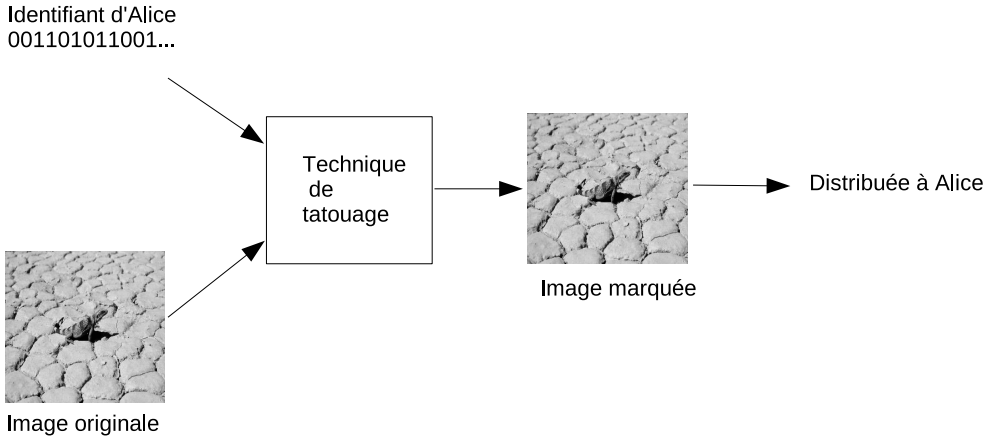


FIGURE 1.2 – Insertion de la marque et distribution

1.1.2 Le marquage de documents numériques

On appelle marquage (ou tatouage) l'insertion dans un document numérique d'une marque portant un ou plusieurs bits. Cette marque peut avoir plusieurs formes et dépend en général du support. Les bits portés par cette marque constituent l'identifiant, dit aussi *fingerprint* qui est utilisé dans le principe du *fingerprinting*. Cette insertion doit respecter certaines contraintes.

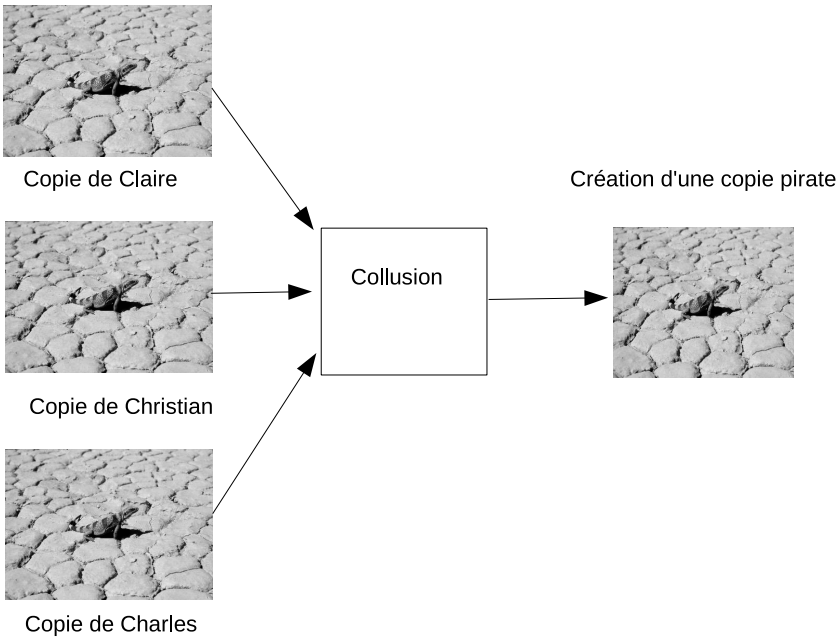


FIGURE 1.3 – Création d'une copie pirate par collusion

- **Invisibilité** : l'opération ne doit pas détériorer la qualité du document. L'utilisateur non averti est a priori incapable de discerner un document marqué d'un document non marqué.
- **Robustesse** : la marque ne peut pas être enlevée sans abîmer le document. Le tatouage est alors robuste aux attaques. Il peut s'agir d'attaques innocentes (compression, par exemple), ou malignes (attaques géométriques).
- **Capacité** : un compromis est à trouver aussi sur la quantité d'information que l'on souhaite insérer. L'insertion d'une faible quantité d'information sera moins visible et plus robuste que l'insertion d'une grande quantité d'information.
- **Sécurité** : l'utilisateur malhonnête ne doit pas pouvoir retrouver les clefs ayant servi à l'insertion. Il serait alors capable non seulement d'enlever la marque, mais aussi de la remplacer par celle de son choix. On peut voir sur ce problème l'article de Cayre *et al* [CFF05].
- **Complexité** : selon les moyens de calcul à disposition de l'appareil chargé de l'insertion, il est intéressant de trouver des techniques dont la complexité n'est pas trop élevée.

Le compromis entre ces contraintes n'est pas facile à avoir. En effet, plus le tatouage est robuste, moins il est invisible et réciproquement. Certains domaines d'étude spécifiques privilégient une contrainte par rapport à l'autre. Il existe par exemple des tatouages fragiles dont le but est de révéler la moindre modification effectuée sur le document. La figure 1.4 montre que les techniques existantes permettent une insertion

FIGURE 1.4 – Image tatouée. L'original est à gauche

complètement invisible.

1.2 Positionnement par rapport aux travaux précédents et organisation du document

Le *fingerprinting* sur des données numériques (Digital fingerprinting) a été introduit dans [BMP86] et ses bases formelles ont été posées par Boneh et Shaw en 1995 [BS95]. Dans cet article, les auteurs proposent un modèle d'attaque par collusion appelé *Marking Assumption* : les utilisateurs malhonnêtes qui construisent un *fingerprint* illégal ne peuvent pas détecter les positions auxquelles ils ont tous le même symbole. Notre travail se place sous cette hypothèse : si les *colluders* possèdent tous un '1' pour un certain bloc, il leur est impossible de construire le même bloc contenant un '0', donc la copie pirate contiendra forcément un '1' dans ce bloc (à moins qu'ils ne réussissent à l'effacer). Boneh et Shaw proposent aussi une construction de code anti-collusion basée sur l'emboîtement d'un code interne et d'un code externe.

Par la suite, ce schéma a été beaucoup étudié, en considérant différents types de codes internes et externes [Sch04]. Staddon, Wei et Stinson ont établi une classification des codes utilisés [SSW01], insistant sur les liens entre les codes correcteurs d'erreurs et le traçage de traîtres, ce travail a aussi été fait dans [CFNP00].

En 2003, G. Tardos [Tar03] est le premier à exhiber des codes traçants binaires dont la longueur atteint la borne inférieure théorique dite de Peikert : $m = O(c^2 log(n/\epsilon))$ [PSS03] (m désigne la longueur du code, c est le nombre de *colluders*, ϵ la probabilité de faux positif). Sa construction est un processus probabiliste des plus faciles à implémenter, où les probabilités d'erreur d'accusation (risque d'accuser un innocent, risque de n'identifier aucun pirate) sont contrôlées. B. Skoric *et al.* ont proposé une version symétrique de ces codes et ont étendu l'application à un alphabet q-aire [SKC08].

Le chapitre 3 présente un état de l'art des codes traçants, afin de mesurer comment les codes de Tardos sont différents des codes correcteurs d'erreurs qui ont été les plus étudiés. En annexe, le lecteur pourra trouver des rappels sur les codes correcteurs d'erreur, notamment les codes de Reed-Solomon, qui sont très utilisés comme codes de

fingerprinting.

1.2.1 Fonctions d'accusation des codes de Tardos

Les codes de Tardos ont la particularité de donner des résultats qui sont les mêmes quelle que soit la stratégie des *colluders* (quelle que soit la stratégie utilisée par les *colluders* pour forger une copie, les performances du code sont les mêmes). Cette constance a un intérêt, comparativement à d'autres codes qui sont construits pour résister très efficacement à certains types d'attaques et qui ne fonctionnent plus du tout si les pirates décident de faire une autre attaque, mais c'est aussi un inconvénient, puisque le code polyvalent est en général bien moins performant pour chaque type d'attaque que les codes construits spécifiquement pour ces attaques. Nous avons donc cherché à optimiser les performances du code en fonction des attaques.

Les codes de Tardos n'ont pas à proprement parler de partie décodage, au sens d'une opération qui donnerait en sortie un mot du code. L'équivalent de cette partie décodage pour les codes de Tardos consiste en un calcul de score, ce score servant à établir la participation d'un utilisateur à la création du faux. Pour calculer le score, on utilise des fonctions dites fonctions d'accusation. les codes de Tardos sont décrits dans la partie 3.3.

T. Furon *et al.* [FGC08b] ont démontré que les fonctions d'accusation proposées par Tardos/Skoric sont optimales dans un contexte général, mais aussi qu'il existe des fonctions plus efficaces si la personne conduisant l'accusation a des informations sur l'attaque qui a été réalisée.

Notre première contribution a été d'améliorer les fonctions d'accusation du code de Tardos en construisant une estimation à la volée de la stratégie ainsi que des fonctions qui tiennent compte de la stratégie utilisée par les *colluders* pour construire la copie pirate. Ces travaux ont donné lieu à une publication à SPIE 2009 [CXFF09], ainsi que dans la revue Traitement du signal [CFF10].

Le chapitre 4 concerne les fonctions d'accusation du code de Tardos, et leur modification afin de les rendre optimales en fonction de la stratégie de construction de la copie pirate.

Nous avons aussi conduit des travaux non publiés visant à mieux comprendre le cas q-aire de ces fonctions d'accusation. Ces travaux sont décrits dans la première annexe.

1.2.2 *Fingerprinting* asymétrique avec des codes de Tardos

Les schémas de *fingerprinting* peuvent être utilisés dans des environnements symétriques, asymétriques ou asymétriques anonymes. Nous décrivons les schémas ainsi que les environnements dans le chapitre 2. Le *fingerprinting* asymétrique est un concept introduit par Birgit Pfitzmann [PS96], [PW97b] en 1996. Dans ce schéma, l'utilisateur participe à la construction de son *fingerprint*, et est le seul à entrer en possession de l'exemplaire du document marqué avec ce *fingerprint*. Cela entraîne une totale réorganisation de la génération des *fingerprints*, ainsi que du protocole de marquage. L'environnement asymétrique a été peu étudié, comparativement au *fingerprinting* asymétrique

anonyme, [PW97a], qui est un schéma asymétrique garantissant en plus l'anonymat de l'acheteur. Ces schémas ont souvent fait abstraction des codes de *fingerprinting*, et notre contribution est de proposer un schéma de *fingerprinting* asymétrique totalement spécifié et opérationnel utilisant les codes de Tardos. Ces travaux sont détaillés dans le chapitre 5.

Ces travaux ont été publiés en 2011 à Information Hiding [CFFC11].

Chapitre 2

Etat de l'art sur les schémas de fingerprinting

Dans ce chapitre, nous allons décrire les éléments qui apparaissent sur la figure 1.1. Dans une première partie nous décrirons les schémas de *fingerprinting*, puis dans une seconde partie l'environnement de ces schémas.

2.1 Schéma de *Fingerprinting*

Nous discutons ici des différents schémas de *fingerprinting*, en nous appuyant sur une classification des attaques auxquelles ils doivent résister. Cette partie concerne les composantes du schéma que sont la technique de tatouage et le code anti-collusion (figure 2.1). Les différents environnements seront décrits dans la section 2.2.

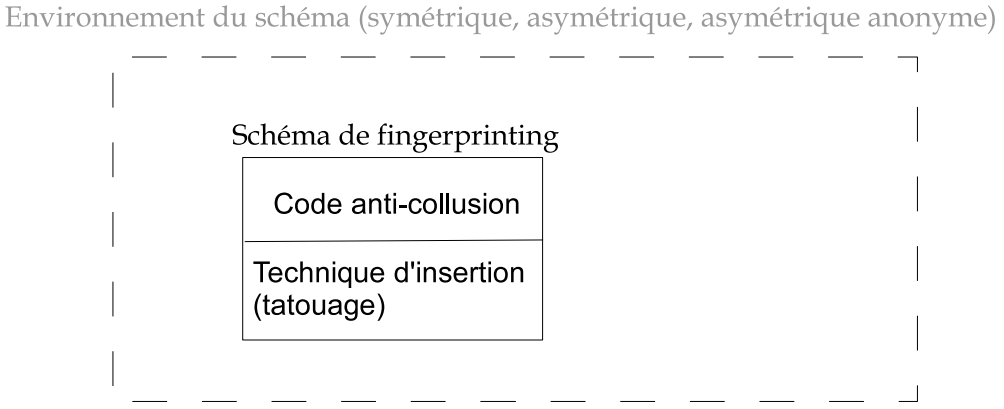


FIGURE 2.1 – La technique de tatouage et le code anti-collusion

2.1.1 Types d'attaques

Le schéma de *fingerprinting* doit résister aux attaques (malhonnêtes ou non) qui sont de trois types : attaques sur l'insertion (elles ne sont pas forcément malhonnêtes, le schéma d'insertion doit par exemple résister à la compression), qui concernent la technique de *watermarking* utilisée pour insérer l'identifiant ; attaques sur le code de *fingerprinting* ; enfin attaques de type fusion qui concernent à la fois l'insertion et le code.

2.1.1.1 Attaques sur l'insertion

Ces attaques ne sont pas spécifiques au *fingerprinting*, mais concernent toutes les techniques de *watermarking*. Le but de cette attaque est de détruire la marque en attaquant la technique de tatouage. Cette attaque peut être menée par un seul utilisateur malhonnête sur sa copie personnelle. S'il réussit à détruire son identifiant, le système de traçage est compromis puisqu'il pourra redistribuer sa copie attaquée sans pouvoir être identifié. Il y a deux façons principales de mener de telles attaques :

- attaque sur la robustesse : la première façon consiste à attaquer brutalement la robustesse de l'insertion avec des attaques de type compression, Stirmark (imprimer puis scanner l'image), agrandissement, *cropping* (prendre une partie du document), quantification, etc...
- attaque sur la sécurité : la deuxième façon de faire est plus subtile, il s'agit d'observer le document afin d'obtenir des informations sur les clefs utilisées pour l'insertion [CFF05]. Cette technique demande néanmoins, pour être efficace, de posséder plusieurs documents tatoués avec les mêmes clefs. C'est ce qui arrive dans une structure de marquage de vidéo, plusieurs séquences de la vidéo pouvant être tatouées avec les mêmes clefs et le même message (si ce message est un bit, par exemple).

Dans cette thèse, nous ne chercherons pas à contrer de telles attaques, cette tâche (ardue) étant laissée à la technique de tatouage que nous considérerons comme robuste.

2.1.1.2 Attaques sur le code

Un schéma de *fingerprinting* est conçu pour résister aux attaques par collusion. On parle de telles attaques lorsque les utilisateurs malhonnêtes se mettent à plusieurs pour créer une fausse copie qui sera redistribuée, comme illustré par la figure 1.3. Ces utilisateurs, appelés *colluders*, ont beaucoup de façons de créer cette fausse copie. Ils peuvent, pour une vidéo, faire un découpage temporel et entrelacer les morceaux de leurs différentes copies. Si l'insertion a elle aussi été faite de manière temporelle, la technique d'insertion aura beau être extrêmement robuste, elle n'empêchera pas la création d'une copie contenant une marque inédite faite de morceaux des marques des colluders. C'est à ces attaques que nous nous intéressons, celles qui touchent le mot qui a été inséré. Pour les contrer, les mots ne doivent pas être pris n'importe comment, ils doivent comporter une structure appropriée. C'est pour construire cette structure que nous utiliserons des

Uniforme les *colluders* choisissent au hasard un bloc parmi leurs copies :

001010111
101100100
001010100
001000110

Majorité les *colluders* choisissent le bloc le plus fréquent :

001010111
101100100
001010100
001010100

Minorité les *colluders* choisissent le bloc le moins fréquent :

001010111
101100100
001010100
101100111

All1 si ils ont au moins un '1', les *colluders* mettent un '1'. On suppose pour cette attaque et la suivante que les *colluders* sont capables d'identifier les '0' et les '1'

001010111
101100100
001010100
101110111

All0 si ils au moins un '0', les *colluders* mettent un '0' :

001010111
101100100
001010100
001000100

FIGURE 2.2 – Exemples d'attaques par collusion sur la couche code.

mots issus d'un code.

Les attaques étudiées par la communauté de codage suivent pour la plupart le modèle de la marking assumption décrit en détail dans le paragraphe 2.1.2.1, ce modèle constituant un cadre très strict dans lequel il est facile d'évoluer pour l'étude des codes anti-collusion. Quelques exemples d'attaques par collusion sur la couche code sont donnés dans la figure 2.2

2.1.1.3 Attaques par fusion

La troisième famille d'attaque est celle des attaques par fusion, qui touchent à la fois la technique de tatouage et la couche de code. C'est aussi une attaque par collusion. Lors de telles attaques, les *colluders* mixent leurs copies, non pas par morceaux, (type Frankenstein), mais plutôt dans leurs atomes (comme ce qui arrive à Jeff Goldblum et

marque du *colluder* 1 :	4	2	**6**	**2**	*2*	**1**
marque du *colluder* 2 :	1	**5**	2	6	*2*	3
marque du *colluder* 3 :	**2**	3	5	1	*2*	4
marque fabriquée :	**2**	**5**	**6**	**2**	*2*	**1**

FIGURE 2.3 – Alphabet q-aire : en jaune apparaissent les positions vérifiant la *Marking Assumption*

à la mouche). Il peut s'agir par exemple d'une moyenne de pixels dans le domaine non compressé, ou alors d'une moyenne des coefficients d'ondelettes.

2.1.2 Différentes approches de la solution

2.1.2.1 Approche codage

Marking Assumption La plupart des travaux sur le *fingerprinting* s'appuient sur le modèle d'attaque proposé par Boneh et Shaw en 1998 [BS98] : les attaquants comparent leurs copies et les pirates ne peuvent détecter que les endroits où leurs marques diffèrent. Le fait qu'ils ne peuvent pas modifier ce que leurs identifiants ont en commun va nous servir à retrouver l'un d'entre eux. C'est ce qu'on appelle la *marking assumption*. Cette hypothèse de travail est illustrée par la figure 2.3, elle sera détaillée dans le chapitre suivant sur les codes anti-collusion.

Quelques variantes du modèle ont été considérées, prenant en compte la possibilité pour les attaquants de mettre n'importe quel symbole pour les positions non communes, autorisant les effacements, ...

Notons que le modèle d'attaque de Boneh et Shaw peut à première vue être considéré comme peu réaliste, car il a tendance à brider l'imagination et les moyens des pirates. Néanmoins, lors de la mise en place d'un système réel de protection, on peut s'arranger pour rendre ce modèle et la « marking assumption » effectifs. Classiquement, on va découper le document à personnaliser en blocs, et cacher un symbole de l'identifiant par bloc. Ainsi, le mélange opéré par les pirates aura des répercussions bloc par bloc, et on pourra raisonner de manière indifférenciée sur les blocs de document ou sur les symboles associés, la préservation d'un bloc donnant lieu directement à la préservation du symbole correspondant. Ceci permet par ailleurs de préparer des versions maîtres tatouées hors-ligne avant toute transaction, chacune contenant toujours le même symbole de l'alphabet ; ces versions maîtres serviront ensuite à composer les documents personnalisés à la volée lors des transactions : il suffira d'intercaler les bons blocs des versions maîtres pour composer le document personnalisé, en fonction de l'identifiant associé.

Niveaux de traçabilité Les études menées dans ce modèle reposent pour la plupart sur les structures combinatoires des codes correcteurs d'erreur, les identifiants étant

précisément des mots de code. Ils ont abouti à la définition de niveaux de sécurité quant au traçage, niveaux regroupés dans deux grandes catégories : la *traçabilité forte*, et la *traçabilité faible*. Dans les deux cas, on essaie d'éviter l'incrimination d'innocents, tout en s'assurant d'attraper un des coupables. Dans le premier cas, on ne tolère aucun faux pas, mais ceci implique d'utiliser de très longs codes, sur de gros alphabets [SSW01, HVLLT98], ce qui rend l'implémentation et le décodage très lourds [HW05b], et même impossible en pratique à l'aide des algorithmes de tatouage actuels. Des exemples de codes sont donnés dans le chapitre 3.

Dans le deuxième cas, on s'autorise quelques erreurs de jugement maîtrisées (borne sur la probabilité d'erreur), et on peut alors utiliser des codes beaucoup plus faciles à manipuler. Pour cette deuxième catégorie, on dispose de candidats binaires issus de la théorie des codes, comme par exemple [BS98, BBK03, Sch04]. C'est dans cette catégorie que se placent les codes de Tardos évoqués dans le chapitre suivant 3.3.

2.1.2.2 Approche signal

Il existe une autre approche du *fingerprinting* n'utilisant pas de code anti-collusion, qui consiste à utiliser comme identifiants des signaux orthogonaux [WWZ+03]. Dans cette approche, chaque utilisateur se voit assigner un signal comme identifiant tel que tous les signaux sont orthogonaux entre eux. Des techniques utilisant des séquences d'étalement de spectre ont été étudiées par [CKLS97]. Le fait que les signaux soient orthogonaux permet de retrouver des utilisateurs malhonnêtes après une attaque par moyennage (cette attaque est la plus considérée dans la littérature car très facile à mettre en place, d'autres attaques ont été considérées dans [ZWWL03])

He et Wu ont conduit plusieurs études sur une utilisation conjointe de codes de *fingerprinting* avec des méthodes orthogonales non codées ([HW05a], [HW06]). Dans leur conclusions, les méthodes utilisant un code ont une meilleure détection en terme de complexité que les méthodes orthogonales, mais leur traçabilité des *colluders* est moins bonne, c'est pourquoi elles ont construit une méthode mixte combinant ces deux approches.

2.2 Environnement

Dans cette section nous décrivons les différents environnements possibles pour les schémas de *fingerprinting* (figure 2.4)

Positionnement des environnements Par le terme de *fingerprinting* symétrique, nous désignerons le schéma original décrit dans l'introduction, au cours duquel le vendeur génère les *fingerprints* de tous les utilisateurs et les insère dans le document original. Attention, contrairement au vocabulaire de cryptographie, ce n'est pas la gestion des clefs qui est ici concernée par les mots symétrique et asymétrique.

En cryptographie, les termes 'symétriques' et 'asymétriques' concernent les clés de chiffrement et de déchiffrement. Dans un schéma de chiffrement symétrique, la clef de chiffrement est la même que la clef de déchiffrement (quelles que soient les personnes

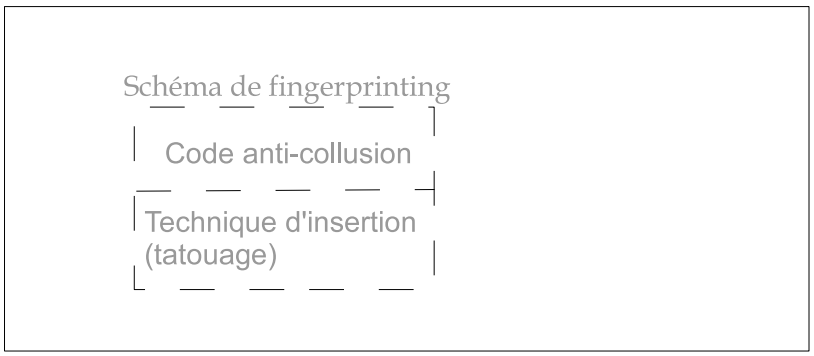


FIGURE 2.4 – Environnement

qui la possèdent). Dans un schéma de chiffrement asymétrique, il existe une clef de chiffrement et une clef de déchiffrement, celui qui ne possède que la clef de chiffrement ne peut pas déchiffrer le message. Il permet une organisation dans laquelle beaucoup de monde peut chiffrer un message si la clef de chiffrement est publique, tandis que le déchiffrement ne peut être effectué que par une seule personne détenant la clef secrète.

En ce qui concerne les schémas de *fingerprinting*, nous parlerons de schéma **symétrique** lorsque c'est le **vendeur seul** qui effectue la génération des identifiants ainsi que l'insertion de ces identifiants dans le document. Dans ce schéma, le vendeur construit le document marqué et le distribue ensuite à l'acheteur. Ce schéma est opposé au schéma **asymétrique** au cours duquel la **participation de l'acheteur** est demandée pour la génération de l'identifiant et/ou l'insertion dans le document, l'acheteur seul ayant en sa possession l'exemplaire du document marqué par son identifiant.
Un environnement asymétrique est dit anonyme si l'acheteur peut rester anonyme au cours de la transaction. Cet environnement rend indispensable l'intervention d'un centre d'enregistrement qui se situe entre l'acheteur et le vendeur. Un environnement asymétrique anonyme est un environnement dans lequel l'anonymat de l'acheteur est garanti même en cas d'entente entre le vendeur et le centre d'enregistrement.

2.2.1 Particularités de l'environnement symétrique

Dans le schéma de *fingerprinting* symétrique, le document marqué avec le *fingerprint* de l'utilisateur j est généré par le vendeur (qui maîtrise la génération des identifiants, ainsi que la technique d'insertion). Le vendeur distribue ensuite ce document marqué à l'utilisateur j. Dans cet environnement, le vendeur opère seul à toutes les étapes. L'acheteur ne connait pas son identifiant.

2.2.2 Particularités de l'environnement asymétrique

Dans l'environnement asymétrique, les tâches sont partagées. Selon les protocoles proposés, il peut y avoir plusieurs intervenants en plus du vendeur et de l'acheteur, comme un centre d'enregistrement ou un juge.

Les premières bases ont été posées par Pfitzmann et al [PS96] en 1996, qui ont d'abord proposé un schéma global pour arriver aux contraintes demandées, ainsi que des applications concrètes pour quelques phases du protocole. Ils ont ensuite proposé le concept de *fingerprinting* anonyme [PW97a]. Peu de recherches ont été menées sur le protocole asymétrique en lui-même, contrairement au protocole anonyme qui a suscité beaucoup plus de travaux. Nous nous intéressons dans cette thèse à l'intégration des codes de Tardos (voir 3.3) dans des protocoles asymétriques, c'est pourquoi les protocoles seront analysés en fonction de l'intégration du code traçant.

Un tel protocole doit remplir les 3 conditions suivantes :

- Condition 1 : le vendeur ne doit pas rentrer en possession de la copie marquée d'un acheteur.
- Condition 2 : l'acheteur ne doit pas rentrer en possession de la version non marquée du document.
- Condition 3 : le vendeur doit pouvoir tracer une copie pirate et retrouver un ou plusieurs utilisateurs malhonnêtes

Les travaux de Birgit Pfitzmann et al ([PS96], [PW97a], [PW97b]) constituent à notre connaissance l'intégralité de l'état de l'art sur les environnements asymétriques non anonymes. L'article [PW97a] qui introduit l'environnement asymétrique anonyme reprend des propositions concernant l'environnement non anonyme afin de les améliorer. Un article de Biehl et al [BM97] traite aussi du sujet en proposant une construction d'un schéma de *fingerprinting* basé sur des codes aléatoires et arrive à la même construction que Pfitzmann et al décrite dans [PW97b] C'est pourquoi nous proposons de détailler les constructions décrites dans ces articles en suivant chronologiquement les propositions de Pfitzmann et al.

2.2.2.1 Un schéma générique

Dans le plus ancien de ces articles, [PS96], il est proposé une vision très globale de ce que pourrait être un schéma de *fingerprinting* dans un environnement asymétrique, ainsi que des propositions de solutions pour certains points du protocole proposé. Nous n'allons pas donner l'intégralité de cet article mais nous trouvons intéressant de connaître l'évolution des solutions proposées.

Définition 1. Un schéma de *fingerprinting* asymétrique consiste en 4 protocoles - Génération des clefs, *fingerprinting*, identification et discussion.

- **Génération des clefs** : l'utilisateur génère une paire de clefs (sk_B, pk_B), qui correspondent à un couple clef privée/clef publique. La clef publique est accessible à tous les vendeurs et aux tierces parties.
- **Fingerprinting** : le protocole de *fingerprinting* est un protocole 2-parties entre le vendeur et l'utilisateur.

- **Identification** est l'algorithme qui permet au vendeur d'obtenir l'identité d'un *colluder* en analysant la marque extraite d'une copie forgée. Cet algorithme prend en entrée le document trouvé et le document original. L'algorithme, s'il réussit, rend en sortie l'identité d'un utilisateur, représentée par pk_B.
- **Discussion** est un protocole 2 ou 3-partie entre le vendeur, une tierce partie appelée arbitre et éventuellement l'utilisateur accusé. Le vendeur et l'arbitre entrent pk_B. Si l'utilisateur accusé prend part, il entre sk_B. La sortie de l'algorithme est un booléen qui vaut 1 si l'arbitre est d'accord avec le vendeur sur l'accusation de l'utilisateur.

La construction proposée par Pfitzmann et al dans cet article reste très vague sur les parties techniques, qui se retrouvent décrites par 'un protocole 2-partie' sans plus de détails. L'utilisation d'un code anti-collusion doit se faire au cours de ce protocole 2-partie qui reste à définir.

2.2.2.2 Des constructions plus précises pour des images

Après avoir proposé un protocole très global, Pfitzmann et al proposent des solutions plus précises pour certaines conditions. Nous décrivons donc un premier exemple de technique permettant l'insertion de la marque dans le document, ainsi que l'utilisation d'un code traçant.

2.2.2.3 Une solution pratique

Définition 2. Un schéma de **fingerprinting symétrique** est basé sur deux algorithmes : un algorithme de *marquage* et un *code*.

- l'algorithme de *marquage* est un algorithme probabiliste qui, étant donné une image, sélectionne un ensemble de positions possibles pour des marques. Chaque marque est un sous-ensemble de toutes les positions des pixels de l'image et va être utilisée pour encoder un bit. Les marques sont disjointes.
- *code* : il s'agit d'un **algorithme déterministe**. Son but est d'encoder les identités de telle sorte que certaines collusions ne puissent pas retrouver les marques en comparant leurs versions de l'image.

Pour leur solution, Pfitzmann et al considèrent le code comme une application déterministe. Cette construction ne peut donc pas être utilisée avec un code de Tardos, puisque ce code n'est pas déterministe.

Construction utilisant un engagement de bit (bit commitment) En considérant la définition donnée ci-dessus d'un protocole symétrique, les auteurs construisent un protocole asymétrique. Ce schéma est basé sur des engagements de bits **homomorphiques**. C'est le premier schéma complet de *fingerprinting* asymétrique, c'est pourquoi il nous semble intéressant de le détailler. $\boxed{b}$ représente l'engagement de b. Le *fingerprint* de longueur m est découpé en deux parties : une partie de longueur $m_1 = \lceil log_2(N) \rceil$

(N est le nombre de copies, c'est-à-dire le nombre d'utilisateurs) et le reste est de longueur m_2. La partie de longueur m_1 sert à encoder le numéro de la distribution. Doc représente le document non marqué.

Dans cette construction, le vendeur utilise un simple compteur no_seq pour différencier les différentes ventes de son document.

1. Le vendeur envoie à l'acheteur la valeur courante de no_seq puis l'incrémente.
2. L'acheteur choisit une valeur $id\_proof \in \{0,1\}^{m_2}$ **de façon aléatoire**. et obtient $id = (no\_seq, id\_proof)$. Il encode id en suivant le schéma de *fingerprinting* symétrique et s'engage sur le résultat $com = \boxed{code(id)}$. Il envoie $msg = (com, text)$ et une signature sig sur msg au vendeur.
3. L'acheteur prouve avec un protocole Zero-Knowledge qu'il connaît une valeur id dont le début est no_seq, et telle que com est un engagement sur $code(id)$
4. Le vendeur vérifie sig. Puis il **encode le document** en $\boxed{Doc}$ et multiplie les engagements obtenus dans l'image aux positions des marques : $com_i = \boxed{code(id)(i)}$. Il envoie le résultat $\boxed{Doc_{bought}}$ à l'acheteur.
5. L'acheteur déchiffre Doc_{bought} grâce à la propriété homomorphique du procédé d'engagement.
6. Le vendeur conserve $record_M = (no\_seq, pk_B, text, msg, sig)$.
   L'acheteur conserve $record_B = (no\_seq, texte, id)$.

Vendeur		Acheteur
no_seq	$\longrightarrow$	
		$id\_proof \in_R$, $id = (no\_seq, id\_proof)$
		$com = \boxed{code(id)}$
	$\longleftarrow$	$msg = (com, text)$, $sig(msg)$
		Prouve avec ZK que id commence par no_seq
vérifie sig		
$\boxed{Doc_{bought}} = \boxed{Doc}.\boxed{code(id)}$	$\longrightarrow$	obtient Doc_{bought}
		grâce à la propriété homomorphique
$record_M = (no\_seq, pk_B, text, msg, sig)$		$record_B = (no\_seq, texte, id)$

FIGURE 2.5 – *Fingerprinting* asymétrique avec utilisation d'un engagement homomorphique.

Dans cette partie, l'acheteur a choisi lui-même son identifiant qui reste totalement inconnu du vendeur. Le vendeur identifie un utilisateur malhonnête uniquement avec son numéro de séquence. Etant donné cela, la partie 4 au cours de laquelle l'acheteur donne une preuve que id commence bien par le numéro de séquence est fondamentale.

Lorsqu'une copie illégale Doc_{found} est trouvée, la partie identification se déroule comme suit :

- $identify_{asym}$: Avec les données Doc_{found} et Doc, le vendeur exécute $identify_{sym}(Doc_{found}, Pic, marks)$. Si l'algorithme aboutit, il faut vérifier que la sortie est bien de la forme $id = (no\_seq, id\_proof)$ avec no_seq qui est bien un numéro attribué dans $record_M$. Si c'est le cas, le vendeur retrouve l'entrée correspondante $record_M = no\_seq, pk_B, text, msg, sig$ et sort $pk_B, text$ et $proof = (id, msg, sig)$.
- ***dispute***.
  1. L'arbitre avec les données $(pk_B, text)$ et $proof = (id, msg, sig)$, vérifie que sig est une signature valide de msg avec pk_B et que la seconde partie de msg est *texte*. Si ce n'est pas le cas, le processus rend 'échec', sinon, il retrouve com à partir de msg.
  2. L'arbitre doit maintenant vérifier que $code(id)$ est bien le contenu de l'engagement com. Pour faire ceci, il demande à l'utilisateur accusé de donner des informations afin de prouver son innocence. Il donne à l'arbitre une preuve *Zero-Knowledge* que le contenu d'au moins un engagement de com n'est pas le bit correspondant de $code(id)$. Si l'acheteur réussit cela, l'arbitre répond 'rejeté', sinon, il répond 'OK'.

Cette méthode offre peu de résistance aux attaques par collusion. Dans ce protocole, la partie tatouage est décrite précisément, alors que l'utilisation du code n'est pas détaillée, mis à part l'information comme quoi le code correspond à une fonction déterministe.

2.2.2.4 *Fingerprinting* asymétrique basé sur des codes aléatoires

Ce schéma est proposé dans [PW97b], le deuxième article de B Pfitzmann sur le *fingerprinting* asymétrique, une construction similaire étant proposée quasiment au même moment par [BM97]. Au moment où est écrit l'article, la construction proposée par Boneh-Shaw [BS98] est considérée par l'auteur comme la meilleure, c'est pourquoi le schéma est basé sur cette construction avec un code interne et un code externe. Une description de cette construction est faite en 3.2. Là encore l'hypothèse est faite que le schéma d'insertion consiste en une collection de marques, chaque marque servant à insérer un bit.

2.2.2.5 Construction du schéma asymétrique

Soit c le nombre de *colluders*, N le nombre d'utilisateurs. Les trois paramètres du code concaténé sont l, n et d (l longueur du code externe, n taille de l'alphabet du code externe et nombre de mot du code interne, la longueur du code interne est $d(n-1)$). σ est un paramètre qui permet de faire varier la probabilité d'erreur.
Les paramètres utilisés sont les suivants : $l = 64c(\sigma + log_2(N))$, $n = 48c$ et $d = 2n^2(log(4nl) + \sigma)$.

- Génération de clef pour l'acheteur. Chaque acheteur génère une paire de clefs (sk_B, pk_B) pour le schéma de signature donné et distribue pk_B.

- Initialisation des données : Ce sous-protocole est opéré par le vendeur pour une entité du document à distribuer. Le vendeur sélectionne les positions des marques pour ces données.
- *Fingerprinting* : l'acheteur choisit **un mot de code au hasard** $word_B$ de longeur l sur l'alphabet $\{1, \ldots, n\}$.
  - L'acheteur fait un engagement com_B sur $word_B$ et l'envoie au vendeur. L'information pour l'ouvrir est appelée $open_B$.
  - L'acheteur signe le message $msg_B = (texte, com_B)$ avec sa clef secrète sk_B, où *texte* est un texte qui précise le contenu de la signature. Le résultat est appelé sig_B.
  - Le vendeur vérifie que sig_B est une signature valide sur msg_B grâce à pk_B.

  Pour les opérations suivantes, le protocole 2-partie est utilisé.
  - Entrées : L'acheteur fournit $word_B$ et $open_B$. Le vendeur fournit la donnée qui sera distribuée , les positions des marques, un ensemble aléatoire $Set_B \subseteq \{1, \ldots, l\}$ contenant $l/2$ éléments, et l'engagement de l'acheteur com_B.
  - Il est vérifié que Set_B contient bien $l/2$ éléments, et que $open_B$ permet d'ouvrir com_B.
  - Les symboles de $word_B$ qui sont aux positions de Set_B sont sélectionnés et donnés au vendeur. Ils constitueront $halfword\_trace_B$, alors que l'autre moitié sera appelée $halfword\_evid_B$.
  - **Les symboles de $word_B$ sont mis en correspondance avec les mots du code interne.** Le mot correspondant, qui est donc dans le code concaténé, est encodé aux positions des marques. Ceci donne alors le contenu marqué, qui est envoyé à l'acheteur.
- Pour la partie traçage, le vendeur a trouvé une version redistribuée du document, et identifié un mot binaire inséré dans les marques.
  - La première étape consiste à appliquer l'algorithme de traçage donné par le schéma de Boneh-Shaw.
  - Ceci donne le mot $word_{found} = y_1 \ldots y_l$. Le vendeur cherche alors parmi les paires $(Set_B, halfword\_trace_B)$ une correspondance sur au moins $l/(4c)$ symboles avec $word_{found}$. Il accuse l'acheteur concerné de tricherie et retrouve sig_B et $msg_B = (text, com_B)$.
- Dans la dispute, le vendeur utilise $proof = (msg_B, sig_B, word_{accuse})$ où $word_{accuse}$ est formé de $halfword\_trace_B$ aux positions de Set_B et des symboles de $word_{found}$ aux autres positions. Le juge vérifie la signature avec la clef publique de l'acheteur pk_B. Le vendeur est déclaré avoir raison si $word_{accuse}$ a au moins $l/2+l/(16c)$ symboles en commun avec $word_B$. Si cette affirmation est fausse, l'acheteur désavoue en ouvrant l'engagement ou en utilisant une preuve Zero-Knowledge.

Cette construction ne s'adapte pas à l'utilisation de codes traçants autres que les codes concaténés. En effet, la construction du schéma repose sur l'utilisation des deux codes : l'acheteur choisit un mot du code externe, qui est mis en correspondance avec le code interne dans un protocole 2-partie, donnant un mot du code concaténé. C'est le code concaténé qui a des propiétés traçantes. On peut voir que cette construction ne peut pas être utilisée avec des codes de Tardos.

2.2.3 Particularités de l'environnement asymétrique anonyme

Le développement du commerce sur internet permet d'acheter de multiples objets. L'anonymat y est possible beaucoup plus facilement que dans un magasin réel. Pour certains, l'anonymat est même souhaité, car les achats d'une personne révèlent des informations sur cette personne, informations qui peuvent se retrouver sur la toile et servir à des fins publicitaires ou autres. Dans un schéma de *fingerprinting* asymétrique, l'acheteur doit quand même s'identifier pour obtenir un document doté d'un *fingerprint*. Il est dommage que les acheteurs renoncent à leur anonymat pour satisfaire un schéma de *fingerprinting*. C'est pourquoi des schémas de *fingerprinting* anonymes ont été proposés.

2.2.3.1 Vue globale du schéma

Le problème est séparé en deux blocs : un bloc (bloc 1) qui gère l'enregistrement des utilisateurs, la génération des clefs, les contacts entre l'utilisateur et le centre d'enregistrement. Le résultat de cette étape est la valeur *emb* qui doit être insérée dans le document distribué. C'est ce bloc qui assure l'anonymat de l'utilisateur. Le bloc 2 gère l'insertion de *emb* dans le document. Cette étape mélange l'utilisation de code de *fingerprinting* et d'une technique de tatouage. Elle doit répondre à plusieurs contraintes :

- *emb* est connu de l'acheteur et doit rester inconnu du vendeur.
- L'image originale non marquée doit rester inconnue de l'acheteur.
- L'insertion doit avoir les propriétés suivantes :
  - **exactitude** : le procédé d'extraction associé à l'insertion appliqué à l'image distribuée doit permettre de retrouver *emb*
  - **récupération et résistance anti-collusion** : le procédé d'extraction doit permettre de retrouver au moins un des *colluders* sauf avec une probabilité négligeable.

La littérature propose en général des solutions pour un des deux blocs, rarement les deux. Les solutions concernant le traitement du bloc 1 sont beaucoup plus nombreuses que celles qui concernent le bloc 2. Nous proposons d'en résumer quelques-unes dans les paragraphes suivants. Des solutions pour le bloc 2 peuvent évidemment servir pour un schéma de *fingerprinting* asymétrique. Cependant, l'inverse n'est pas toujours vrai, car dans le cas d'un schéma anonyme, la valeur *emb* qui doit être insérée est construite par le bloc 1, elle est d'abord contrainte par la construction d'anonymat, alors que dans un schéma asymétrique non anonyme, *emb* a la seule contrainte du code traçant. Ce découpage est illustré sur la figure 2.6

2.2.3.2 Modèle et Schéma général (Pfitzmann 97)

Tout d'abord, dans l'article [PW97a], l'auteur s'intéresse au modèle et aux définitions.

L'identité d'un acheteur est ici définie par sa clef publique. Il peut s'enregistrer sous cette identité. Les endroits où cet enregistrement peut se faire sont appelés des **centres d'enregistrement**. **Attention : ici, on ne considère pas que ces centres sont dignes de confiance, et le modèle suppose qu'une collusion entre le**

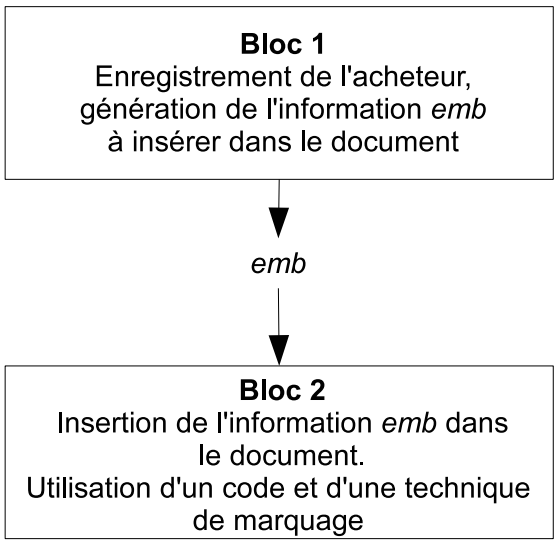


FIGURE 2.6 – Découpage d'un schéma de *fingerprinting* anonyme

centre et le vendeur est susceptible de se produire. Dans le modèle, la seule action malhonnête que peut faire un centre d'enregistrement avec succès est de refuser l'enregistrement.

Forces en présence Il y a quatre protagonistes : le vendeur, l'acheteur, le centre d'enregistrement et l'arbitre. L'arbitre est celui qui doit se faire convaincre lors de la dispute. Le rôle n'est donc pas restreint, il concerne n'importe qui qui doit être convaincu en connaissant un certain nombre de données publiques (les clefs publiques, par exemple). Nous supposons que le vendeur est malhonnête et essaie de connaitre l'identifiant de l'acheteur. Nous supposons que l'acheteur est malhonnête et essaie d'effacer son identifiant, en faisant les attaques décrites en 2.1.1. Le centre d'enregistrement est susceptible de communiquer des informations au vendeur.

2.2.3.3 Composants

Voici les composants d'un schéma de *fingerprinting* anonyme tels que proposés par Pfitzmann [PW97a]. Les solutions élaborées par la suite conservent ces quatre composants principaux.

Définition 3. Composants d'un schéma de *fingerprinting* anonyme. Un tel schéma consiste en quatre protocoles principaux. Tous ces algorithmes sont éventuellement probabilistes et peuvent répondre '*failed*'.

1. *Enregistrement - Bloc 1* : Au cours d'un protocole 2-parties, l'acheteur s'enregistre auprès du centre d'enregistrement. Les entrées connues sont les clefs publiques pk_B

et pk_{RC}, et éventuellement un nombre N_B correspondant au maximum d'achats que l'acheteur peut faire sous cet enregistrement. Le centre d'enregistrement met en entrée sa clef secrète. Les sorties sont les traces d'enregistrement conservées par l'acheteur et le centre d'enregistrement.

2. *Fingerprinting - Bloc 2* : Le *fingerprinting* a lieu au cours d'un protocole 2-parties entre le vendeur et l'acheteur anonyme. Le vendeur fournit le document à vendre et la trace des ventes précédentes, ainsi que la clef publique du centre d'enregistrement auprès duquel est enregistré l'acheteur. L'acheteur fournit la trace de son enregistrement, et les deux rentrent un texte commun qui décrit les circonstances de la transaction. Le résultat pour le vendeur est une trace de la transaction. Le résultat pour l'acheteur est le document marqué. Il peut aussi obtenir une mise à jour de son journal d'achat.
3. *Identification* : Il s'agit à cette étape d'un algorithme que le vendeur peut appliquer seul, ou avec l'aide du centre d'enregistrement. Le vendeur pose en entrée la donnée redistribuée dont il veut connaître l'acheteur original, la version originale de ce document, le journal venant de l'initialisation de données, ainsi que tous les journaux de vente de ce document. La sortie est l'identité d'un acheteur, le texte correspondant à cette transaction, et une autre donnée appelée $proof$.
4. *Procès* : C'est un protocole 2 à 4 parties entre au moins le vendeur et un arbitre, et éventuellement un acheteur et le centre d'enregistrement. Les entrées sont l'identité de l'acheteur accusé et le texte correspondant à la vente concernée. Le vendeur fournit aussi la donnée $proof$ obtenue lors de l'identification.

2.2.3.4 Solutions pour le bloc 2

Fingerprinting asymétrique avec un procès à 2 parties Dans ce schéma, proposé dans [PW97a], il n'y a pas de supposition de culpabilité que l'acheteur a à désavouer. L'idée principale est de fixer des mots de code qui peuvent seulement être changés dans leur globalité et qui peuvent identifier un seul *colluder*. Ainsi, le vendeur trouvera dans une copie illégale certains de ces mots intacts.

2.2.3.5 Construction

Initialisation des données . Le vendeur choisit des positions de marques pour le document selon sa méthode de tatouage. Pour chacune des l positions du code externe, il choisit une permutation aléatoire π_i sur l'alphabet $\{1, \ldots, q\}$.

Insertion Soit *emb* la valeur qui doit être insérée. Le vendeur possède un engagement de cette valeur.

- Le vendeur sélectionne K_1 bits aléatoires pour chacun des l symboles du code externe. On les appelle $halfsymboles$ et on a

$$halfword\_search_B = halfsym\_search_{B,1}, \ldots, halfsym\_search_{B,l}$$

– *emb* **est encodé avec le code correcteur d'erreur en** l **halfsymboles de** K_2 **bits**. C'est l'acheteur qui fait cela, et il devra cacher le résultat dans un engagement et prouver par la suite que son calcul est valide. Les *halfsymboles* du vendeur et de l'acheteur sont réunis par l'opération suivante :

$$sym_{B,i} = \pi_i(halfsym\_search_{B,i}||halfsym\_emb_{B,i})$$

Cette étape et la suivante sont réalisées au cours d'un protocole 2-partie. Le mot de code externe obtenu est donc

$$word_B = (sym_{B,1} \dots sym_{B,l})$$

– Chaque symbole $sym_{B,i}$ est codé suivant le code interne Γ_0 et le résultat est inséré dans le document. Le document marqué est envoyé à l'utilisateur.

Extraction – Pour chacune des l positions du code externe, le vendeur utilise la procédure d'identification du code Γ_0 pour identifier un symbole $sym_{red,i}$ (red pour redistribué). Il le déchiffre en utilisant π^{-1} et sépare le résultat en morceaux de longueur K_1 et K_2. On appelle le mot externe trouvé $word_{red}$ et le mot constitué des toutes les premières parties $halfword\_search_{red}$.
– Le vendeur cherche parmi ses fichiers le mot $halfword\_search_T$ qui a au moins (l/c) symboles en commun avec $halfword\_search_{red}$.
– Il essaie alors de trouver la valeur emb_T qui correspond aux deuxièmes parties de $word_{red}$. Il exclut les symboles $sym_{red,i}$ dont les premières parties sont différentes de celles de $halfword\_search_T$. Les symboles qui restent $halfword\_emb_{red,i}$ constituent un mot incomplet sur lequel il applique le code correcteur d'erreur et obtient emb_T.

2.2.4 Les protocoles d'insertion asymétrique par tatouage

Pour les problématiques que nous considérons, un protocole 2-partie qui réalise un tatouage asymétrique est une primitive très intéressante. Elle permet de réaliser la phase d'insertion au cours de laquelle le vendeur fournit le document non marqué et l'utilisateur fournit son identifiant. Les algorithmes connus se basent sur des fonctions homomorphes.

2.2.4.1 Description

Dans un protocole de tatouage asymétrique, le vendeur fournit le document original, l'acheteur fournit l'identifiant à insérer et seul l'acheteur a connaissance du document marqué. Ces techniques ne tiennent pas compte d'un code anti-collusion. Soit ce problème n'est pas considéré, soit c'est à la technique d'insertion d'avoir un rôle anti-collusion, mais cela ne s'applique pas en cas de collusion sur la couche message.

2.2.4.2 Utilisation d'un chiffrement homomorphique

Les schémas de chiffrement à clef publique homomorphiques (comme celui de Paillier [Pai99] ou de Okamoto-Uchiyama [OU98]) ont été utilisés dans plusieurs protocoles.

Définition 4 (Propriété homomorphique)**.** : Soit $E(m)$ une fonction de chiffrement d'un message m. La première opération est souvent la multiplication,
la deuxième f peut être addition, multiplication ou XOR selon les procédés. Cette fonction est celle qui gère l'insertion.
On a alors

$$E(m_1).E(m_2) = E(f(m_1, m_2)) \tag{2.1}$$

C'est la propriété qui est utilisée dans le schéma proposé par Pfitzmann afin de résoudre l'insertion dans un protocole 2-partie 2.2.2.3.
L'acheteur chiffre son identifiant, envoie le résultat au vendeur qui a de son côté chiffré le document à tatouer. Le vendeur applique la méthode de tatouage sur les données chiffrées et envoie le résultat à l'acheteur.

L'acheteur déchiffre et obtient l'image marquée en clair. Ces échanges sont illustrés sur la figure 2.7

2.3 Conclusion

Nous avons présenté dans ce chapitre les schémas proposés pour du *fingerprinting* symétrique, asymétrique , puis asymétrique anonyme. Les schémas proposés dans la littérature font le plus souvent abstraction des codes utilisés, ou utilisent des codes peu performants. Pour transformer un schéma asymétrique en schéma anonyme, il faut ajouter un bloc de construction de l'information *emb* (bloc 1) qui doit être insérée dans le document. Ce bloc est construit avec l'aide d'une entité supplémentaire, le centre d'enregistrement (sauf dans un modèle semi-honnête, où il est possible de se passer de tierce partie, comme dans [AGC10]). Puisque l'information *emb* est construite selon de nouvelles contraintes, le protocole d'insertion (bloc 2) doit parfois être modifié car *emb* ne peut pas être un élément d'un code anti-collusion. De nombreux travaux ont apporté des améliorations sur le bloc 1, proposés par la communauté des chercheurs en cryptographie (par exemple [PS99], [Cam00]), les recherches sur le bloc 2 ont plutôt été conduites par les personnes travaillant dans le domaine du traitement du signal (par exemple [DF02], [DFHJ00]). Ce positionnement à cheval sur plusieurs domaines explique la rareté de solutions entièrement spécifiées.

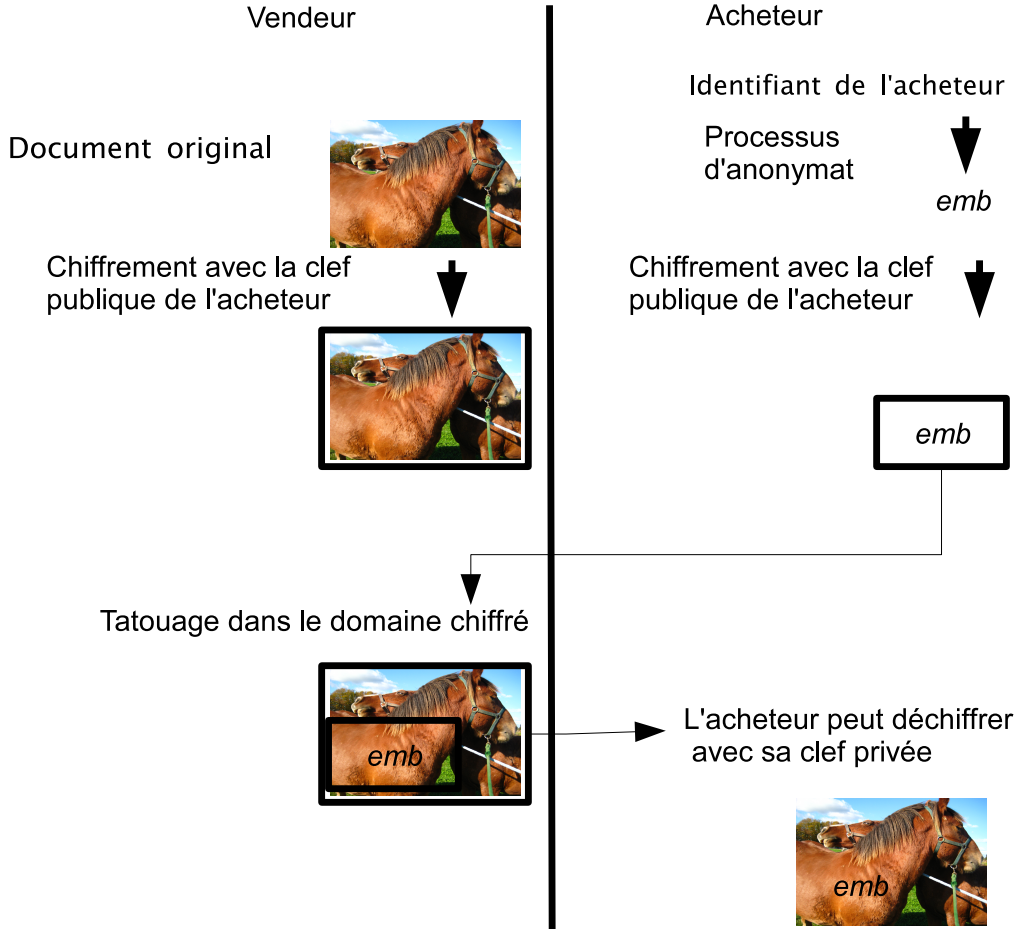


FIGURE 2.7 – Utilisation d'un chiffrement homomorphique pour l'insertion de l'identifiant de l'acheteur qui reste inconnu du vendeur.

Chapitre 3

Etat de l'art sur les codes anti-collusion

3.1 Types de codes anti-collusion

3.1.1 Définition

Nous reprenons ici des définitions données dans [BS98], qui est l'article fondateur pour l'étude des codes anti-collusion.

La définition est un peu différente de celle des codes correcteurs d'erreur.

Définition 5. Un (m, n)-code $\mathcal{C}$ est un ensemble de mots (n mots de longueur m), ces mots étant composés de symboles appartenant à un alphabet $\mathbb{Q}$. $\mathcal{C} = \{w_1, \ldots, w_n\} \subseteq \mathbb{Q}^m$ Le mot de code w_i sera associé à l'utilisateur i pour $1 \leqslant i \leqslant n$.

Nous travaillons principalement avec un alphabet binaire. La longueur m du code est la longueur des mots (tous les mots ont la même longueur). Un code peut aussi être parfois défini par la taille de l'alphabet sur lequel sont pris les mots : un (m, n, q)-code est donc un code ayant n mots de longueur m sur un alphabet de taille q.

3.1.2 La Marking Assumption

Nous décrivons ici le modèle de collusion appelé *Marking Assumption*, décrit en détail dans [BS98]. : c'est une hypothèse très forte qui contraint la fabrication de la copie pirate.

Définition 6. On appelle coalition un ensemble d'utilisateurs. Dans notre contexte, ce terme désigne l'ensemble des *colluders*, qui sont susceptibles de mener une attaque par collusion.

Définition 7. Soit $\mathcal{C}$ un (m, n)-code et $\mathcal{C}_0$ une coalition d'utilisateurs. Pour $i \in \{1, \ldots, m\}$, on dit que la position i est **indétectable** pour $\mathcal{C}_0$ si les mots associés aux utilisateurs dans $\mathcal{C}_0$ sont égaux à la i-ième position. Formellement, soit $\mathcal{C}_0 = \{u_1, \ldots, u_c\}$, alors la position i est indétectable si $w_{u_1}[i] = w_{u_2}[i] = \ldots = w_{u_c}[i]$.

Définition 8. : Soit $\mathcal{C}$ un (m, n)-code, $\mathcal{C}_0$ une coalition d'utilisateurs de taille c.
On appelle *ensemble des descendants* au sens strict ("feasible set" dans [BS98]) l'ensemble suivant :
$desc(\mathcal{C}_0) = \{\hat{w}$ *tel que* $\hat{w}[i] \in \{w_{u_1}[i], w_{u_2}[i], \ldots, w_{u_c}[i]\}$.

C'est donc l'ensemble des mots que l'on peut former avec les lettres des mots de la coalition (en les conservant à la même place). On considère que les positions identiques ne sont pas modifiées (c'est la *marking assumption*). Des exemples vont permettre de mieux comprendre :

Exemple 1. : Les utilisateurs A, B et C se sont vu assigner les mots suivants :

A : 1 **8** 3 **6** 3 8 **0** 5
B : 5 **8** 3 **6** 1 4 **0** 8
C : 6 **8** 2 **6** 1 4 **0** 1

Alors l'ensemble des descendants de (A,B,C) est de la forme $\{a\mathbf{8}b\mathbf{6}cd\mathbf{0}e\ \}$.
a est à choisir entre 1, 5 ou 6, b sera 3 ou 2, etc.
Il existe un modèle, dit problème au sens large, dans lequel a, b, c, d, e peuvent prendre toutes les valeurs de l'alphabet. On peut remarquer que dans le cas binaire, les deux modèles sont identiques.

Exemple 2. :

A : 1 **0 0** 0 **1** 1 **0**
B : 0 **0 0** 1 **1** 0 **0**

desc(A, B, C) = {0,1}**00**{0,1} **1**{0,1}**0**
Soit un ensemble à $2^3 = 8$ éléments.

Les positions *indétectables* pour (A, B, C) sont les positions sur lesquelles toutes les marques de la coalition sont égales (dans l'exemple précédent, les positions 2, 3, 5 et 7).

Marking assumption : les *colluders* ne peuvent pas modifier les positions indétectables.

3.1.3 Effacements

Boneh et Shaw ont introduit la notion d'effacement en supposant que les *colluders* ont la possibilité de rendre la marque illisible. Une position illisible sera notée '?'.

Définition 9. Soit $\mathcal{C}$ un (m, n)-code et $\mathcal{C}_0$ une coalition d'utilisateurs. Soit R l'ensemble des positions indétectables par $\mathcal{C}_0$. L'ensemble des descendants au sens large de $\mathcal{C}_0$ est alors

$$desc(\mathcal{C}, \mathcal{C}_0) = \{w \in (\mathbb{Q} \cup \{?\})^m\} \text{ tels que } w|_R = w^{(u)}|_R$$

Pour u un utilisateur dans $\mathcal{C}$.

marque du colluder 1 :	0	0	**1**	**0**	*1*	**1**	**1**	0	*0*	1	0	*1*	1	**0**
marque du colluder 2 :	0	**0**	0	1	*1*	0	0	1	*0*	**0**	**1**	*1*	**0**	0
marque du colluder 3 :	**1**	1	0	0	*1*	0	1	**1**	*0*	0	0	*1*	0	1
fausse marque	**1**	**0**	**1**	**0**	*1*	**1**	**1**	**1**	*0*	**0**	**1**	*1*	**0**	**0**

FIGURE 3.1 – Alphabet binaire : en surligné apparaissent les positions vérifiant la *Marking Assumption*

On note $\mathbb{Q} \cup \{?\} = \mathbb{Q}'$, et $F(\mathcal{C}, \mathbb{Q}) = F(\mathcal{C})$ en omettant $\mathbb{Q}$.

Exemple 3. Si deux utilisateurs A et B ont les mots suivants :

A : 1 2 3 2 1 3
B : 3 2 1 3 2 3

Alors leur ensemble faisable est $desc(A, B) = \mathbb{Q}'.2.\mathbb{Q}'.\mathbb{Q}'.\mathbb{Q}'.3$.

3.1.4 Types de codes

A l'accusation, il faut s'assurer que l'algorithme de décodage retrouve bien un des utilisateurs malhonnêtes. On cherche alors des codes ayant des propriétés spécifiques.
Le but pour le distributeur est de trouver une méthode sûre pour associer les marques aux utilisateurs.
Boneh et Shaw ont donné les premières définitions afin d'élaborer une classification des codes anti-collusion. Staddon et al [SSW01] ont continué ces descriptions. Ces définitions sont basées sur les propriétés de détection des *colluders* et de non-accusation des utilisateurs honnêtes. Cependant, ces propriétés ne constituent pas une classification, car il existe des codes n'appartenant à aucune de ces catégories.

Les exemples et définitions ont été puisés dans [FFG06], qui est déjà une synthèse sur l'état de l'art en *fingerprinting*.

3.1.4.1 Codes *frameproof*

L'objectif est de se protéger contre la redistribution naïve.
Un utilisateur redistribue sa copie sans l'avoir modifiée. S'il se fait prendre, il peut prétendre qu'une coalition a créé un objet avec sa marque.
C'est pour cela qu'un code doit aussi être *frameproof*, c'est-à-dire que la coalition ne peut pas impliquer un innocent.

Définition 10. : Un code $\mathcal{C}$ est *c-frameproof* (c-FP) si

$$\forall \mathcal{C}_0 \subset \mathcal{C} \text{ tel que } |\mathcal{C}_0| \leq c, desc(\mathcal{C}_0) \cap \mathcal{C} = \mathcal{C}_0$$

Exemple 4. Le code { 100, 010, 001} est 3-*frameproof*.
Les coalitions de taille 2 ne peuvent pas créer le troisième mot :
$desc(100, 010)$ = {100, 010, 110}-ne contient pas 001.
$desc(100, 001)$ = {100, 001, 101}-ne contient pas 010.
$desc(010, 001)$ = {010, 001, 011}-ne contient pas 100.
Et une coalition de taille 3 ne peut pas impliquer d'innocent puisqu'elle regroupe tous les utilisateurs.

3.1.4.2 Codes *secure frameproof*

Définition 11. Un code $\mathcal{C}$ est dit *c-secure frameproof* (c-SFP) si un mot ne peut pas être engendré par deux coalitions disjointes de taille au plus c.

$$w \in desc(\mathcal{C}_0) \cap desc(\mathcal{C}'_0) \Longrightarrow \mathcal{C}_0 \cap \mathcal{C}'_0 \neq \emptyset$$

Si un mot permet de remonter jusqu'à un groupe d'utilisateurs, les membres de ce groupe ne peuvent pas prétendre que le mot a pu être engendré par une autre coalition.

Exemple 5. Le code composé des mots suivants est 2-*secure frameproof* : c'est à dire que deux coalitions disjointes de taille 2 ne peuvent pas engendrer le même mot.

$$w_1 = (111)$$
$$w_2 = (100)$$
$$w_3 = (010)$$
$$w_4 = (001)$$

En effet, si on regarde toutes les coalitions de taille 2 :
la coalition formée par $\{w_1, w_2\}$ engendre des mots de la forme $w = (1**)$
et $\{w_3, w_4\}$ des mots de la forme $w = (0**)$.
Il est donc impossible que ces deux coalitions engendrent le même mot.
La coalition $\{w_1, w_3\}$ engendre des mots de la forme $w = (*1*)$
et $\{w_2, w_4\}$ des mots de la forme $w = (*0*)$.
La coalition formée par $\{w_1, w_4\}$ engendre des mots de la forme $w = (**1)$
et $\{w_2, w_3\}$ des mots de la forme $w = (**0)$.

3.1.4.3 Codes ayant la propriété *Identifiable parent property*

Définition 12. On dit que $\mathcal{C}$ satisfait la propriété *identifiable parent property (c-IPP)* si $\forall \hat{\mathbf{w}} \in \text{desc}(\mathcal{C})$,

$$\bigcap_{\mathcal{C}_o | \hat{\mathbf{w}} \in \text{desc}(\mathcal{C}_o)} \mathcal{C}_o \neq \emptyset. \tag{3.1}$$

Cela signifie que pour tous les mots de $desc(\mathcal{C})$, au moins un des parents peut être identifié.

3.1.4.4 Codes traçants

Définition 13. Le code $\mathcal{C}$ est dit *c-traçant* (c-TA) si le mot piraté $\hat{w}$ est plus proche (pour la distance de Hamming) d'un mot de la coalition (de taille c) que de n'importe quel mot hors de la coalition.

$$d(\hat{w}, (\mathcal{C}_0)) < d(\hat{m}, \mathcal{C} \setminus \mathcal{C}_0)$$

Exemple 6. (Code N-traçant). Posons $\mathcal{C} = \{(1, \ldots, 1); \ldots; (N, \ldots, N)\}$
Ce code de paramètres (L, N, N) est $N - tracant$. Tous les mots sont à distance L les uns des autres.
Pour tout utilisateur u_i ne participant pas à la coalition, $d(\hat{w}, w_{(u_i)}) = L$
La valeur de $\hat{w}[i]$ prouve que l'utilisateur numéro $\hat{w}[i]$ fait partie de la coalition : $d(\hat{w}, w_{\hat{w}[i]}) \leq L - 1$.

Théorème 1. *Relation entre les différents types de code [SSW01]*
- *c-TA implique c-IPP,*
- *c-IPP implique c-SFP,*
- *c-SFP implique c-FP.*

Notons qu'en général la réciproque n'est pas vraie.

Malheureusement, aucun code traçant existe pour un alphabet binaire [BS98].

Théorème 2. *Pour $c \geq 2$ et $n \geq 3$, il n'existe pas de codes c-traçant binaire.*

Il faut pour obtenir cette propriété travailler sur de très grands alphabets avec des mots très longs [Sch04], [Sch08].

3.1.5 Traçabilité faible vs traçabilité forte

Devant la difficulté à construire des codes traçants, on considére alors des codes *presque* sûrs.

Définition 14. : Le code $\mathcal{C}$ est dit *c-sûr (c-secure) avec une erreur* ϵ s'il existe un algorithme A de traçage qui vérifie la condition suivante : si une coalition $\mathcal{C}_0$ crée un mot w alors

$$\mathbb{P}(A(w) \in \mathcal{C}_0) > 1 - \epsilon$$

On voit ici que l'on peut séparer les codes anti-collusion en deux catégories : ceux qui vont assurer une *traçabilité forte*, c'est à dire qui permettent de retrouver un membre de la coalition avec une probabilité 1 (par exemple codes traçants et codes c-*secure*) et ceux qui assurent une *traçabilité faible*, c'est à dire qui permettent de retrouver un membre de la coalition avec une probabilité $1 - \epsilon$ (par exemple codes *frameproof*, codes *secure-frameproof* et codes *c-secure* avec une erreur ϵ). La traçabilité forte demande des longueurs de code énormes et des grands alphabets, ce qui la rend très contraignante. Dans la pratique, on se restreint à la traçabilité faible.

Lorsqu'on parle de traçabilité faible, on considère plusieurs types d'erreur : pour un code $\mathcal{C}$, une coalition $\mathcal{C}_0$, un algorithme de traçage A et un utilisateur j :

- la probabilité d'accuser à tort un utilisateur spécifique $\epsilon_1 = \mathbb{P}(w_{(j)} \in A(w) | w_{(j)} \notin \mathcal{C}_0)$,
- la probabilité qu'il y ait des utilisateurs innocents parmi les accusés $\epsilon_T = \mathbb{P}(A(w) \cap (\mathcal{C} \setminus \mathcal{C}_0) \neq \varnothing)$
- la probabilité de ne pas accuser des pirates $\epsilon_2 = \mathbb{P}(j \notin A(w) | j \in \mathcal{C}_0)$

Les probabilités ϵ_1 et ϵ_T sont liées de la façon suivante :

$$1 - \epsilon_T = (1 - \epsilon_1)^{n-c}$$

La probabilité ϵ_1 est très importante, et doit être très faible, alors qu'ϵ_2 peut atteindre de plus grandes valeurs. Il est en effet beaucoup plus important de ne pas accuser d'innocents que de laisser passer quelques pirates.

3.1.6 Liens avec les codes correcteurs d'erreurs

Pour lutter contre les attaques sur les messages, il existe peu de stratégies pour le *fingerprinting* qui ne font pas appel aux codes correcteurs (codes de Tardos, par exemple). L'utilisation de codes correcteurs est tentante car immédiate :

- les messages insérés dans les différentes copies distribuées sont des mots d'un code correcteur ;
- les pirates créent un faux à partir de leurs copies ;
- on applique l'algorithme de décodage sur la marque extraite du faux, et on retrouve un mot du code, c'est à dire un des utilisateurs (que l'on espère être un membre de la coalition, en regardant les propriétés du code utilisé).

L'état actuel des recherches en *fingerprinting* ne permet pas d'en faire un outil légal d'incrimination de fraudeurs, mais c'est un but. On voit que les propriétés décrites précédemment de non-accusation d'un innocent et de traçage sûr (avec une probabilité $1 - \epsilon$, mais c'est aussi le cas pour des méthodes comme l'ADN) d'une coalition sont indispensables à tout code utilisé dans un cadre de *fingerprinting*.

Le théorème suivant [CFNP00] lie les codes correcteurs aux codes anti-collusion :

Théorème 3. *: Soit $\mathcal{C}$ un code correcteur d'erreurs de paramètres (m, n,q) et de distance minimale $d > m(1 - c^{-2})$, alors $\mathcal{C}$ est c-traçant.*

Les bons codes anti-collusion sont ceux qui ont la plus grande distance minimale. Parmi les codes correcteurs d'erreur, les codes de Reed-Solomon ont la distance minimum qui atteint la borne de Singleton, c'est pourquoi ils ont été privilégiés dans les travaux sur les codes traçants. Pour une description des codes de Reed-Solomon, voir C.3.4 dans le manuscrit, ou bien [HP03].

3.2 Codes concaténés de Boneh-Shaw

Cette construction, proposée dans [BS98], consiste à utiliser les propriétés complémentaires de deux codes afin de réduire la longueur nécessaire (qui est extrêmement grande dans le cas d'un code correcteur d'erreur seul). Les premiers exemples proposés par Boneh et Shaw utilisent des codes à répétition sur lesquels ils appliquent des permutations.

Soit $\mathcal{C}_I$ un code de paramètres $(m_I, q, 2)$, appelé *code interne*, et soit $\mathcal{C}_O$ un code de paramètres (m_O, n, q), appelé *code externe* ; le code $\mathcal{C}$ résultant de la concaténation $\mathcal{C}_I \circ \mathcal{C}_O$ est un code de paramètres $(m_I m_O, n, 2)$ dont voici la construction.
Dans les mots de code de $\mathcal{C}_O$, on remplace les symboles de $\mathcal{Q}_O$ par des mots de code de $\mathcal{C}_I$, appelés *blocs* [Sch04], *coordonnées* [BBK03], ou encore *composantes* [BS02]. On obtient ainsi n suites binaires de longueur $m_I m_O$.

Le décodage d'un code concaténé s'effectue en décomposant la chaîne reçue en blocs de m_I bits. Puis on applique l'algorithme de décodage du code interne qui, à chaque bloc, va associer un ensemble de mots de code de $\mathcal{C}_I$. Cette suite de m_O mots de code est alors l'entrée de l'algorithme de décodage du code externe.

Du côté de l'encodeur, on applique d'abord le code externe, puis le code interne. Dans l'algorithme de traçage, on décode d'abord le code interne, puis le code externe. C'est ce qui explique cette terminologie interne/externe.

L'idée derrière tout cela est que le code interne va influer sur la robustesse face aux coalitions de taille c, mais pour un petit nombre d'utilisateurs. La concaténation avec le code externe permet de considérer un plus grand nombre d'utilisateurs tout en conservant les bonnes capacités de traçage du code interne. Mais il n'est pas facile de régler correctement un tel schéma : si l'on souhaite un bon code interne, avec une faible probabilité d'erreur (voire nulle si l'on souhaite une traçabilité forte), les blocs vont être longs. Il y a clairement un compromis à réaliser afin d'obtenir le meilleur des deux codes.

3.2.1 Boneh-Shaw naïf

Le principe des codes de Boneh-Shaw [BS98] est d'associer 2 codes ayant des propriétés complémentaires. L'utilisation la plus simple consiste à prendre pour un des codes un code à répétition. On considère le code $\mathcal{C}(n, r)$ de paramètres $(r(n-1), n)$ qui consiste en $n-1$ colonnes distinctes qui sont chacune répétées r fois. Ce qui nous donne par exemple le code formé par les six mots suivants :

$$\begin{aligned} w_1 &= (000000000000000000) \\ w_2 &= (000000000000001111) \\ w_3 &= (000000000011111111) \\ w_4 &= (000000001111111111) \\ w_5 &= (000011111111111111) \\ w_6 &= (111111111111111111) \end{aligned}$$

pour $n = 6$ et $r = 4$.

On voit que cette construction donne des mots de taille $r(n-1)$.
Pour que les pirates ne découvrent pas immédiatement la structure du code, on applique habituellement une permutation sur les mots. Les propriétés de ce genre de code font qu'ils sont utilisés comme code interne dans un code concaténé. Le code externe doit avoir une grande distance minimale, et les codes de Reed-Solomon sont de bons candidats, notamment grâce à l'algorithme de décodage de Guruswami-Sudan [GS98]. On obtient alors des codes vraiment très longs, ce qui est illustré par l'exemple suivant.

Théorème 4 (Répétition et permutation [BS98]). *Considérons $n \geq 3$, $1 > \varepsilon > 0$, et $R = 2n^2 \log(2n/\varepsilon)$. Le code $\mathcal{C}(n, R)$ vérifie les propriétés suivantes : quelle que soit la taille de la coalition, l'algorithme de traçage permet d'identifier au moins un des membres de cette coalition et la probabilité d'accuser au moins un innocent est inférieure à ε. Ce code, de longueur $m = O(n^3 \log(n/\varepsilon))$, est donc n-sûr avec ε-erreur.*

Exemple 7. [Sch06] On prend comme code interne un code de Boneh-Shaw de paramètres $q = 2^{10}$ et $r = 3,1.10^7$, et comme code externe , un code de Reed-Solomon généralisé de paramètres $[690, 2]_q$.
Pour $c = 20$ *colluders* et $n = 2^t$ utilisateurs, le code obtenu est de longueur $2,139.10^{10}$ et l'erreur est $\epsilon \leq 0,356.10^{-10}$.

3.2.2 Autres codes concaténés

Dans cette partie, nous décrivons quelques exemples de codes concaténés et leurs performances pour une utilisation dans un schéma de *fingerprinting*. Hans-Georg Schaathun a mené de nombreuses séries de tests sur les codes anti-collusion, les exemples décrits par la suite sont donc issus de ses travaux.

Exemple 8. Ce premier exemple montre la construction d'un code conçu pour combattre une coalition de 2 *colluders*. Ce code est étudié dans [Sch06] qui contient l'étude de plusieurs schémas de Boneh-Shaw utilisant différents codes. Il s'agit d'une concaténation entre un code de Reed-Solomon (RS) généralisé et un code simplexe (nommé Separating System (SS) dans [Sch06]) .
Le code interne est un code simplexe de paramètres [31,5] avec pour distance minimale 15. Ce code est $(2, 2^{-11}) - sr$, c'est à dire que pour une coalition de taille au plus 2, il existe un algorithme qui permet de retrouver un membre de la coalition avec une probabilité de $1 - 2^{-11}$.
Le code externe est un Reed-Solomon de paramètres [4,2] sur un corps à 32 éléments. Ce qui donne un code concaténé de paramètres [124,10] (de longueur 124, permettant d'encoder des mots de longueur 10, ce qui donne un nombre d'utilisateurs potentiels de 2^{10}) qui est $(2, 0.2 \cdot 2^{-12}) - sr$.

Pour voir à quoi ressemblent ces codes, voici leurs matrices génératrices :
code Simplexe(5)

```
[31, 5, 16] Cyclic Linear Code over GF(2)
```

```
Generator matrix:
[1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0]
[0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1]
[0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0]
[0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0]
[0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1]
```

et code de Reed-Solomon $[4, 2]_{32}$:

```
[4, 2, 3] "GRS code" Linear Code over GF(2^5)
Generator matrix:
[   1    0    w w^19]
[   0    1 w^18 w^11]
```

`w` est un élément primitif du corps fini à 32 éléments.

Exemple 9 (Code externe aléatoire [BS98])**.** Ce code concaténé est composé d'un code aléatoire comme code externe, et d'un code à répétition comme code interne.

Soit $\mathcal{C}_O$ un code de paramètres (m_O, n, q) et pour lequel les mots de code sont choisis au hasard et de manière uniforme. On prendra pour $\mathcal{C}_I$ le code $\mathcal{C}(m_I, R)$ défini au théorème 4. Soit $q = m_I = 2c$, $\varepsilon_I = 1/2c$, tels que $R = 8c^2 \log(8c^2)$. Pour le code externe, on prendra $m_O = O(c \log n/\varepsilon)$. Le code concaténé obtenu est c-sûr avec ε-erreur, et une longueur de $L = O(c^4 \log(m_I/\varepsilon) \log c)$ [Sch04].

Exemple 10 (Code externe de Reed-Solomon [BBK03])**.** Ce code concaténé est composé d'un code de Reed-Solomon comme code externe, et d'un code *secure frameproof* comme code interne.

Prenons pour $\mathcal{C}_O$ le code de paramètres (m_O, n, q) défini dans le théorème 3, avec une distance minimale vérifiant

$$d > m_O \left(1 - \frac{1}{c^2} + \frac{c-1}{c(q-1)}\right). \tag{3.2}$$

Les auteurs de [BBK03] ont choisi un code de Reed-Solomon. Soit $\mathcal{C}_I$ un code c-SFP de paramètres $(m_I, q, 2)$. Le code concaténé est alors un code c-secure dont la probabilité d'erreur est bornée par [Sch04] :

$$\varepsilon < n(q-2)^{-(m_O - 6(m_O - d))}. \tag{3.3}$$

L'opération de décodage n'est pas triviale. Premièrement, il n'y a pas de décodage efficace (et encore moins de construction) de codes c-SFP pour $c \geq 3$. Habituellement, la seule construction connue repose sur l'utilisation d'une table précalculée : aux $\binom{q}{c}$ c-ensembles $\mathcal{C}_{Io}$, on associe desc($\mathcal{C}_{Io}$). Le décodage du code interne analyse la table jusqu'à trouver le bloc $\hat{\mathbf{w}}_i$ dans un ensemble desc($\mathcal{C}_{Io}[i]$). La sortie n'est pas constituée d'un seul bloc (un mot de code de $\mathcal{C}_I$), mais d'une coalition $\mathcal{C}_{Io}[i]$. Mais attention, cette technique n'est gérable que si c est petit. Lors de la deuxième étape, on cherche à trouver le mot de code $\mathcal{C}_O$ le plus proche de l'ensemble $\mathcal{H} = \{\hat{\mathbf{w}} \in \mathcal{Q}_O^{m_O} | \hat{\mathbf{w}}[i] \in \mathcal{C}_{Io}[i]\}$. Un code externe de la nature des Reed-Solomon offre une solution efficace pour cette étape avec un algorithme de décodage en liste

3.2.3 Performances

Comme mentionné précédemment, les codes proposés dans l'exemple 10 ne peuvent gérer que des tailles de coalition petites. La table 3.1 récapitule les propriétés de quelques exemples pratiques. On notera que la longueur des codes suit en gros une progression en $\log n$ alors qu'elle explose lorsque c grandit. Les codes reposant sur l'exemple 9 sont plus longs, comme le montre la table 3.2. Néanmoins, il leur est plus facile de faire face à une grosse coalition pour un coût en complexité relativement faible.

n utilisateurs	longueur m	c traîtres	prob. d'erreur ε
2^{28}	1 386	2	$0,23.10^{-12}$
2^{42}	4 919	2	$0.14.10^{-10}$
2^{14}	111 872	3	$0,32.10^{-33}$
2^{14}	49 818	3	$0,61.10^{-10}$
2^{21}	66 424	3	$0,63.10^{-10}$
2^{32}	80 606	3	$0,55.10^{-10}$

TABLE 3.1 – Une famille de codes construits sur l'exemple 10. Résultats empruntés à [Sch04].

n utilisateurs	longueur m	c traîtres	prob. d'erreur ε
2^{10}	299 889	2	1.10^{-10}
2^{20}	367 359	2	1.10^{-10}
2^{30}	435 471	2	1.10^{-10}
2^{10}	$4,78.10^{8}$	10	1.10^{-10}
2^{20}	$6,56.10^{9}$	20	1.10^{-10}
2^{30}	$4,16.10^{10}$	30	1.10^{-10}

TABLE 3.2 – Une famille de codes construits sur l'exemple 9. Résultats empruntés à [Sch04].

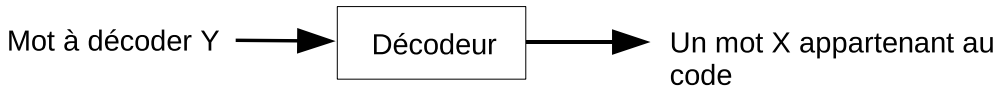
Les codes correcteurs d'erreurs permettent donc de construire des codes anti-collusion assez performants, à condition d'avoir des longueurs de mots extrêmement grandes. C'est un problème car ces mots doivent ensuite être insérés dans des documents numériques, et les techniques de tatouage ne permettent pas l'insertion d'aussi grandes données dans des conditions satisfaisantes.

Pour les codes anti-collusion, la longueur est un élément très important à considérer, on préferera un code plus court même s'il est moins performant.

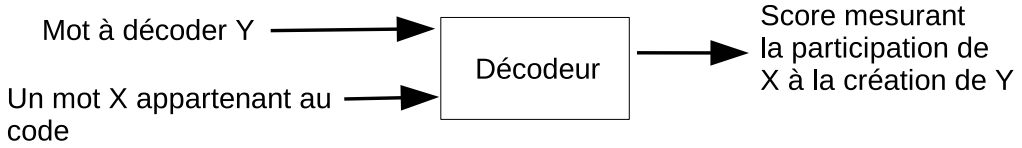
3.3 Codes de Tardos

Les codes de Tardos appartiennent aux codes à traçabilité faible. Ce sont des codes probabilistes, différents des codes linéaires. Il n'existe pas d'encodeur, et non plus de décodeur de ces codes. La génération de mots avec les mêmes paramètres peut donner des codes différents.

Ces codes, contrairement aux codes correcteurs d'erreurs, ont été construits spécifiquement pour le *fingerprinting* et atteignent la borne de longueur minimale ($m = O(c^2 \log(1/\epsilon_1))$) [PSS03]. La phase dite de décodage, qui n'en est pas vraiment une, ne donne pas en sortie un mot du code. Elle prend un mot du code en entrée avec le mot à décoder, et sort un score qui correspond à une mesure de l'implication de l'utilisateur concerné dans la création du mot à décoder. Cette différence est illustrée sur la figure 3.2.



(a) Décodage d'un code correcteur d'erreurs



(b) Décodage d'un code de Tardos

FIGURE 3.2 – Fonctionnement différent de la phase d'accusation selon l'utilisation d'un code correcteur d'erreurs ou d'un code de Tardos.

3.3.1 Les codes présentés par Tardos

Nous présentons les codes binaires de Tardos et l'accusation binaire symétrique de Skoric.
Dans la première version proposée par Gabor Tardos dans [Tar03], les paramètres utilisés sont les suivants : c est le nombre de *colluders*, $0 < \epsilon_1 < 1$ et $k = \lceil log(1/\epsilon_1) \rceil$, ϵ_1 étant la probabilité de fausse alarme, soit la probabilité d'accuser un utilisateur innocent. La longueur m dépend du nombre de colluders $m = 100c^2k$, soit n le nombre total d'utilisateurs.

Les mots de code distribués forment une matrice $n \times m$ binaire $\mathbf{X}$. L'utilisateur j se voit associé le mot binaire $\mathbf{X}_j = (X_{j1}, X_{j2}, \ldots, X_{jm})$.

- **Initialisation :** Pour générer cette matrice, m nombres réels $p_i \in [t, 1-t]$ sont distribués indépendamment selon la façon suivante : $p_i = sin^2 r_i$, avec la valeur r_i prise uniformément dans l'intervalle $r_i \in [t', \pi/2 - t']$ avec $0 < t' < \pi/4, sin^2 t' = t$ On note $\mathbf{p} = (p_1, \ldots, p_m)$.
- **Construction :** Chaque élément de la matrice $\mathbf{X}$ est ensuite indépendamment tiré en suivant la probabilité $\mathbb{P}(X_{ji} = 1) = p_i$.

Chacun de ces n mots de code est caché dans la copie délivrée à l'utilisateur associé comme expliqué dans l'introduction.

FIGURE 3.3 – Construction des mots du code de Tardos

	p_1	p_2	p_3	$\cdots$	p_m
X_1	1	0	1	...	0
X_2	0	1	0	...	1
X_3	1	1	0	...	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
X_n	0	0	0	...	1

$\mathbb{P}(X_{ji} = 1) = p_i$

- **Accusation :** On extrait la séquence $\mathbf{Y}$ de la copie pirate. Afin de savoir si l'utilisateur j est impliqué dans la production de la contrefaçon, on calcule un score d'accusation S_j. Si ce score est supérieur à un certain seuil Z, alors on considère l'utilisateur j comme coupable. Une variante est d'accuser l'utilisateur le plus probablement coupable, *ie.* celui qui a le plus gros score. Le calcul des scores repose sur deux fonctions d'accusation, qui évaluent la corrélation entre la séquence $\mathbf{X}_j$, associée à l'utilisateur j, et la séquence extraite $\mathbf{Y}$:

$$S_j = \sum_{i=1}^{m} y_i U(X_{ji}, p_i), \tag{3.4}$$

Nous notons $g_{Y_i X_i}(p_i)$ les fonctions U_{ji}.

$$U_{ji} = g_{10}(p_i) = -\sqrt{\frac{p_i}{1-p_i}} \; si \; X_{ji} = 0 \tag{3.5}$$

$$U_{ji} = g_{11}(p_i) = \sqrt{\frac{1-p_i}{p_i}} \; si \; X_{ji} = 1 \tag{3.6}$$

L'utilisateur j est accusé si $S_j > Z$, avec $Z = 20ck = 20c(ln(1/\epsilon_1))$. Les valeurs des différents paramètres ont été discutées dans différents articles. Une particularité du code de Tardos est que la distribution des scores (moyenne et variance) est la même quelle que soit la stratégie utilisée par les *colluders* pour fabriquer la copie pirate.

La fonction de distribution est donnée sous la forme suivante :
$f(p) : [t, 1-t] \rightarrow \mathbb{R}^+$

$$f(p) = \frac{1}{2arcsin(1-2t)} \frac{1}{\pi\sqrt{p(1-p)}} \tag{3.7}$$

dans l'article de Skoric. Cette fonction est symétrique par rapport à 0.5 et de telle sorte qu'il y a beaucoup de valeurs proches de 0 et de 1. On peut voir la fonction f sur la figure 3.4.

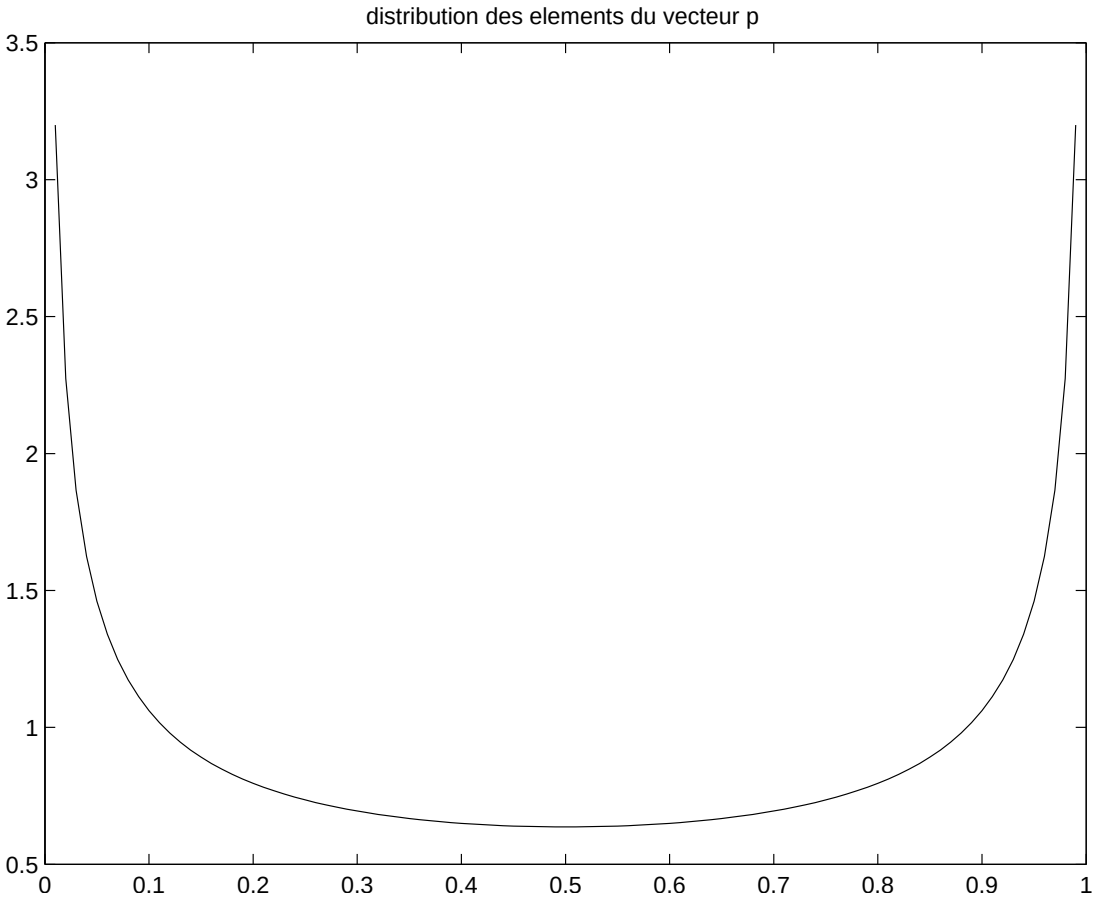


FIGURE 3.4 – Tracé de la fonction f représentant la distribution des probabilités

3.3.2 Amélioration de Skoric *et al*

L'article de Skoric est fondamental pour la suite de l'utilisation des codes de Tardos. Dans cet article, les auteurs proposent une amélioration dans le calcul des scores qui permet de réduire la taille des mots utilisés. Alors que Tardos n'utilisait dans le protocole d'accusation que les positions auxquelles Y_i valait 1, Skoric propose d'utiliser toutes les positions et ajoute donc deux fonctions d'accusations lorsque $Y_i = 0$:

$$U_{ji} = g_{00}(p_i) = \sqrt{\frac{p_i}{1-p_i}} \; si \; X_{ji} = 0 \tag{3.8}$$

$$U_{ji} = g_{01}(p_i) = -\sqrt{\frac{1-p_i}{p_i}} \; si \; X_{ji} = 1 \tag{3.9}$$

Y	1	1	0	...	1
X_j	1	0	1	...	0
	$g_{11}(p_1)$	$g_{10}(p_2)$	$g_{01}(p_3)$	...	$g_{10}(p_m)$

FIGURE 3.5 – Accusation

Score : $S_j = \Sigma_{i=1}^{m} g_{Y_i X_{ji}}(p_i)$
Les scores suivent des distributions quasi-gaussiennes, voir figure 3.6, les scores des innocents sont centrés en 0, les scores de *colluders* sont beaucoup plus élevés, centrés en $2m/c\pi$.

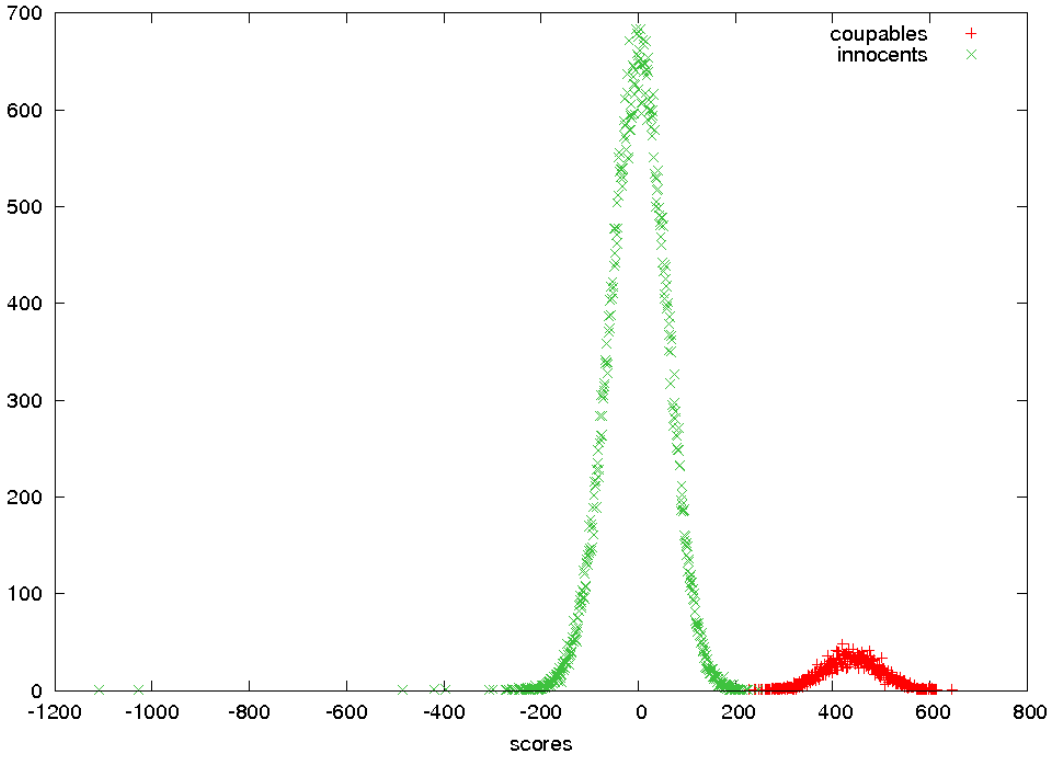


FIGURE 3.6 – Histogramme représentant la distribution des scores

Quelques commentaires pour comprendre le code de Tardos :
– Le code est mi-dense mi-creux. Il est dense pour les indices i tels que $p_i \approx 0.5$ au sens où ces colonnes de $\mathbf{X}$ comportent en espérance autant de symboles '0' que de

'1'. Il est creux pour les indices i tels que p_i est proche de 0 (ou 1) au sens où ces colonnes de $\mathbf{X}$ comportent presque que des symboles '0' (respectivement, presque que des '1'). Pour une densité de code donnée, il existe une attaque au pire cas. Les valeurs de $\mathbf{p}$ étant secrètes, les *colluders* ne peuvent adapter leur attaque à la densité du code. Ils n'ont donc qu'une stratégie globale alors que le code possède une pluralité de densités.

– Pour un utilisateur donné, un test d'hypothèse est réalisé. Le score d'accusation est comparable à un faisceau de preuves, type Cluedo. Avoir le même symbole que celui retrouvé dans la copie pirate (*i.e.* $Y_i = X_{ji}$) n'est pas une preuve irréfutable de la culpabilité de l'utilisateur, mais cela tend à corroborer cette hypothèse (comme le fait de s'être trouvé dans le boudoir alors que c'est là qu'a été trouvé le corps du colonel Moutarde) : la somme d'accusation est augmentée d'un poids positif $g_{Y_i X_{ji}}(p_i)$. A l'inverse, un symbole différent n'est pas une preuve irréfutable de son innocence, mais cela tend à l'innocenter, d'où un poids négatif (on n'a pas touché au chandelier). L'amplitude de l'accusation dépend de la rareté de correspondance entre la valeur de la copie pirate et celle de l'utilisateur. On sent bien par exemple que si $Y_i = 0$ et p_i est proche de 1, alors très peu d'utilisateurs ont un symbole '0' à cet indice : par conséquent, le poids sera positif et de forte amplitude pour ces derniers.

L'exemple 3.3.2 montre le processus complet. On y montre la matrice correspondant aux identifiants et un exemple de collusion. Le calcul des scores permet de voir que l'accusation fonctionne très bien en ce qui concerne les utilisateurs 3 et 1 (scores de 10.9 et 7.9 respectivement) qui sont effectivement colluders. On voit que l'utilisateur 2, pourtant colluder, échappe à l'accusation. Le troisième plus grand score est celui d'un innocent (4.1 pour l'utilisateur 7), et on remarque qu'il est assez éloigné du score le plus élevé.

Dans cette thèse, nous étudions les codes de Tardos avec l'amélioration symétrique de Skoric.

3.3.3 Discussion sur les paramètres constants

Les 3 paramètres numériques Z, m et t ont fait l'objet d'un article complet de Blayer et al [BT08]. Dans cet article, les auteurs observent en détail les conditions que doit remplir le code et les traduisent en contraintes sur ces paramètres. Cela leur permet de réduire considérablement la longueur du code, ce qui est très important pour des codes anti-collusion qui doivent être insérés dans des documents multimédia.

– $m = d_m c^2 k$
– $Z = d_z ck$
– $t = 1/d_t c$

Le code de Tardos doit permettre de retrouver les *colluders* sans accuser un innocent quelle que soit la stratégie utilisée par les *colluders* pour créer le faux.

Cette étude a donné les résultats suivants :

– d_t peut être de l'ordre de 50 (valeur donnée par Tardos $d_t = 300$)
– $d_z = 2\pi$, alors que la valeur donnée par Tardos était $d_z = 20$

Exemple 11. Exemple de vecteur de probabilité pour $m = 12$

```
0.96 0.84 0.48 0.41 0.43 0.25 0.51 0.51 0.88 0.86 0.69 0.34
```

Exemple de matrice correspondant à un code de Tardos pour une longueur $m = 12$ et $n = 10$ utilisateurs. Le vecteur de probabilité est indiqué au-dessus de la matrice.

```
0.96 0.84  0.48  0.41  0.43  0.25  0.51  0.51  0.88  0.86  0.69  0.34
1     1     1     0     0     0     1     0     1     0     1     0
1     1     0     0     0     0     1     0     1     1     1     1
0     1     0     1     0     0     0     0     0     1     1     1
1     1     0     0     1     0     1     1     1     1     1     0
1     0     1     0     0     0     1     1     1     1     0     0
1     1     0     0     0     1     1     0     1     0     1     1
1     1     1     1     0     0     0     1     1     1     1     0
1     1     1     0     1     0     0     0     1     1     0     1
1     1     0     1     0     0     0     1     1     1     1     1
1     0     1     0     1     0     1     0     1     1     1     0
```

La première ligne correspond à l'identifiant du premier utilisateur :

```
1     1     1     0     0     0     1     0     1     0     1     0
```

Séquence produite par une collusion des trois premiers utilisateurs avec une stratégie Minorité.

```
1     1     1     0     0     0     1     0     1     0     1     0
1     1     0     0     0     0     1     0     1     1     1     1
0     1     0     1     0     0     0     0     0     1     1     1

0     1     1     1     0     0     0     0     0     0     1     0
```

Scores

```
 7.9424 2.8145 10.9035 -0.8395 -1.7079 -4.6573 4.1383 2.9499 3.0522 -5.7026
```

– $d_m = 2\pi^2$, ce qui est beaucoup moins qu'à l'origine, $d_m = 100$

Le tableau 3.3 donne des exemples de paramètres pour les codes de Tardos.

utilisateurs n	colluders $1 << c$	fausse alarme $\varepsilon_1 (<< \varepsilon_2)$	longueur $m = 2\pi^2 c^2 \lceil \ln(1/\varepsilon_1) \rceil$	seuil $Z = 20c\lceil \ln(1/\varepsilon_1) \rceil$
10^2	5	10^{-3}	3455	220
10^3	5	10^{-4}	4935	315
	5	10^{-7}	8390	535
	7	10^{-4}	9673	440
	7	10^{-7}	16443	748
	10	10^{-4}	19740	629
	10	10^{-7}	33557	1069
	20	10^{-4}	78957	1257
10^4	5	10^{-5}	5922	377
	7	10^{-5}	11687	528
	10	10^{-5}	23688	754
	20	10^{-5}	94749	1508
10^5	5	10^{-6}	6909	440
	7	10^{-6}	13542	616
	10	10^{-6}	27635	880
	20	10^{-6}	110540	1760
10^6	5	10^{-7}	8390	535
	7	10^{-7}	16443	748
	10	10^{-7}	33557	1069
	20	10^{-7}	134227	2137

TABLE 3.3 – Exemples de codes de Tardos/Skoric pour différents nombres d'utilisateurs n, différents nombres de *colluders* c, et différentes probabilité de fausse accusation ϵ_1.

3.3.4 Etude sur les fonctions d'accusation

Parmi les nombreux travaux faits sur les codes de Tardos, nous faisons une description plus précise des travaux de Furon et al [FGC08b]. Dans cet article, les auteurs s'interrogent sur le choix des fonctions d'accusation.

Furon et al se sont donc demandé si les fonctions 3.8, 3.9, 3.5 et 3.6 étaient les meilleures qu'on puisse utiliser pour cette construction.

Tout d'abord, il faut savoir quelles sont les contraintes que l'on veut remplir.

– Les scores des utilisateurs sont décorrélés (Covariance = 0)

– La distribution des scores est la même quelle que soit la stratégie des *colluders*.

Connaissant ces deux contraintes, on calcule ensuite les espérances et variances des scores des innocents et des *colluders*. Ces éléments ne doivent pas être fonctions de la stratégie. Pour faire ces calculs, les auteurs ont introduit la variable $q_i = \mathbb{P}(Y_i = 1)$. q_i est la probabilité que la copie pirate contienne un 1 à la position i. Les espérances et

variances doivent donc être des fonctions qui ne dépendent pas de q. On obtient alors des relations entre les différentes fonctions d'accusation :

$$pg_{10}(1-p) = -(1-p)g_{11}(1-p)$$

$$(1-p)g_{11}(1-p) = pg_{11}(p)$$

Ensuite, la maximisation de l'espérance des scores des *colluders* conduit au résultat de Tardos concernant la fonction de répartition f ainsi que la fonction d'accusation g_{10}

$$f(p) = \frac{1}{\pi\sqrt{p(1-p)}},\ g_{10}(p) = \sqrt{\frac{1-p}{p}}$$

3.4 Conclusion

Les codes anti-collusion sont un des outils principaux des schémas de *fingerprinting*. Les résultats des nombreuses recherches conduites sur le sujet sont des codes de plus en plus performants et suffisamment courts pour permettre une mise en pratique, c'est-à-dire l'insertion dans un document numérique.

Ce sont les codes correcteurs d'erreurs qui ont été étudiés en premier, en particulier les codes de Reed-Solomon. Le schéma de code concaténé proposé par Boneh-Shaw a été très étudié, de multiples combinaisons code interne code externe ont été testées, là encore en utilisant très souvent des codes de Reed-Solomon. Le problème de ces codes est qu'ils ne peuvent être utilisés en pratique à cause de leur longueur qui est beaucoup trop grande. Les codes de Tardos, proposés récemment, atteignent la borne de longueur minimale démontrée par Peikert : $m = O(c^2 log(1/\epsilon_1))$ [PSS03], ce qui les rend très intéressants. Ils sont de plus faciles à calculer et à manipuler. Cependant, un processus complet de la phase d'accusation des codes de Tardos implique de calculer les scores de tous les utilisateurs, ce qui donne une complexité de la phase d'accusation en $O(n)$. Lors de l'utilisation de codes correcteurs d'erreur, l'accusation consiste en un décodage, qui dépend alors de la longueur du code ($O(m^3)$ pour un code de Reed-Solomon). On voit que cette accusation par parcours exhaustif des scores est une particularité du code de Tardos, qui peut être un avantage (si un utilisateur semble suspect, on peut calculer son score sans considérer les autres utilisateurs), ou une limitation (il faut calculer tous les scores pour conduire une accusation complète).

Chapitre 4

Décodage EM du code de Tardos

Nous présentons dans ce chapitre une amélioration de la phase d'accusation des codes de Tardos binaires. Plus spécifiquement nous montrons comment l'optimiser en fonction de la stratégie d'attaque des pirates. Nous proposons également des moyens pour estimer à partir d'une copie le nombre d'attaquants qui se sont rassemblés pour la créer, ainsi que la stratégie qu'ils ont employée. Notre solution s'appuie sur un algorithme itératif *a la* EM, dans lequel une meilleure estimation de la stratégie permet une meilleure détection des attaquants, qui permet à son tour une meilleure estimation de la stratégie, etc.

T. Furon *et al.* [FGC08b] ont démontré que les fonctions d'accusation sont optimales dans un contexte général, mais aussi qu'il existe des fonctions plus efficaces si le décodeur a des informations sur l'attaque qui a été réalisée. Ici, nous continuons dans cette direction en proposant des mécanismes d'estimation dynamique de la stratégie des pirates ainsi que de leur nombre, et des fonctions d'accusation optimisées pour cette stratégie.

Ces travaux ont été publiés lors de la conférence SPIE [CXFF09], ainsi que dans la revue Traitement du Signal dans une version plus complète [CFF10].

4.1 Contexte

4.1.1 Comment choisir la fonction f et les fonctions d'accusation de façon adéquate ?

Dans l'article original [Tar03], Tardos donne les trois fonctions f, g_{10} et g_{11} suivantes données en 3.5 et 3.6 (g_{00} et g_{01} étaient initialement égales à 0) et prouve qu'avec ce choix, les performances de l'accusation ne dépendent pas de la stratégie des *colluders*.

Mais il n'a pas expliqué comment il avait trouvé ces fonctions. Ces choix ont été conservés par Skoric *et al.*, qui ont proposé une accusation 'symétrique', donnant à g_{00} et g_{01} des valeurs non nulles [SKC08] données en 3.8 et 3.9

Une relecture statisticienne apporte une nouvelle interprétation. La stratégie de la collusion n'est pas connue à l'accusation : c'est un paramètre de nuisance. L'idée de G. Tardos est de créer un test basé sur une quantité pivot indépendante du paramètre de nuisance. Les fonctions choisies assurent cette indépendance uniquement pour les

moments d'ordre 1 et 2 :

$$\text{Innocent :} \quad E[S_j] = 0, \qquad E[S_j^2] = m \tag{4.1}$$

$$\text{Colluder :} \quad E[S_j] = \tfrac{2m\pi}{c}, \qquad E[S_j^2] = m \tag{4.2}$$

Grâce à la borne de Markov-Chebyshev, les probabilités d'accuser à tort et de rater un coupable sont majorées et inférieures aux niveaux donnés dans le cahier des charges pour une longueur de code $m = O(c^2 \log(n/\epsilon))$ [1].

Plus récemment, Furon *et al.*[FGC08b] ont donné des améliorations si on relâche certaines hypothèses : connaître la taille maximum de la collusion c_{max} lors de la génération du code aide à déterminer une meilleure fonction f, quelle que soit la stratégie ; connaître la stratégie de la collusion aide à déterminer de meilleures fonctions lors de l'accusation. Lorsque les fonctions d'accusation correspondent à la stratégie de collusion, l'espérance des scores des *colluders* est supérieure à l'espérance pour le score de Tardos. Mais en revanche, lorsque qu'il y a une mauvaise correspondance, les effets peuvent être désastreux.

Une alternative à l'accusation très conservatrice de G. Tardos est ouverte. Cependant, l'hypothèse admise dans [FGC08b] sur la stratégie de collusion est critiquable : les auteurs ont optimisé les fonctions d'accusation en fonction de la stratégie de collusion mais ont conservé une contrainte (Eq. (14) de [FGC08b] [qui peut être réécrite avec nos notations comme $(1-p)g_{00}(p) = pg_{11}(p)$)] liée à l'indépendance entre la variance des scores des innocents et la stratégie des *colluders*. Ici, nous relâchons cette hypothèse, en ne conservant que le minimum de contraintes possible sur les fonctions d'accusation et en les optimisant en utilisant une estimation de la stratégie des *colluders*. Pour mesurer les performances de notre solution, nous la comparons avec la version symétrique des codes de Tardos faite par Skoric *et al.* [SKC08] ainsi qu'avec l'optimisation de Furon *et al.* [FGC08b].

Lors des itérations de notre algorithme, les distributions vont se 'séparer', comme illustré sur la figure 4.1. La courbe en trait noir épais représente les scores des innocents, la courbe en pointillé représente les scores obtenus avec les fonctions de Tardos. En rouge apparaissent les distributions telles que nous souhaitons les obtenir grâce aux nouvelles fonctions d'accusation.

4.1.2 Comment réduire la taille du code ?

L'une des propriétés les plus intéressantes des codes de Tardos est sa longueur minimale. Cette longueur est déterminée par la taille maximum de la collusion et les probabilités d'erreur (d'accuser un innocent, et de ne pas trouver de *colluder*). Malgré cela, dans un schéma réel on est amené à utiliser des codes plus courts, en ce sens où la contrainte vient de la taille du contenu à protéger et du taux d'insertion de la technique de marquage choisie. Dans ce cas, il est intéressant d'utiliser les codes de Tardos pour

1. Nous avons omis de nombreux détails, la démonstration est en fait bien plus technique (cf. Blayer [BT08]).

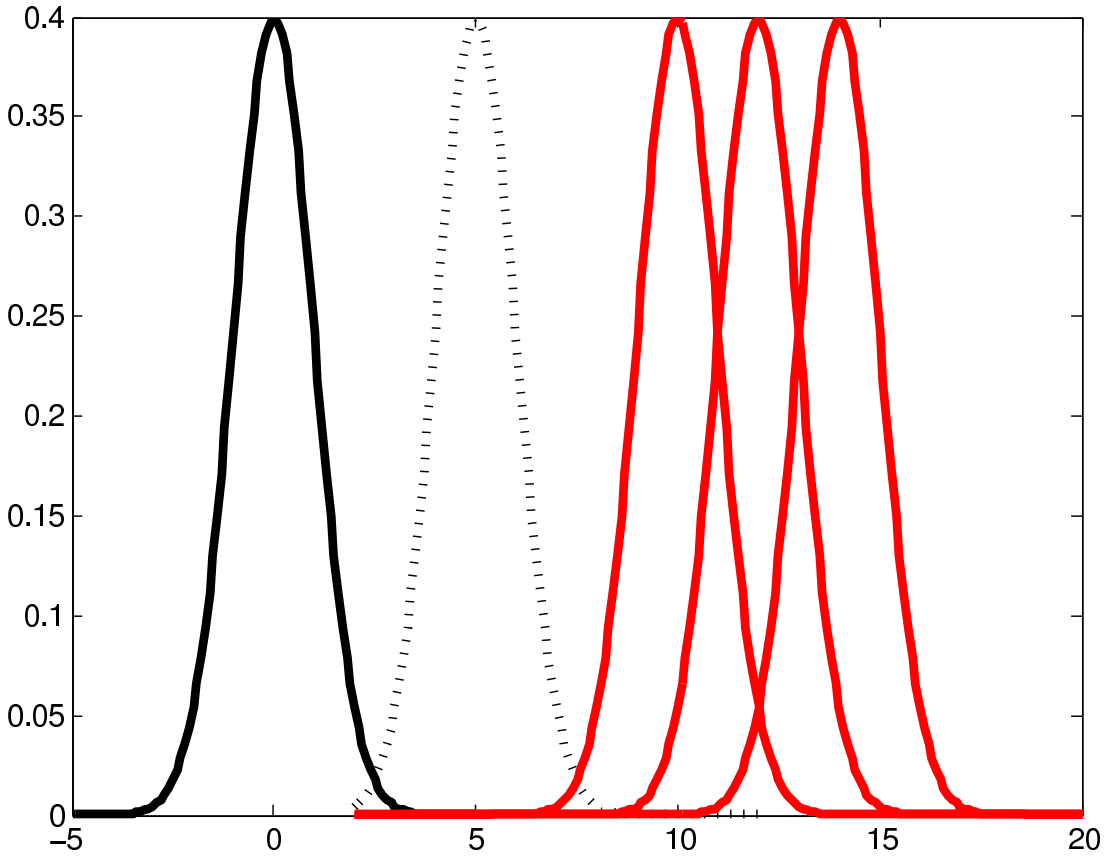


FIGURE 4.1 – Illustration de la séparation des distributions des scores par application de l'algorithme EM. En abscisse les valeurs des scores, les valeurs de l'axe des ordonnées ne sont pas significatives.

leur implémentation facile, et d'adapter le processus d'accusation pour gérer la longueur et conserver une bonne efficacité de traçage.

Nous proposons ici d'utiliser un algorithme d'accusation itératif *a la* EM (Expectation-Maximization) avec afin d'obtenir une meilleure accusation même lorsque la longueur du code est plus courte que celle préconisée par G. Tardos.

4.1.3 Comment estimer la pertinence de l'accusation ?

Dans le processus original, chaque utilisateur j est testé indépendamment et on n'a donc pas besoin de calculer tous les scores pour rendre un verdict. Cet utilisateur est considéré comme un *colluder* si son score est supérieur au seuil Z. Le seuil est choisi pour garantir une certaine probabilité de fausse alarme. Ainsi, lorsqu'on accuse un utilisateur on a une borne supérieure de la probabilité d'erreur, mais on n'a pas d'estimation précise de cette erreur. On adopte ici une autre stratégie, en calculant les scores de tous les utilisateurs et en accusant celui qui a le plus grand. On peut noter que bien que cet

utilisateur soit celui qui est le plus sûrement coupable, il n'y a pas de garantie de ne pas accuser un innocent. On peut ou bien comparer ce score à un seuil afin de refuser de porter une accusation lorsque même le score maximal est trop faible, ou bien estimer expérimentalement la probabilité d'accuser à tort [FGC08a].

4.2 Une estimation itérative de la stratégie

En suivant les pas de T. Furon *et al.* [FGC08b], notre objectif est d'optimiser les fonctions d'accusation en fonction de la stratégie des *colluders*. Nous procédons en deux étapes : d'abord nous estimons la stratégie, puis nous optimisons les fonctions d'accusation. Ce procédé est itéré, chaque itération tirant profit d'une nouvelle estimation de l'ensemble des *colluders via* Expectation-Maximization (EM). Nous avons remarqué que les résultats évoluent peu après 2 itérations. L'algorithme est donc forcé à s'arrêter au bout de 2 itérations.

On modélise la stratégie de la collusion par l'ensemble des probabilités $\{\mathbb{P}(Y_i = 1|\Sigma_i = \sigma_i), \sigma_i = 0..c\}_{i=1..m}$; la variable aléatoire $\Sigma_i = \sum_{j\in\mathcal{C}} X_{ji}$ correspond au nombre de mots des *colluders* ayant un '1' à la position i. On admet que la même stratégie a été utilisée pour chaque position $1 \leq i \leq m$. Afin d'alléger les formules, nous omettrons l'indice i qui indique la position considérée et appellerons $\boldsymbol{\theta}$ le modèle de collusion : $\boldsymbol{\theta} = \{\mathbb{P}(Y = 1|\Sigma = \sigma), \sigma = 0..c\}$. Nous décrivons maintenant les étapes du processus d'estimation itérative en détail, en utilisant les notations allégées. Le processus est illustré par la 4.2.

S1. Initialisation : nous calculons tous les scores d'accusation avec les fonctions d'accusation de B. Skoric *et al* : $g_{11}^{(S)}(p) = g_{00}^{(S)}(1-p) = -g_{01}^{(S)}(p) = -g_{10}^{(S)}(1-p) = \sqrt{\frac{1-p}{p}}$. Ces scores sont placés dans le vecteur $\mathbf{S}$.

S2. Décodage EM : la séquence $\mathbf{S}$ contient un mélange de scores d'innocents et de *colluders*. Un algorithme EM classique estime le statut, « innocent » ou « colluder », de chaque utilisateur. EM prend en entrée le vecteur $\mathbf{S}$ et les moyennes et variances théoriques des scores des innocents et *colluders*. En sortie, on a le vecteur $\hat{\mathbf{T}}$; $\hat{T}_j$ correspond à l'estimation de la probabilité que le score S_j de l'utilisateur j soit celui d'un *colluder*. Un exemple classique de l'algorithme EM est le mélange de gaussiennes où il s'agit de séparer les échantillons de plusieurs (deux dans notre cas) distributions dont on sait qu'elles sont gaussiennes mais de variances et espérances à déterminer [Del10, Sec. II.7]. Nous l'appliquons pour séparer les distributions des scores des innocents, d'une part et les scores des *colluders*, d'autre part. Nous avons choisi d'initialiser l'algorithme avec des données pertinentes, ces paramètres d'espérance et de variance sont donc remplis avec les valeurs données théoriquement pour les scores de Tardos. Le nombre de *colluders* c est initialisé à 2.

S3. Avec $\hat{\mathbf{T}}$ et $\mathbf{S}$, on estime la taille de la collusion, notée $\hat{c}$, ainsi que sa stratégie, notée $\hat{\boldsymbol{\theta}}$.

S4. En considérant $\hat{\boldsymbol{\theta}}$ on optimise les fonctions d'accusation $g_{00}(p, \hat{\boldsymbol{\theta}})$, $g_{11}(p, \hat{\boldsymbol{\theta}})$, $g_{10}(p, \hat{\boldsymbol{\theta}})$, et $g_{01}(p, \hat{\boldsymbol{\theta}})$.

S5. On calcule les nouveaux scores, conservés dans le vecteur **S**. On revient alors à l'étape S2 pour itérer.

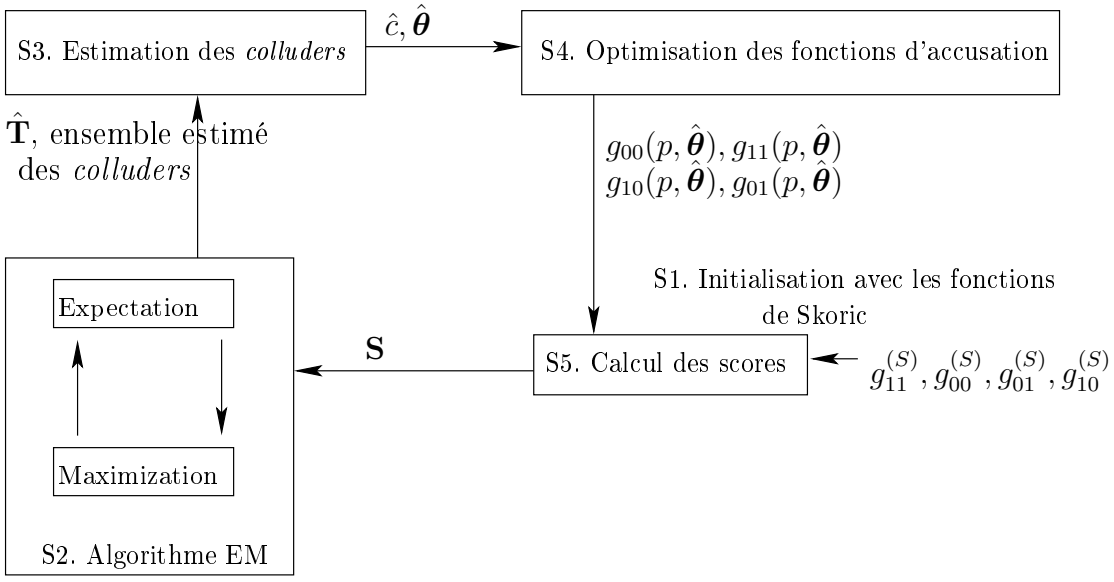


FIGURE 4.2 – Calcul des nouvelles fonctions d'accusation

Les fonctions d'accusation optimisées de [FGC08b] sont obtenues sous l'hypothèse que les scores des innocents ne sont pas corrélés entre eux. De fait, pour une valeur de m assez grande, les scores sont considérés comme suivant des distributions gaussiennes (somme de variables aléatoires i.i.d.). C'est pourquoi les scores des innocents seront supposés indépendants. C'est une condition nécessaire pour appliquer l'algorithme EM à l'étape 2. Cette affirmation est fausse pour les scores des *colluders*, mais nos résultats expérimentaux montrent que cela n'empêche pas la convergence de cet algorithme pour un petit nombre de *colluders* $c \ll n$.

Nous allons maintenant détailler les étapes clés de notre algorithme.

4.2.1 Estimation de la stratégie : étape S3

On considère les scores **S** et les probabilités associées dans le vecteur $\hat{\mathbf{T}}$; $\hat{T}_j = 1$ signifie que l'utilisateur j est un *colluder*, $\hat{T}_j = 0$ signifie qu'il est innocent. Dans la pratique, et les $\hat{T}_j$ que nous obtenons ne sont pas binaires, ils sont une probabilité que l'utilisateur j ait participé à la création de la copie pirate. Nous estimons la taille de la collusion par $\hat{c} = \lceil \Sigma_{j=0}^{n} \hat{T}_j \rceil$. On considère ensuite les $\hat{c}$ utilisateurs correspondant aux plus grandes probabilités $\hat{T}_j$, et on utilise leurs séquences pour calculer le modèle de collusion $\hat{\boldsymbol{\theta}}$. Pour chaque position i, on calcule σ_i comme somme des X_{ji} pour chaque utilisateur j dans l'ensemble estimé des *colluders*. Pour chaque valeur possible de $0 \leq \sigma_i \leq c$, on calcule la moyenne des éléments de $\mathbf{Y}$ correspondants.

4.2.2 Optimisation de l'accusation : étape S4

Les nouvelles fonctions d'accusation sont obtenues par une optimisation sous contraintes, pour une collusion estimée donnée $\hat{\boldsymbol{\theta}}$. On note μ_{Inn} et ν_{Inn} (resp. μ_{Coll} et ν_{Coll}) l'espérance et la variance de la distribution des scores des utilisateurs innocents (resp. *colluders*), et $\kappa(S_j, S_k)$ la covariance entre les scores des utilisateurs j et k. De par la construction du code, les symboles sont i.i.d. d'un index à l'autre. Ceci implique, avec la formule de calcul des scores3.4, que les statistiques des scores sont linéaires en m :

$$\mu_{Inn} = m\tilde{\mu}_{Inn}, \qquad \nu_{Inn} = m\tilde{\nu}_{Inn}, \tag{4.3}$$

$$\mu_{Coll} = m\tilde{\mu}_{Coll}, \qquad \nu_{Coll} = m\tilde{\nu}_{Coll}. \tag{4.4}$$

On résume les contraintes principales :
- Les scores des innocents sont centrés : $\tilde{\mu}_{Inn} = 0$,
- Les scores des innocents sont normalisés : $\tilde{\nu}_{Inn} = 1$,
- deux utilisateurs innocents ont des scores indépendants, ce qui se traduit sous les affirmations gaussiennes, par $\kappa(S_j, S_k) = 0$.

Ce sont les mêmes contraintes que dans T. Furon *et al* [FGC08b], excepté qu'ici on ne contraint pas la variance des scores des *colluders*.

La distance de Kullback-Leibler mesure la « distance » entre les distributions des scores des *colluders* et des innocents. La théorie de la détection nous dit qu'elle doit être la plus grande possible afin d'assurer une bonne séparation entre les innocents et les *colluders*, ce qui donne des verdicts fiables. Comme nous avons déjà considéré que les scores des *colluders* suivent une distribution normale $\mathcal{N}_{Coll}$, et que les scores des utilisateurs innocents une distribution normale $\mathcal{N}_{Inn}$, la distance de Kullback-Leibler entre deux distributions normales satisfaisant $\tilde{\mu}_{Inn} = 0$ et $\tilde{\nu}_{Inn} = 1$ est la suivante :

$$D_{KL}\left(\mathcal{N}_{Coll}, \mathcal{N}_{Inn}\right) = \frac{1}{2}\left(m\tilde{\mu}_{Coll}^2 - \log(\tilde{\nu}_{Coll}) + \tilde{\nu}_{Coll} - 1\right). \tag{4.5}$$

Comme m est très grand, le terme prépondérant de la somme est $m\tilde{\mu}_{Coll}^2$. Notre objectif est de maximiser $\tilde{\mu}_{Coll}$ sous les contraintes $\mu_{Inn} = 0$, $\kappa(S_j, S_k) = 0$, et $\tilde{\nu}_{Inn} = 1$.

Considérant ces conditions, les fonctions qui maximisent $\tilde{\mu}_{Coll}$ sont

$$g_{11}(p, \boldsymbol{\theta}) = \frac{1}{2\lambda}\frac{1-p}{q(p,\boldsymbol{\theta})}A(p,\boldsymbol{\theta}), \qquad g_{00}(p,\boldsymbol{\theta}) = \frac{1}{2\lambda}\frac{p}{1-q(p,\boldsymbol{\theta})}A(p,\boldsymbol{\theta}),$$

$$g_{10}(p,\boldsymbol{\theta}) = -\frac{p}{1-p}g_{11}(p,\boldsymbol{\theta}), \qquad g_{01}(p,\boldsymbol{\theta}) = -\frac{1-p}{p}g_{00}(p,\boldsymbol{\theta}) \tag{4.6}$$

avec

$$\lambda = \frac{1}{2}\sqrt{\mathbb{E}_p\left[A^2(p,\boldsymbol{\theta})\frac{p}{q(p,\boldsymbol{\theta})}\frac{1-p}{1-q(p,\boldsymbol{\theta})}\right]}, \tag{4.7}$$

$$q(p,\boldsymbol{\theta}) = \mathbb{P}(Y=1|P=p,\boldsymbol{\theta}), \tag{4.8}$$

$$A(p,\boldsymbol{\theta}) = \mathbb{P}(Y=1|X=1,P=p,\boldsymbol{\theta}) - \mathbb{P}(Y=1|X=0,P=p,\boldsymbol{\theta}). \tag{4.9}$$

Les résultats nous permettent de calculer l'expression du μ_{Coll} maximisé :

$$\tilde{\mu}_{Coll} = \sqrt{\mathbb{E}_p\left[A^2(p,\boldsymbol{\theta})\frac{p}{q(p,\boldsymbol{\theta})}\frac{1-p}{1-q(p,\boldsymbol{\theta})}\right]}. \tag{4.10}$$

Preuve :
On maximise cette expression en utilisant un Lagrangien $J(g_{11}, g_{00}) = \tilde{\mu}_{Coll} - \lambda(\tilde{\nu}_{Inn} - 1)$. voir les détails en section 4.3. □

4.2.3 Résultats théoriques

Lors des expériences, nous considérons différentes stratégies utilisées par les *colluders* pour créer $\mathbf{Y}$:

Uniforme les *colluders* choisissent au hasard un symbole parmi leurs copies :

$$\mathbb{P}(Y = 1|\Sigma = \sigma) = \sigma/c;$$

Majorité les *colluders* choisissent le symbole le plus fréquent :

$$\mathbb{P}(Y = 1|\Sigma = \sigma) = 1 \text{ si } \sigma > c/2, \quad 0 \text{ sinon};$$

Minorité les *colluders* choisissent le symbole le moins fréquent :

$$\mathbb{P}(Y = 1|\Sigma = \sigma) = 0 \text{ si } \sigma > c/2, \quad 1 \text{ sinon};$$

$\mathbb{P}(Y = 1|\Sigma = 0) = 0$, $\mathbb{P}(Y = 1|\Sigma = c) = 1$ (marking assumption)

All1 si ils ont au moins un '1', les *colluders* mettent un '1' :

$$\mathbb{P}(Y = 1|\Sigma = \sigma) = 1 \text{ si } \sigma \neq 0;$$

$\mathbb{P}(Y = 1|\Sigma = 0) = 0$, $\mathbb{P}(Y = 1|\Sigma = c) = 1$ (marking assumption)

All0 si ils au moins un '0', les *colluders* mettent un '0' :

$$\mathbb{P}(Y = 1|\Sigma = \sigma) = 0 \text{ si } \sigma \neq c.$$

$\mathbb{P}(Y = 1|\Sigma = 0) = 0$, $\mathbb{P}(Y = 1|\Sigma = c) = 1$ (marking assumption)

Ces stratégies sont conformes à la « marking assumption » car on a toujours $\mathbb{P}(Y = 1|\Sigma = 0) = 0$ et $\mathbb{P}(Y = 1|\Sigma = c) = 1$. Pour les stratégies de majorité et minorité, plusieurs comportements sont envisageables au cas où les colluders possèdent le même nombre de '0' et de '1'. Nous avons choisi de mettre un '1', mais il est possible de mettre un '0', ou bien de refaire un tirage aléatoire ($\mathbb{P}(Y = 1|\Sigma = c/2) = 0.5$). Il existe dans la littérature de nombreuses autres stratégies, notamment la stratégie dite *coin flip* qui consiste à faire un tirage aléatoire pour toutes les positions : $\mathbb{P}(Y = 1) = 0.5$ quel que soit σ (en respectant la marking assumption). Nous avons fait une sélection parmi toutes ces stratégies.

On calcule le ratio $cm\tilde{\mu}_{Coll}/\sqrt{\tilde{\nu}_{Inn}}$ obtenu avec les fonctions optimisées du théorème 4.2.2, et on les compare dans le tableau 4.1 aux précédents résultats de T. Furon *et al* [FGC08b], pour lesquels $\tilde{\nu}_{Coll}$ était contrainte à être égale à 1. On voit que nos fonctions d'accusation sont, comme espéré, plus efficaces pour la stratégie des *colluders* pour laquelle elles ont été calculées. On voit aussi qu'elles sont plus efficaces que celles de [FGC08b] dans tous les cas, y compris lorsque les stratégies ne coïncident pas.

TABLE 4.1 – Valeurs de $cm\tilde{\mu}_{Coll}/\sqrt{\tilde{\nu}_{Inn}}$ obtenues après optimisation des fonctions d'accusation pour $m = 100$, $c = 3, 4, 5$. Entre parenthèses sont données celles obtenues par T. Furon *et al* . Rappelons qu'avec les fonctions de B. Skoric *et al* on a la valeur 64 dans tous les cas.

		Stratégie des *colluders*				
c	accusation	Uniforme	Majorité	Minorité	All1	All0
3	Uniforme	**98 (71)**	106 (80)	100 (53)	97 (66)	97 (66)
	Majorité	96 (67)	**110 (84)**	100 (34)	95 (59)	95 (59)
	Minorité	81 (50)	59 (38)	**112 (75)**	89 (56)	89 (56)
	All1	83 (69)	88 (73)	88 (62)	**114 (68)**	84 (68)
	All0	83 (69)	88 (73)	88 (62)	84 (68)	**114 (68)**
4	Uniforme	**98 (71)**	106 (80)	105 (44)	99 (62)	99 (62)
	Majorité	96 (67)	**110 (84)**	105 (17)	97 (50)	97 (50)
	Minorité	61 (34)	25 (15)	**128 (91)**	88 (53)	88 (53)
	All1	79 (65)	83 (63)	88 (72)	**121 (67)**	87 (67)
	All0	79 (65)	83 (63)	88 (72)	87 (67)	**121 (67)**
5	Uniforme	**98 (71)**	110 (83)	110 (33)	100 (58)	100 (58)
	Majorité	94 (63)	**120 (93)**	113 (-22)	98 (35)	98 (35)
	Minorité	37 (19)	-20 (-17)	**155 (121)**	82 (52)	82 (52)
	All1	77 (59)	83 (47)	90 (90)	**128 (69)**	90 (69)
	All0	77 (59)	83 (47)	90 (90)	90 (69)	**128 (69)**

4.2.4 Résultats expérimentaux

Pour tous les résultats qui suivent, les vecteurs $\mathbf{p}$ ont été générés avec la fonction f donnée par Tardos. Suite aux travaux de Blayer [BT08], nous avons choisi d'augmenter la valeur du paramètre t par rapport à celle donnée par Tardos. Dans ce qui suit, on a $t = 1/100$. Les résultats se présentent sous la forme suivante : nous observons les k utilisateurs ayant les plus grands scores, et pour chacun, nous donnons la probabilité d'être un *colluder*. Sur la 4.3(a), on voit ainsi que le plus grand score correspond à celui d'un *colluder* dans 100% des cas pour toutes les stratégies observées alors que le cinquième plus grand score est celui d'un *colluder* dans seulement 60 % des cas pour la stratégie majorité, par exemple. Ces probabilités sont estimées par des simulations numériques de type Monte-Carlo.

4.2.4.1 Première expérience : test des fonctions

Nous regardons d'abord le comportement des fonctions, en supposant que l'on a correctement estimé la stratégie ainsi que le nombre de *colluders*.

Les figures 4.3(b) et 4.4(b) montrent que ces nouvelles fonctions donnent de meilleures performances. Deux remarques sont à faire : les performances varient beaucoup suivant la stratégie de la collusion. Ceci est tout à fait normal. Contrairement à Tardos qui donne une accusation la plus invariante possible à ce paramètre de nuisance, nous choisissons au contraire d'exploiter sa connaissance. Du coup, il y a des stratégies de collusion qui sont pires que d'autres. Le prix à payer pour être invariant à la stratégie de collusion est donc une perte des performances plus ou moins grande suivant la nocivité de la stratégie.

Deuxièmement, l'amélioration par rapport au décodage de Tardos semble plus grande lorsque le nombre de *colluders* croît. Les deux expériences, $c = 5$ et $c = 8$, étant réalisées pour $m = 1000$, cela montre que le code de Tardos est trop court pour lutter contre une collusion de taille $c = 8$, sauf si on exploite la connaissance de la stratégie de collusion. Autrement dit, cette idée permettrait de réduire la taille des codes pour un niveau de performances donné. En réalité, ceci est faux car rien ne dit que les *colluders* vont choisir une stratégie peu dangereuse. Si on veut garantir un niveau de performance quelle que soit la collusion, alors il ne faut retenir que la stratégie la plus nocive. Dans nos expériences, c'est la stratégie 'uniforme', et l'écart de performances par rapport au décodage de Tardos n'est pas si grand que cela. Autrement dit, le code de Tardos conserve la même longueur m, mais si les *colluders* ne sont pas les plus malicieux, ils seront plus sûrement trouvés avec notre décodage.

4.2.4.2 Seconde expérience : test de l'estimateur de stratégie

Nous supposons maintenant que le décodeur connaît c et il doit estimer la stratégie. Le fait de connaître le nombre de *colluders* permet d'améliorer la phase d'initialisation de notre schéma : nous pouvons, lors de la première itération, calculer les probabilités T_j sans algorithme EM car nous connaissons les distributions des scores : des Gaussiennes $G(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\,\pi}} \exp -\frac{(x-\mu)^2}{2\sigma^2}$ d'espérance et de variance données par (4.2) et (4.1).

Ainsi :

$$\hat{T}_j = \frac{cG(S_j; \mu_{Coll}, \sigma_{Coll})}{cG(S_j; \mu_{Coll}, \sigma_{Coll}) + (n-c)G(S_j; \mu_{Inn}, \sigma_{Inn})}.$$

On voit sur les figures 4.3(c) et 4.4(c) que cette méthode donne de moins bons résultats. L'estimation de la stratégie de collusion est imparfaite, et cela se répercute sur les performances de l'accusation. Les stratégies déterministes ('Majorité', 'Minorité', 'All1', 'All0'), au sens où y_i est une fonction déterministe de σ_i, sont plus difficiles à estimer que pour la stratégie 'uniforme'. Ceci est un moindre mal puisque nos fonctions d'accusation étaient très efficaces contre ce type de stratégie.

4.2.4.3 Troisième expérience : test de l'estimateur du nombre de *colluders*

Maintenant, le décodeur ignore *a priori* la valeur de c. C'est l'algorithme EM qui va en fournir une estimation. La précision de cette estimation apparaît comme fortement liée au quotient c/n. Nous avons choisi de forcer l'estimateur à trouver un nombre de *colluders* entre $c_{\min} = 2$ et $c_{\max} = 10$. Pour la stratégie 'uniforme', l'importance de c est nulle car les fonctions d'accusation sont les mêmes pour deux tailles de collusion différentes. En revanche, elles sont au contraire très dépendantes de c pour les stratégies déterministes.

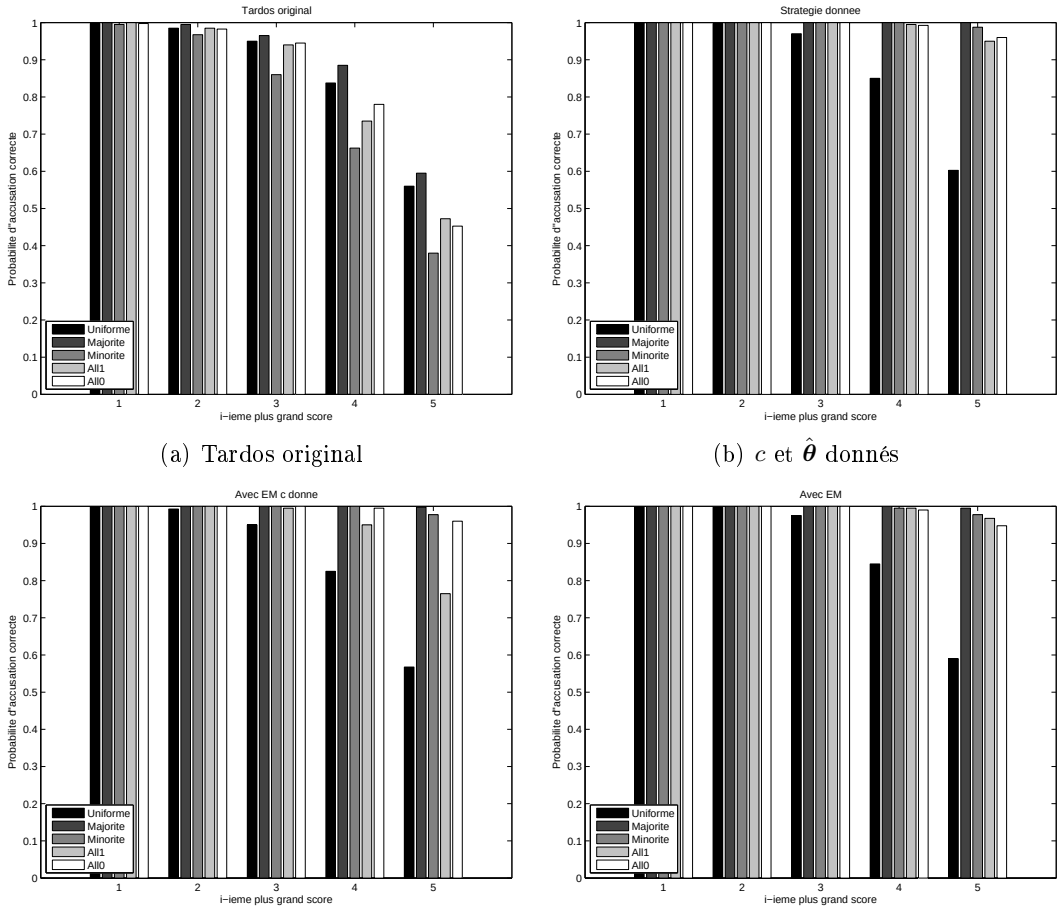
Nous obtenons parfois de meilleurs résultats que lorsque c est donné et que lorsque la stratégie est donnée. Cela vient du fait que le score est optimisé dans l'Annexe *pour une valeur c donnée*, et que si $\hat{c} \neq c$, nous ne sommes pas sûrs de remplir la contraire $\tilde{\nu}_{Inn} = 1$ ni même d'avoir maximisé $\tilde{\mu}_{Coll}$. Ces deux valeurs dépendent de la vraie stratégie de collusion $\boldsymbol{\theta}$ qui restera toujours inconnue pour le décodeur. Il ne faut surtout pas se fier à l'estimation $\hat{c}$ de la taille de la collusion fournie par l'algorithme EM pour accuser un nombre donné de personnes, comme par exemple, accuser les $\hat{c}$ plus gros scores. Il est plus sage de rester sur une attitude plus sûre qui est d'accuser uniquement le plus grand score, même si les scores sont calculés sur une hypothétique taille de collusion.

La figure 4.4(d) montre que quand la taille de la collusion est grande, l'algorithme EM joue son rôle avec précision et les performances de notre décodage sont bien meilleures que le décodage de Tardos, mais elles sont très dépendantes de la stratégie de la collusion.

4.3 Détails des calculs

Nous détaillons ici les calculs correspondant à l'étape S4 (voir 4.2.2)

On veut trouver les fonctions $g_{11}(p, \boldsymbol{\theta})$, $g_{10}(p, \boldsymbol{\theta})$, $g_{01}(p, \boldsymbol{\theta})$ et $g_{00}(p, \boldsymbol{\theta})$ qui maximisent $\tilde{\mu}_{Coll}$ sous les contraintes $\tilde{\mu}_{Inn} = 0$, $\kappa(S_j, S_k) = 0$, et $\tilde{\nu}_{Inn} = 1$. Tout d'abord, calculons tous les éléments.



(a) Tardos original

(b) c et $\hat{\boldsymbol{\theta}}$ donnés

(c) c donné, $\hat{\boldsymbol{\theta}}$ estimé

(d) avec EM, c et $\hat{\boldsymbol{\theta}}$ estimés

FIGURE 4.3 – Probabilité d'accuser correctement le k-ième plus grand score, pour $k \in \{1, \ldots, 5\}$. $m = 1000$, $c = 5$, $n = 5000$. 400 expériences réalisées.

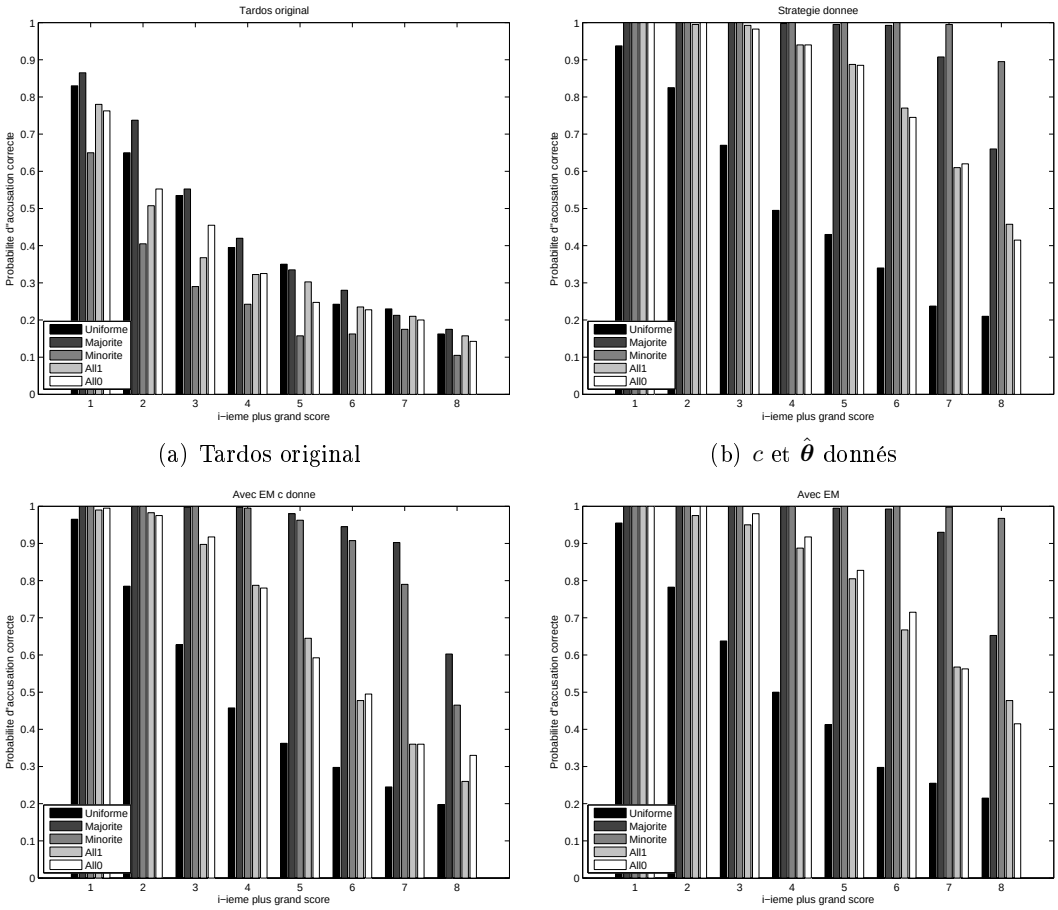
4.3.1 Statistiques concernant les utilisateurs innocents

On pose $q(p, \boldsymbol{\theta}) = \mathbb{P}(Y = 1 | p, \boldsymbol{\theta})$, ce qui peut s'exprimer en terme de modèle de collusion par :

$$q(p, \boldsymbol{\theta}) = \sum_{\sigma=0}^{c} \mathbb{P}(Y = 1 | \Sigma = \sigma) \binom{c}{\sigma} p^{\sigma} (1-p)^{c-\sigma}.$$

On suppose que les scores des innocents sont centrés. Dans ce cas, ils ne sont pas corrélés si :

$$\begin{aligned} \kappa(S_j, S_k) &= \mathbb{E}_p \Big[q(p, \boldsymbol{\theta}) \left(p g_{11}(p, \boldsymbol{\theta}) + (1-p) g_{10}(p, \boldsymbol{\theta}) \right)^2 \\ &+ \left(1 - q(p, \boldsymbol{\theta}) \right) \left(p g_{01}(p, \boldsymbol{\theta}) + (1-p) g_{00}(p, \boldsymbol{\theta}) \right)^2 \Big] = 0. \end{aligned}$$



(a) Tardos original

(b) c et $\hat{\boldsymbol{\theta}}$ donnés

(c) c donné, $\hat{\boldsymbol{\theta}}$ estimé

(d) avec EM, c et $\hat{\boldsymbol{\theta}}$ estimés

FIGURE 4.4 – Probabilité d'accuser correctement le k-ième plus grand score, pour $k \in \{1, \ldots, 8\}$. $m = 1000$, $c = 8$, $n = 5000$. 400 expériences réalisées.

Ce qui donne :

$$\begin{aligned} pg_{11}(p, \boldsymbol{\theta}) + (1-p)g_{10}(p, \boldsymbol{\theta}) &= 0, & (4.11) \\ pg_{01}(p, \boldsymbol{\theta}) + (1-p)g_{00}(p, \boldsymbol{\theta}) &= 0. & (4.12) \end{aligned}$$

Les fonctions g_{10} et g_{01} sont déterminées par les fonctions g_{11} et g_{00}. Donc, il y a seulement deux fonctions à optimiser.

Y a t-il de nouvelles contraintes sur g_{00} et g_{11} données par la relation $\tilde{\mu}_{Inn} = 0$? Par définition, nous avons :

$$\begin{aligned} \tilde{\mu}_{Inn} &= \mathbb{E}_p \left[q(p, \boldsymbol{\theta}) \left(pg_{11}(p, \boldsymbol{\theta}) + (1-p)g_{10}(p, \boldsymbol{\theta}) \right) \right. \\ &+ \left. (1 - q(p, \boldsymbol{\theta})) \left(pg_{01}(p, \boldsymbol{\theta}) + (1-p)g_{00}(p, \boldsymbol{\theta}) \right) \right]. \end{aligned}$$

Mais, avec les relations (4.11) et (4.12), le terme de droite est égal à zéro. Donc nous n'avons pas de nouvelle relation entre g_{00} et g_{11}. La variance a une expression similaire

mais avec les fonctions au carré, et peut se simplifier en utilisant (4.11) et (4.12) :

$$\begin{aligned}\tilde{\nu}_{Inn} &= \mathbb{E}_p\left[q(p,\boldsymbol{\theta})\left(pg_{11}^2(p,\boldsymbol{\theta})+(1-p)g_{10}^2(p,\boldsymbol{\theta})\right)\right.\\ &+ \left.(1-q(p,\boldsymbol{\theta}))\left(pg_{01}^2(p,\boldsymbol{\theta})+(1-p)g_{00}^2(p,\boldsymbol{\theta})\right)\right]\\ &= \mathbb{E}_p\left[q(p,\boldsymbol{\theta})\frac{p}{1-p}g_{11}^2(p,\boldsymbol{\theta})+(1-q(p,\boldsymbol{\theta}))\frac{1-p}{p}g_{00}^2(p,\boldsymbol{\theta})\right].\end{aligned}$$

4.3.2 Statistiques concernant les *colluders*

Nous simplifions l'expression de la moyenne des scores des *colluders* :

$$\begin{aligned}\tilde{\mu}_{Coll} = \mathbb{E}_p \quad [\quad & p\mathbb{P}(Y=0|X=1,p,\boldsymbol{\theta})g_{01}(p,\boldsymbol{\theta})\\ + \quad & p\mathbb{P}(Y=1|X=1,p,\boldsymbol{\theta})g_{11}(p,\boldsymbol{\theta})\\ + \quad & (1-p)\mathbb{P}(Y=0|X=0,p,\boldsymbol{\theta})g_{00}(p,\boldsymbol{\theta})\\ + \quad & (1-p)\mathbb{P}(Y=1|X=0,p,\boldsymbol{\theta})g_{10}(p,\boldsymbol{\theta})].\end{aligned}$$

En utilisant les relations (4.11) et (4.12), et les propriétés

$$\begin{aligned}\mathbb{P}(Y=0|X=0,P=p,\boldsymbol{\theta})+\mathbb{P}(Y=1|X=0,P=p,\boldsymbol{\theta}) &= 1\\ \mathbb{P}(Y=0|X=1,P=p,\boldsymbol{\theta})+\mathbb{P}(Y=1|X=1,P=p,\boldsymbol{\theta}) &= 1\end{aligned}$$

on obtient :

$$\tilde{\mu}_{Coll} = \mathbb{E}_p\left[A(p,\boldsymbol{\theta})\left(pg_{11}(p,\boldsymbol{\theta})+(1-p)g_{00}(p,\boldsymbol{\theta})\right)\right], \tag{4.13}$$

avec $A(p,\boldsymbol{\theta}) = \mathbb{P}(Y=1|X=1,P=p,\boldsymbol{\theta}) - \mathbb{P}(Y=1|X=0,P=p,\boldsymbol{\theta})$. Ces termes sont calculés comme ceci :

$$\begin{aligned}\mathbb{P}(Y=1|X=1,p,\boldsymbol{\theta}) &= \sum_{\sigma=1}^{c}\mathbb{P}(Y=1|\Sigma=\sigma)\binom{c-1}{\sigma-1}p^{\sigma-1}(1-p)^{c-\sigma},\\ \mathbb{P}(Y=1|X=0,p,\boldsymbol{\theta}) &= \sum_{\sigma=0}^{c-1}\mathbb{P}(Y=1|\Sigma=\sigma)\binom{c-1}{\sigma}p^{\sigma}(1-p)^{c-\sigma-1}.\end{aligned}$$

4.3.3 Lagrangien

On veut maximiser $\tilde{\mu}_{Coll}$ sous les contraintes que $\tilde{\nu}_{Inn}=1$, en utilisant un Lagrangien :

$$J(g_{11}(p,\boldsymbol{\theta}),g_{00}(p,\boldsymbol{\theta})) = \tilde{\mu}_{Coll} - \lambda(\tilde{\nu}_{Inn}-1).$$

Cette expression atteint un extremum lorsque de faibles perturbations des fonctions d'entrée ne changent pas sa valeur. Nous définissons une dérivée en considérant la fonction comme un terme linéaire de l'expansion de Taylor : $J(g_{11}(p,\boldsymbol{\theta})+\epsilon(p),g_{00}(p,\boldsymbol{\theta})) = J(g_{11}(p,\boldsymbol{\theta}),g_{00}(p,\boldsymbol{\theta})) + \frac{\partial J(g_{11}(p,\boldsymbol{\theta}),g_{00}(p,\boldsymbol{\theta}))}{\partial g_{11}(p,\boldsymbol{\theta})} + \mathbb{E}_p[o(\epsilon(p))]$.

$$\frac{\partial J(g_{11}(p,\boldsymbol{\theta}),g_{00}(p,\boldsymbol{\theta}))}{\partial g_{11}(p,\boldsymbol{\theta})} = \mathbb{E}_p\left[p\epsilon(p)\left(A(p,\boldsymbol{\theta})-2\lambda\frac{q(p,\boldsymbol{\theta})}{1-p}g_{11}(p,\boldsymbol{\theta})\right)\right].$$

Ceci est égal à zéro pour tout $\epsilon(p)$ si

$$g_{11}(p,\boldsymbol{\theta}) = \frac{1}{2\lambda}\frac{1-p}{q(p,\boldsymbol{\theta})}A(p,\boldsymbol{\theta}). \tag{4.14}$$

En annulant la dérivée selon la seconde fonction, on obtient de même :

$$g_{00}(p,\boldsymbol{\theta}) = \frac{1}{2\lambda}\frac{p}{1-q(p,\boldsymbol{\theta})}A(p,\boldsymbol{\theta}), \tag{4.15}$$

avec λ donné par la contrainte :

$$\lambda = \frac{1}{2}\sqrt{\mathbb{E}_p\left[A^2(p,\boldsymbol{\theta})\frac{p}{q(p,\boldsymbol{\theta})}\frac{1-p}{1-q(p,\boldsymbol{\theta})}\right]}. \tag{4.16}$$

On insère (4.15) et (4.14) dans (4.13), ce qui donne :

$$\tilde{\mu}_{Coll} = \sqrt{\mathbb{E}_p\left[A^2(p,\boldsymbol{\theta})\frac{p}{q(p,\boldsymbol{\theta})}\frac{1-p}{1-q(p,\boldsymbol{\theta})}\right]}. \tag{4.17}$$

4.4 Conclusion

Nous avons exhibé de nouvelles fonctions d'accusation, meilleures que les fonctions précédentes lorsque l'on identifie correctement la stratégie de la collusion. Le gain de performance est très inégal selon les stratégies. Le décodage originel de G. Tardos et B. Skoric *et al* se comporte de la même façon quelle que soit la stratégie, masquant le fait que les stratégies déterministes sont bien moins dangereuses que les probabilistes. La stratégie 'Uniforme' est la plus dure à contrer. La question qui reste à résoudre est de trouver pourquoi certaines stratégies de collusion sont plus dures à contrer que d'autres pour une taille de collusion donnée c, et ainsi identifier la pire d'entre elles. La structure itérative de notre décodeur nous permet d'estimer dynamiquement la taille et la stratégie de la collusion. L'estimation de c est un sujet qui devrait être amélioré, bien qu'une mauvaise estimation de c permet parfois d'obtenir de meilleurs résultats. Il faudrait alors optimiser les fonctions quel que soit le nombre de *colluders*. Une autre technique d'estimation d'attaque a été proposée dans [FPF09a].

Chapitre 5

Un protocole de *fingerprinting* asymétrique basé sur le code de Tardos

Dans ce chapitre, nous proposons la construction détaillée d'un protocole de *fingerprinting* asymétrique utilisant les codes de Tardos. Les protocoles asymétriques ont été proposés il y a une quinzaine d'années, et à notre connaissance il n'existait pas dans la littérature de protocole permettant l'utilisation de codes de Tardos. Nous en avons donc construit un, en nous inspirant des travaux de Pfitzmann et al [PS96].

5.1 Protocole asymétrique

5.1.1 Description du problème

Nous cherchons à construire un schéma de *fingerprinting* asymétrique en utilisant les codes traçants de Tardos. Un état de l'art des schémas de *fingerprinting* asymétrique est proposé dans le chapitre 2, et un état de l'art des codes de Tardos dans la section 3.3. Les codes de Tardos ne présentent pas de structure déterministe, c'est pourquoi leur utilisation nécessite une attention particulière. Le schéma de *fingerprinting* asymétrique permet aux acheteurs honnêtes de se protéger d'un vendeur malhonnête.

Nous souhaitons construire un protocole qui ne fait intervenir que le nombre minimal de tierces parties. En effet, la potentielle corruption des parties supposées neutres est un problème. C'est pourquoi nous essayons autant que possible que toutes les opérations se déroulent entre le vendeur et l'acheteur.

5.1.2 Rappel des notations concernant les codes de Tardos

Nous utilisons un code de Tardos binaire. Soit n le nombre d'utilisateurs, m la longueur du code et c le nombre maximum de *colluders*. Soit $\mathbf{X}$ la matrice contenant l'ensemble des mots du code. On désigne par $\mathbf{X}_j = (X_{j1}, X_{j2}, \ldots, X_{jm})$ le mot de code de l'utilisateur j. On tire m valeurs $p_i \in [0, 1]$ indépendantes et identiquement

distribuées suivant la fonction de densité $f(p) = \frac{1}{\pi\sqrt{p(1-p)}}$. Ces valeurs constituent le vecteur $\mathbf{p}$. Les X_{ji} sont alors tirés de façon aléatoire et indépendante, en suivant la loi de Bernoulli de paramètre p_i, c'est à dire $\mathbb{P}(X_{ji} = 1) = p_i$.
Chaque utilisateur reçoit une copie contenant une marque différente. Pour la partie accusation, soit $\mathbf{Y}$ la marque extraite de la copie pirate. Avec cette marque et le mot de code qui lui est associé, on calcule un score S_j d'accusation pour l'utilisateur j, sachant qu'on n'est pas obligé de considérer tous les utilisateurs pour mener l'accusation. Le score est calculé de telle sorte que les plus grands scores en espérance sont ceux des *colluders*. Pour ce calcul, on utilise quatre fonctions d'accusations.

$$S_j = \sum_{i=1}^{m} U(Y_i, X_{ji}, p_i) \tag{5.1}$$

$$U(1,1,p) = g_{11}(p), \qquad U(0,0,p) = g_{00}(p), \qquad U(0,1,p) = g_{01}(p), \qquad U(1,0,p) = g_{10}(p).$$

Pour une bonne utilisation du code de Tardos, il faut impérativement la condition suivante :

Les utilisateurs ne doivent pas connaître le vecteur de probabilité $\mathbf{p}$, sinon ils peuvent contrer l'accusation.

5.1.3 Comportements honnêtes et malhonnêtes de l'acheteur et du vendeur

Le vendeur intervient dans différentes phases du protocole de *fingerprinting*, c'est pourquoi nous pouvons considérer différents modèles de vendeur malhonnête.

5.1.3.1 Triche lors de la phase d'accusation

Le vendeur triche au moment de l'accusation. Il change le vecteur $\mathbf{p}$ au moment de calculer les scores afin de faire accuser un innocent. Cette idée vient de Teddy Furon et Ingemar Cox, qui ont voulu étudier la possibilité d'une attaque de la part d'un vendeur malhonnête au moment de l'accusation. Cette attaque porte sur le vecteur $\mathbf{p}$, qui a servi à générer les identifiants et qui doit aussi servir à calculer les scores d'accusation. Que se passe-t-il si ce n'est pas le même vecteur qui est utilisé ?

La figure 5.1 montre que l'effet est très important, puisque même une modification assez faible (environ 20 %) du vecteur change complètement les scores des acheteurs. Si tous les scores des acheteurs sont calculés, le juge peut se rendre compte de la modification, mais si le calcul du score porte sur un seul utilisateur, alors cet acheteur peut être faussement accusé.

Ce résultat montre l'importance de garantir l'intégrité du vecteur $\mathbf{p}$ tout au long du processus.

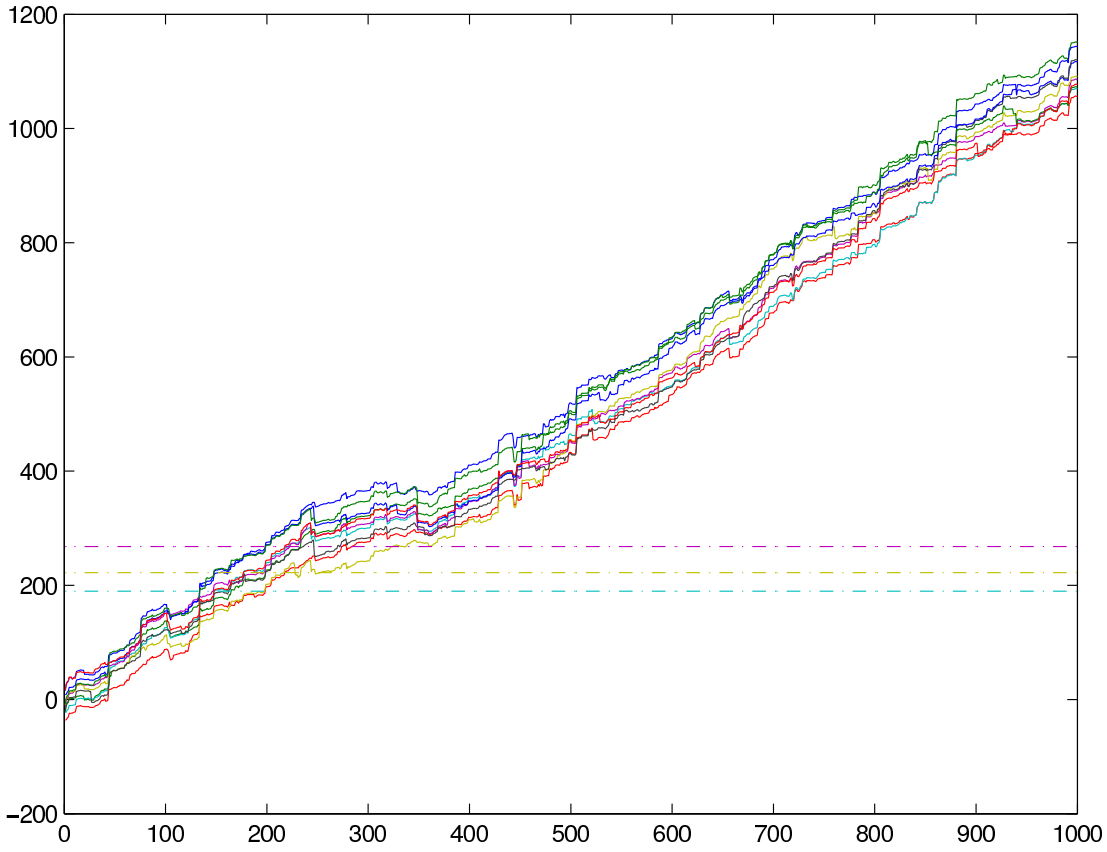


FIGURE 5.1 – Les scores d'accusation en fonction du nombre d'éléments du vecteur $\mathbf{p}$ modifiés. $m = 1000$ et $c = 3$. Les lignes colorées en plein montrent comment les scores de 10 acheteurs innocents pris au hasard augmentent. Les lignes en pointillé montrent les scores des *colluders* avant la modification.

5.1.3.2 Redistribution

Le vendeur malhonnête triche en distribuant la copie marquée par l'identifiant d'un acheteur afin de faire accuser faussement celui-ci de redistribution illégale.

Ce problème correspond à celui posé lors d'une utilisation classique des schémas de *fingerprinting* de la littérature.

La technique d'insertion est connue du vendeur seul, mais il ne doit pas avoir en sa possession la copie marquée par l'identifiant d'un acheteur. Seul l'acheteur doit posséder l'exemplaire marqué par son identifiant.

On voit donc que l'identifiant ne doit pas être intégralement connu du vendeur.

La phase d'insertion et de distribution des copies est donc compliquée. Pour cette partie, il existe deux façons de procéder. Dans la première, l'insertion se fait au cours d'un protocole 2-partie dans lequel l'acheteur fournit sa marque et le vendeur fournit le document qui doit être marqué. Dans la seconde, le vendeur construit deux versions de son document, l'un étant marqué avec des '0', l'autre avec des '1', et un protocole 2-partie se charge de construire le document de l'acheteur sans que le vendeur ait connaissance de l'identifiant de l'acheteur et sans que l'acheteur puisse avoir connaissance de la version qui ne correspond pas à son identifiant. Pour chaque position, si l'identifiant de l'acheteur est, par exemple, un '1', l'acheteur doit recevoir le morceau de document correspondant marqué avec un '1', et ne doit avoir aucune information sur le morceau marqué avec un '0'.

Mais le vendeur doit aussi être capable de retrouver des utilisateurs malhonnêtes en cas de distribution frauduleuse du document. Il doit donc connaître une partie de l'identifiant, qui lui servira à mener une pré-accusation, la deuxième partie de l'identifiant servira à confirmer ou infirmer cette accusation. Ceci nous amène à considérer une phase au cours de laquelle l'acheteur va devoir fournir au vendeur une partie de son identifiant. Il faudra s'assurer que c'est bien le même morceau d'identifiant qui est fourni au vendeur que celui qui est inséré dans le document.

Nous proposons que l'acheteur génère tout l'identifiant et en fournisse une partie au vendeur. Une autre façon de faire serait la suivante : l'acheteur génère un identifiant de la même longeur que la partie fournie au vendeur dans la première façon de faire, et l'insère lui-même dans le document, et l'acheteur génère le reste de l'identifiant dont il ne révèle rien au vendeur. Cela revient au même pour le vendeur, qui de toute façon apprend une partie de l'identifiant qui est inséré dans la copie. On peut l'appliquer au code de Tardos assez facilement puisque dans un mot du code de Tardos, toutes les lettres de l'identifiant sont générées de façon indépendante. Cela change pour l'acheteur qui se retrouve avec une copie marquée par un identifiant dont il ignore une partie. Cependant, l'avantage de la première façon de procéder est que la révélation de cette partie du mot va pouvoir servir de vérification d'intégrité au moment de l'insertion. Le vendeur va utiliser ces bits révélés pour deux choses : d'abord pour pouvoir mener une pré-accusation si un document piraté est retrouvé, mais aussi pour s'assurer que les bits fournis par l'acheteur dans le protocole 2-partie utilisé pour l'insertion sont bien issus de la phase de génération de l'identifiant.

Les problèmes qui se posent peuvent être résumés ainsi : Comment garantir l'inté-

grité des données manipulées aux cours des différentes opérations ?

Vecteur p : doit être le même qui sert pour la génération et pour l'accusation, doit être inconnu des acheteurs,

Identifiant : l'identifiant doit être un mot du code de Tardos décidé par le vendeur et doit être inconnu du vendeur. C'est bien entendu le même mot qui doit être inséré dans le document. Le vendeur doit en connaître une partie afin de mener à bien l'accusation.

5.2 Utilisation d'un *Oblivious Transfer*

5.2.1 Idée générale

Pour permettre la mise en place d'un schéma de *fingerprinting* asymétrique avec les codes de Tardos, il faut que ce soit l'utilisateur qui génère lui-même son identifiant. Cet identifiant doit être un mot d'un code de Tardos dont les paramètres ont été choisis par le vendeur. En particulier, c'est le vendeur qui a construit le vecteur $\mathbf{p}$, et ce vecteur doit rester inconnu des utilisateurs (sous peine de voir des utilisateurs malhonnêtes mener à bien des attaques par collusion).

Voici donc notre problème :

Bob doit effectuer un tirage aléatoire de 0 ou 1 selon une loi de Bernoulli de paramètre q sans connaître ce paramètre et sans rien en apprendre.

Pour résoudre ce problème, nous avons utilisé une quantification du paramètre q. Le tirage aléatoire selon la probabilité q devient alors un tirage uniforme d'un élément dans une liste, la liste étant construite de telle sorte que le nombre d'éléments égaux à 1 dans la liste de 0 et de 1 correspond à la quantification de la valeur de q. Pour l'application au vecteur de biais de Tardos, nous procédons donc comme suit : pour la position i, nous prenons pour valeur du paramètre q la probabilité p_i. Le vendeur construit une liste de longueur N contenant L_i éléments ayant la valeur '1' et $N - L_i$ éléments ayant la valeur '0'. L_i est calculé de telle sorte que $L_i/N \simeq p_i$.

Le problème à résoudre devient alors

Bob doit effectuer un tirage aléatoire d'un élément dans une liste sans rien apprendre des éléments non choisis et sans qu'Alice sache lequel il a tiré.

5.2.2 Oblivious Transfer

Un *Oblivious Transfer* est un schéma qui permet à Bob de choisir k éléments au hasard dans une liste de N éléments possédée par Alice, de telle sorte que :

1. Bob obtient des éléments qui sont réellement dans la liste ;
2. Bob n'obtient pas d'information sur les éléments qu'il n'a pas choisis ;
3. Alice ne sait pas quels sont les éléments choisis par Bob.

Les protocoles permettant de réaliser cet échange, *Oblivious Transfer protocols (OT)*, ont été introduits par les cryptographes dans [Rab81] et ont donné par la suite un grand nombre de travaux et de publications, *e.g.* [NP99b, CT05, GH07].

5.2.2.1 Etat de l'art

Les protocoles d'Oblivious Transfer sont étudiés de la même façon que les protocoles multi-parties. On notera OT_k^N un protocole *Oblivious Transfer* permettant un choix de k parmi N. Dans la littérature, on peut trouver des protocoles OT_1^2, OT_1^N et OT_k^N. Lorsque $k \geq 1$, on parle de protocoles OT adaptifs si les k éléments sont choisis un par un de façon adaptive (Bob attend de recevoir un élément avant d'en choisir un autre), et de protocoles OT non-adaptifs si ils sont désignés simultanément. Un protocole adaptif k parmi N est noté $OT_{k\times 1}^N$. Naor and Pinkas [NP99b, NP99a] ont été les premiers à étudier ce cas adaptif. Les premiers protocoles proposés étaient des protocoles OT_1^2, introduits par [Rab81] en 1981 . On peut considérer que les protocoles OT_k^N peuvent aussi être utilisés comme OT_1^N en prenant $k = 1$. Cependant, les protocoles OT_k^N sont en général construits spécifiquement pour permettre le choix de plusieurs éléments, et les utiliser pour en choisir un seul est en général plus coûteux que d'utiliser un protocole OT_1^N, qui est, lui, construit spécifiquement pour choisir un seul élément. Mais certains protocoles OT_k^N peuvent s'utiliser en OT_1^N en restant optimums. C'est intéressant pour des schémas globaux qui utilisent un protocole OT_1^N et un protocole OT_k^N puisque cela permet d'implémenter un seul protocole.

La sécurité des Oblivious Transfer est étudiée spécifiquement selon les modèles liés aux comportements des protagonistes. Les classes de sécurité sont classées et discutées dans [CNS07] et [GH07]. Les modèles de sécurité pour les OT sont, du plus faible au plus fort :

- *honest-but-curious model* dans lequel personne ne triche pendant l'exécution du protocole, mais les joueurs sont curieux et essaient d'obtenir des informations en observant les données échangées.
- *half simulation* (introduit par [NP05], le vendeur tricheur ou le receveur tricheur sont étudiés séparément(étude de sécurité locale).
- *full simulation* (introduit dans [CNS07], étude globale des vendeurs et receveurs tricheurs ; étude de sécurité globale).
- En plus, le modèle UC (*Universal Composability*) a été introduit dans [Can02] pour étudier le comportement et la sécurité des protocoles qui sont basés sur des primitives cryptographiques concurrentes et composables entre elles.

L'attaque de *selective failure* concerne les protocoles adaptifs $OT_{k\times 1}^N$. Le vendeur peut faire échouer certains échanges volontairement (par exemple, lors du premier tirage, le transfert du premier élément échoue, lors du deuxième tirage, c'est le transfert du deuxième élément qui échoue, etc.). Si l'acheteur se plaint de l'échec du transfert, il révèle son choix.

La sécurité des protocoles adaptifs est plus dure à obtenir que celle des cas non-adaptifs [Lin08].

Camenish et al. [CNS07] ont établi que les protocoles présentés dans [NP05] sont

	Echanges
Half simulation	
Naor-Pinkas [NP99b]	$lklogN + 1/2$
Chu [CT05]	$O(k) + 1/2$
Full simulation	
Camenish1 [CNS07]	$4k + 1/2$
Camenish2 [CNS07]	$O(k) + 1/2$
Green1 [GH07]	$k + 1/2$
Universal Composability	
Green2 [GH08]	$k + 1/2$

TABLE 5.1 – Comparaison de divers protocoles adaptifs $OT^N_{k\times 1}$, [GH08]

sûrs sous les conditions de *half simulation*, et vulnérables à l'attaque de *selective failure* (mais peuvent être améliorés pour résister à cette attaque). Le schéma de [OK04] est sûr dans le modèle *half simulation*.

5.2.2.2 Protocole présenté par Chu

Nous avons choisi d'utiliser le protocole proposé par Chu et Tzeng [CT05] car il présente un bon compromis sécurité/complexité (voir en annexe le tableau et 5.2.2.1). Il est en effet sûr pour le modèle *half simulation* en étant moins coûteux que le schéma proposé par Naor et Pinkas. Les schémas sûrs dans le modèle de *full simulation* sont très compliqués à manipuler.

Voici le protocole d'Oblivious Transfer proposé par Chu dans [CT05].
C'est un protocole $OT^N_{k\times 1}$ qui peut être utilisé en OT^N_1 sans perte de performance.

- Les paramètres du système sont (g, H_1, H_2, G_q). Les H_i sont des fonctions de hachage. Ces paramètres sont publics.
- Le Vendeur V possède les messages $m_1, m_2, \ldots, m_N$
- l'Acheteur A veut choisir les messages correspondants aux indices : $\sigma_1, \sigma_2, \ldots, \sigma_k$

Phase d'engagement :

1. $x \in_R \mathbb{Z}_q^*$, $w_i = H_1(i)$. V calcule $c_i = m_i \oplus H_2(w_i^x)$ et $y = g^x$.
2. V envoie à A $y, c_1, c_2, \ldots, c_N$

Phase de transfert :
Pour chaque σ_j, faire les actions suivantes :

1. A choisit aléatoirement $a_j \in \mathbb{Z}_q^*$ et calcule $w_{\sigma_j} = H_1(\sigma_j)$, $A_j = w_{\sigma_j} g^{a_j}$.
2. A envoie A_j à V.
3. V renvoie $D_j = (A_j)^x$.
4. A calcule $K_j = D_j / y^{a_j}$ et obtient $m_{\sigma_j} = c_{\sigma_j} \oplus H_2(K_j)$.

5.2.3 Approche *Commutative Encryption Scheme*

Le *Commutative Encryption Scheme*, tel qu'introduit dans [BD01] constitue une première solution possible pour résoudre notre problème. Un tel schéma s'appuie sur

l'utilisation d'une primitive de chiffrement commutatif.

5.2.3.1 Chiffrement commutatif

Un schéma de chiffrement **CE** est qualifié de *chiffrement commutatif* si

Définition 15 (définition habituelle)**.** pour n'importe quel ensemble de cléfs k_A et k_B et n'importe quel texte clair m, on a

$$\mathbf{CE}(k_B, \mathbf{CE}(k_B, m)) = \mathbf{CE}(k_A, \mathbf{CE}(k_B, m)).$$

Nous proposons ici une réécriture équivalente qui permet de couvrir le cas des chiffrements probabilistes (afin de prendre en compte les schémas de chiffrement qui atteignent la sécurité sémantique) [1]

Définition 16 (reformulation)**.**

$$\mathbf{CE}^{-1}(k_B^*, \mathbf{CE}^{-1}(k_A^*, \mathbf{CE}(k_B, \mathbf{CE}(k_A, m)))) = m,$$

où k^* désigne la clef de déchiffrement associée à k. Dans le cas d'un chiffrement symétrique, $k^* = k$ est une clef secrète, tandis que dans le cas d'un chiffrement asymétrique, k est la clef publique et k^* la clef privée.

5.2.3.2 *Commutative Encryption Scheme* (CES)

Bob veut accéder à un élément d'une liste détenue par Alice.

1. Soient m_1, m_2, ...,m_n les données d'Alice. Alice choisit n clefs secrètes K_1, K_2, ..., K_n pour un schéma de chiffrement symétrique **E** (*e.g.* AES, DES) et une clef k_A pour la primitive de chiffrement commutatif **CE** 15 . Soit

$$\begin{aligned} C_1 &= \mathbf{E}(K_1, m_1) & D_1 &= \mathbf{CE}(k_A, K_1) \\ C_2 &= \mathbf{E}(K_2, m_2) & D_2 &= \mathbf{CE}(k_A, K_2) \\ &\cdots & &\cdots \\ C_n &= \mathbf{E}(K_n, m_n) & D_n &= \mathbf{CE}(k_A, K_n) \end{aligned}$$

   Les couples (C_j, D_j) sont accessibles publiquement.

2. Supposons que Bob veut acquérir l'élément m_i. Bob charge (C_i, D_i) et choisit une clef secrète k_B pour **CE**. Il chiffre D_i avec et envoie le résultat $U = \mathbf{CE}(k_B, D_i)$ à Alice.
3. Alice déchiffre U avec k_A et envoie $W = \mathbf{CE}^{-1}(k_A^*, U)$ à Bob. Bob déchiffre W avec k_B pour avoir $K_i = CE^{-1}(k_B^*, W)$, et peut obtenir $m_i = \mathbf{E}^{-1}(K_i, C_i)$.

1. ce n'est pas écrit sous cette forme dans la littérature, mais nous trouvons cette équation plus rigoureuse.

Pour utiliser le schéma de façon pratique, il faut choisir la bonne primitive **CE**. On peut remarquer que dans ce protocole, **CE** est toujours utilisé de façon 'symétrique', le chiffrement avec k_A et le déchiffrement avec k_A^* sont tous les deux effectués par Alice, le chiffrement avec k_B et le déchiffrement avec k_B^* sont tous les deux effectués par Bob. On voit alors qu'utiliser un chiffrement symétrique ou asymétrique importe peu, fonctionnellement parlant. Ce qui est important est le niveau de sécurité que l'on souhaite atteindre. Ici, on ne peut pas atteindre la sécurité inconditionelle : le seul schéma de chiffrement qui permet ce niveau de sécurité est le One-Time-Pad à condition de n'utiliser chaque clef de chiffrement qu'une seule fois, mais vu qu'ici on utilise la même clef k_A pour chiffrer toutes les clefs K_i, on ne pourra pas atteindre un aussi haut niveau de sécurité . De fait, le meilleur niveau de sécurité que l'on peut espérer atteindre est la sécurité sémantique, qui implique d'utiliser un schéma de chiffrement probabiliste.

Les schémas pratiques proposés dans la littérature [BD01, HC05, WZW03] n'atteignent pas la sécurité sémantique. Les auteurs de ces articles ont remarqué cette faiblesse et ont ajouté quelques lignes pour indiquer que le schéma pourrait atteindre cette sécurité sémantique en ajoutant des motifs aléatoires (*padding*). La sécurité sémantique est très importante pour nous, car les éléments qui sont échangés sont des bits et l'utilisateur doit être incapable de distinguer les 0 chiffrés des 1 chiffrés, que ce soit pendant l'étape de génération du *fingerprint* ou la découverte du *halfword*. Un exemple de schéma de chiffrement asymétrique probabiliste qui atteint la sécurité sémantique est ElGamal, qui est IND-CPA. A notre connaissance, il n'y a pas de chiffrement commutatif qui soit IND-CCA1 (et non plus IND-CCA2). Un rappel sur la sécurité des chiffrements est fait en annexe D.1. Pour plus de détails sur les classes de sécurité mentionnées ou les schémas de chiffrement, on peut se référer à [Gol04, vT05, FG07].

On devrait pouvoir construire des schémas d'Oblivious Transfer atteignant la sécurité IND-CCA1 à partir de schémas de chiffrement IND-CCA1 et IND-CCA2 , comme Cramer-Shoup par exemple, mais il s'agit alors d'un travail complet de recherche pour les définir proprement et prouver leur sécurité étant donné qu'une telle preuve nécessiterait dans notre cas plusieurs oracles, contrairement à leur version usuelle

Une variante du schéma CES a été proposée dans [WZW03], sous le nom de *Two-Lock cryptosystem*, dans laquelle Alice et Bob sont susceptibles d'utiliser des schémas de chiffrement différents, soient $\mathbf{CE}_A$ et $\mathbf{CE}_B$ respectivement, ce qui donne pour n'importe quelle clef k_A utilisée par Alice, et k_B utilisée par Bob, et n'importe quel message m,

$$\mathbf{CE}_B(k_B, \mathbf{CE}_A(k_A, m)) = \mathbf{CE}_A(k_A, \mathbf{CE}_B(k_B, m))$$

ou plus précisément, si on veut couvrir le cas probabiliste

$$\mathbf{CE}_A{}^{-1}(k_A^*, \mathbf{CE}_B{}^{-1}(k_B^*, \mathbf{CE}_A(k_A, \mathbf{CE}_B(k_B, m)))) = m.$$

Ces primitives sont alors utilisées comme suit

1. Alice envoie à Bob : $D = \mathbf{CE}_A(k_A, m)$.
2. Bob choisit une clef secrète k_B, chiffre D avec et envoie le résultat $U = \mathbf{CE}_B(k_B, D_i)$ à Alice.

3. Alice déchiffre U et envoie le résultat $W = \mathbf{CE}_A^{-1}(k_A, U)$ à Bob.
4. Bob déchiffre : $m = \mathbf{CE}_B^{-1}(k_B, W)$.

Lorsque $\mathbf{CE}_A = \mathbf{CE}_B$, on retombe sur le *Commutative Encryption Scheme* mentionné plus tôt, sauf que dans le schéma proposé, ils n'utilisent pas de seconde liste.

Quelques implementations pratiques de ce two-lock cryptosystem ont été proposées : une première basée sur le problème du sac à dos [WZW03], qui a été cassée dans [ZWFB04], avec une étude montrant qu'on ne peut pas améliorer le schéma, une deuxième basée sur le problème du logarithme discret [WZW03], et une troisième basée sur RSA [HC05].

5.2.4 Comparaison

On constate que le *Commutative Encryption Scheme* remplit lui aussi la fonction d'*Oblivious Transfer*. Cependant, le fait que cette construction ait été proposée par des chercheurs en dehors de la communauté des cryptographes fait qu'elle n'a pas été reconnue comme un *Oblivious Transfer* immédiatement et que ses exigences de sécurité n'ont pas été évaluées et validées comme les autres *OT* issus de la communauté cryptographique.

C'est pour cette raison que nous déclinons notre solution en deux versions : une version qui utilise un *Commutative Encryption Scheme*, qui est très intéressante construite avec un chiffrement commutatif qui atteint la sécurité sémantique (un tel schéma n'ayant pas encore été détaillé à ce jour, mais nous pensons qu'il existe), et une version utilisant un protocole *OT* issu de la communauté cryptographique, dont la sécurité a été étudiée et validée par cette communauté.

On notera que le commutative Encryption Scheme possède déjà la structure que nous voulons utiliser de par son utilisation de deux listes, dont une est publique.

5.3 Description du protocole asymétrique

5.3.1 Phase 1 : Génération du *fingerprint*

La génération du *fingerprint* se fait en deux étapes, l'étape 1 concerne la génération des listes communes à tous les utilisateurs, et se fait donc une seule fois, et l'étape 2 concerne la génération des listes personnalisées pour chaque utilisateur. Au cours de l'étape 2, les deux parties procèdent à un Oblivious Transfer et vont devoir conserver certaines données échangées au cours de ce protocole afin de prouver l'intégrité de données qui seront échangées par la suite. Chaque bit du *fingerprint* donne lieu à un tirage indépendant.

Les données à conserver dépendent du protocole d'Oblivious Transfer utilisé.

Etape 1 Cette étape ne concerne que le vendeur. Pour toutes les positions, le vendeur construit une liste de longueur N. Chaque liste correspond à une position i du vecteur

$$
\begin{array}{lcccc}
 & \overset{quantification}{} & & \overset{Engagement}{} & \\
p_1 & \longrightarrow & (0\|\mathtt{ref}_{0,1}, 1\|\mathtt{ref}_{1,1}, \ldots, 1\|\mathtt{ref}_{1,1}) & \longrightarrow & \mathcal{C}_1 = (C_{1,1}, C_{2,1}, \ldots, C_{N,1}) \\
p_2 & \longrightarrow & (0\|\mathtt{ref}_{0,2}, 0\|\mathtt{ref}_{0,2}, \ldots, 0\|\mathtt{ref}_{0,2}) & \longrightarrow & \mathcal{C}_2 = (C_{1,2}, C_{2,2}, \ldots, C_{N,2}) \\
p_3 & \longrightarrow & (0\|\mathtt{ref}_{0,3}, , 1\|\mathtt{ref}_{1,3}, \ldots, 1\|\mathtt{ref}_{1,3}) & \longrightarrow & \mathcal{C}_3 = (C_{1,3}, C_{2,3}, \ldots, C_{N,3}) \\
 & & \vdots & & \\
p_m & \longrightarrow & (1\|\mathtt{ref}_{1,m}, 0\|\mathtt{ref}_{0,m}, \ldots, 1\|\mathtt{ref}_{1,m}) & \longrightarrow & \mathcal{C}_m = (C_{1,m}, C_{2,m}, \ldots, C_{N,m})
\end{array}
$$

FIGURE 5.2 – Etape 1 : les listes $\mathcal{C}_i = \{C_{k,i}\}_{k=1}^{N}$ représentant les p_i engagés qui sont stockées dans un WORM

p. Pour chaque liste, une valeur L_i correspondant à la quantification de p_i est calculée : $L_i/N \simeq p_i$. Les objets de la liste sont la concaténation d'un symbole binaire et d'une chaîne de caractères. Il n'y a que deux versions d'un objet dans la liste C_i. Pour L_i objets, $O_{k,i} = (1\|\mathtt{ref}_{1,i})$, et $O_{k,i} = (0\|\mathtt{ref}_{0,i})$ pour les $N - L_i$ restants. L'utilisation de la chaîne de caractères $\{\mathtt{ref}_{X,i}\}$ dépend de la méthode de distribution comme expliqué dans la Sec. 5.4.1. L'objet $O_{k,i}$ fait l'objet d'un engagement avec la clef $K_{k,i}$ et est conservé dans la liste $\mathcal{C}_i = \{C_{k,i}\}_{k=1}^{N}$. Il y a donc autant de listes différentes $\mathcal{C}_i$ que la longueur m du *fingerprint*, voir figure5.3.1. Ces listes sont accessibles dans un répertoire Write Once Read Many (WORM) [OB09] dont l'accès est autorisé pour tous les utilisateurs. Comme son nom l'indique, personne ne peut modifier ou effacer ce qui a été mis pour la première fois dans un répertoire WORM ; de plus, n'importe qui peut vérifier son intégrité. Cette liste correspond à la liste C_i dans le schéma *Commutative Encryption* (5.2.3.2) décrit auparavant.

Etape 2 Cette étape est un échange entre le vendeur et l'acheteur. A l'opposé des listes $\mathcal{C}$, les listes $\mathcal{D}$ sont construites spécifiquement pour un acheteur j.

L'acheteur doit faire son choix de façon totalement aveugle, rien ne doit pouvoir le guider, l'attribution doit se faire de manière aléatoire, afin de conserver au code de Tardos ses propriétés traçantes. Par exemple, l'acheteur ne doit pas pouvoir décider de prendre le même élément qu'un autre acheteur qui aurait déjà effectué son tirage. Plus précisément, l'utilisateur choisit un élément mais ne doit pas savoir ce qu'il choisit. C'est vraiment l'idée de tirer une carte dans un jeu présenté face cachée.

C'est pour obtenir cette propriété que le vendeur choisit une permutation $\pi_j(.)$ sur $[N]$ qui s'applique sur les listes $\mathcal{D}$. Cette permutation étant différente pour chaque utilisateur, elle empêche des utilisateurs malhonnêtes de mettre en place une stratégie qui leur permettrait de ne pas faire un tirage aléatoire (ces *colluders* pourraient, par exemple, faire exprès d'avoir le même élément, ou alors faire exprès d'avoir un élément différent).

Acheteur et vendeur procèdent à un Oblivious Transfer sur les listes de clefs, avec un Commutative Encryption Scheme (section 5.2.3.2) ou bien avec un protocole de Chu (section 5.2.2.2)

L'acheteur obtient suite à ce protocole une clef. Cette clef lui permet l'accès à l'objet $O_{\mathsf{ind}(j,i),i}$, stocké sous forme chiffrée dans le WORM. Il contient le symbole $b_{\mathsf{ind}(j,i),i}$. Ceci va correspondre à la valeur du i-ième bit de son *fingerprint*, $X_{j,i} = b_{\mathsf{ind}(i,j),i}$, qui est égal

à '1' avec la probabilité p_i.

Au cours de cette étape, nous voulons que l'acheteur génère un *fingerprint* que lui seul connaîtra et qui respectera les propriétés d'un code de Tardos dont le vecteur de probabilité $\mathbf{p}$ a été généré par le vendeur.

Détail de l'utilisation du CES L'acheteur a accès à une liste $\mathcal{D}_{j,i}$ de N éléments telle que $D_{j,i,k} = \mathbf{CE}(S_j, (\pi_j(k) \| K_{\pi_j(k),i}))$. L'acheteur j choisit un objet dans la liste, par exemple le $k(j,i)$-ième objet. Il envoie le chiffré correspondant $U_{k(j,i),i} = \mathbf{CE}(R_{j,i}, D_{j,i,k(j,i)})$ qui sera déchiffré par le vendeur avec S_j puis renvoyé à l'acheteur qui, finalement, obtient l' index $\mathsf{ind}(j,i) = \pi_j(k(j,i))$ et la clef $K_{\mathsf{ind}(j,i),i}$. Cette clef lui permet l'accès à l'objet $O_{\mathsf{ind}(j,i),i}$, stocké sous forme chiffrée dans le WORM. Il contient le symbole $b_{\mathsf{ind}(j,i),i}$. Ceci va correspondre à la valeur du i-ième bit de son *fingerprint*, $X_{j,i} = b_{\mathsf{ind}(i,j),i}$, qui est égal à '1' avec la probabilité p_i.

Le vendeur conserve dans un fichier de logs les valeurs S_j et $U_{k(j,i),i}$, l'utilisateur conserve $R_{j,i}$ dans ses fichiers.

Le schéma général est illustré par la figure 5.3

Détail de l'utilisation du protocole de Chu [CT05]

- **Phase d'engagement :**
  1. $x \in_R \mathbb{Z}_q^*$, $w_i = H_1(i)$ Le vendeur calcule $c_i = \mathcal{D}_i \oplus H_2(w_i^x)$ et $y = g^x$.
  2. Le vendeur envoie à l'utilisateur $y, c_1, c_2, \ldots, c_N$
- **Phase de transfert :**
  1. L'utilisateur choisit aléatoirement $a_j \in \mathbb{Z}_q^*$ et calcule $w_{\sigma_j} = H_1(\sigma_j)$, $A_j = w_{\sigma_j} g^{a_j}$.
  2. L'utilisateur envoie A_j au vendeur.
  3. Le vendeur renvoie $D_j = (A_j)^x$.
  4. L'utilisateur calcule $K_j = D_j / y^{a_j}$ et obtient $\mathcal{D}_{\sigma_j} = c_{\sigma_j} \oplus H_2(K_j)$.

L'utilisateur conserve a_j dans ses fichiers. Dans les étapes suivantes, il en aura besoin pour prouver qu'il fournit au vendeur des bits qui proviennent bien de son *fingerprint*.

5.3.2 Phase 2 : Révélation du *halfword*

Pour que le processus d'accusation fonctionne correctement, (voir Sec. 5.4.2), le vendeur va demander à l'acheteur j de révéler $m_h < m$ bits de son *fingerprint* (qui a été généré en phase 1). Ceci afin de construire ce que l'on va appeler le *halfword* [PS96], qui permet au vendeur d'établir lors d'un pré-accusation une liste d'utilisateurs suspects qui pourra être transmise au juge. (voir Sec. 5.4.2). Les propriétés suivantes sont très importantes : l'acheteur j ne sait pas quels bits de son *fingerprint* ont été révélés, et le vendeur demande les mêmes indices à tous les utilisateurs.

Là encore, nous proposons d'utiliser un protocole d'Oblivious Transfer de la partie. 5.2.2.

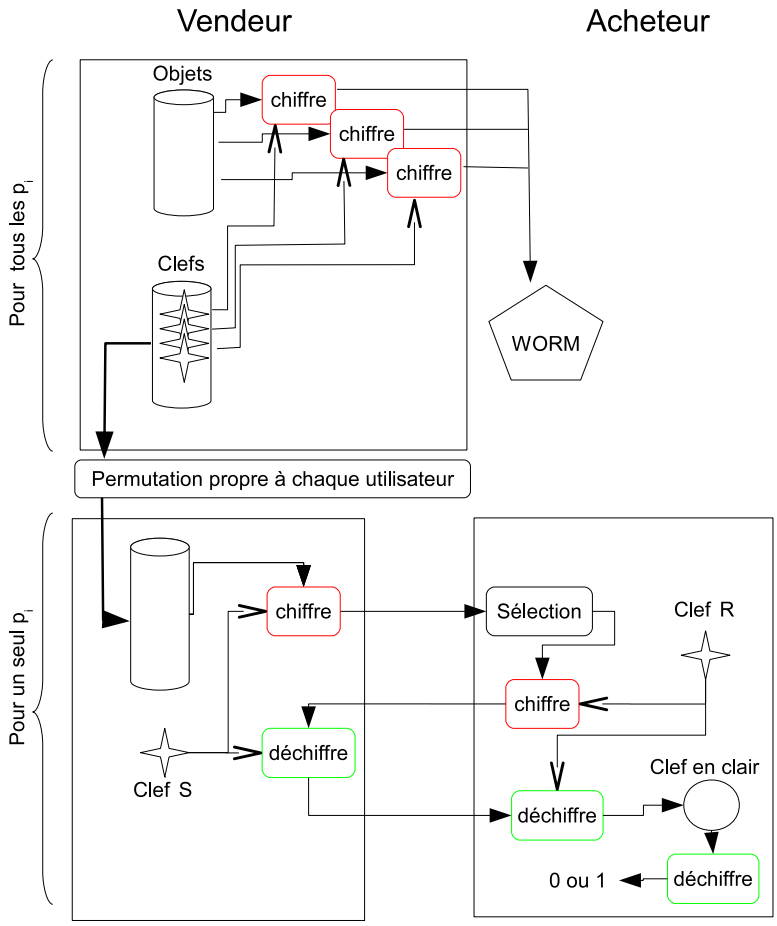


FIGURE 5.3 – Utilisation du *Commutative Encryption Scheme* pour la génération du *fingerprint*

- Maintenant, l'acheteur j joue le rôle d'Alice `A`, et le vendeur le rôle de Bob `B`, $N = m$, et les objets $O_i = \boxed{\text{choix}} \| \texttt{alea}_{i,j}$.
- Les $\boxed{\text{choix}}$ sont les éléments générés dans les échanges lors de la génération du *fingerprint* ($(R_{i,j}$ si il a été utilisé un CES, $a_{i,j}$ dans le cas du protocole de Chu). Ce sont les 'clefs' qui cachent le choix de l'acheteur.
- Ces éléments sont les m clefs secrètes sélectionnées par l'acheteur j pendant Sec. 5.3.1 concaténées avec des chaînes aléatoires $\texttt{alea}_{i,j}$ qui seront crées par l'acheteur j. Cet alea sera utile pendant la partie de personnalisation du contenu. (voir Sec. 5.4.1).
- En suivant le protocole, le vendeur sélectionne m_h de ces objets.
- Ils lui permettent d'obtenir l' index de l'objet sélectionné lors de la phase précédente.
- **Ceci empêche un *colluder* de nier le symbole de son *fingerprint* et de copier le symbole d'un complice.**
- à la fin du processus, le vendeur a appris quel élément a été choisi par l'acheteur j à l' index i. A cet effet, il obtient pour finir m_h couples $(X_{j,i}, \texttt{alea}_{k(i,j),i})$ associés à un acheteur j donné.

Grâce à cette seconde partie de notre protocole, le vendeur a connaissance de m_h bits du *fingerprint* sans rien savoir des autres, et l'acheteur j ne sait pas quels sont les bits de son *fingerprint* qui sont connus même si le vendeur choisit toujours les mêmes indices pour tous les utilisateurs. Bien sûr, l'acheteur j doit refuser de continuer l'échange après la révélation de m_h objets.

Ces bits vont servir d'entrée aux fonctions d'accusation des codes de Tardos en cas de découverte d'une copie forgée en possession d'un utilisateur non autorisé.

5.4 D'autres détails d'implémentation

Nous avons proposé une solution permettant à l'acheteur de générer son identifiant tout en respectant les conditions liées aux codes de Tardos. Nous avons aussi trouvé une solution pour la transmission du *halfword* de l'acheteur au vendeur.

Cependant, il reste à résoudre le problème de l'insertion.

5.4.1 Watermarking

Tout d'abord, nous avons besoin d'un algorithme qui permet au vendeur d'envoyer à l'acheteur une copie de la vidéo tatouée avec son *fingerprint*, en sachant que le vendeur ne connaît pas ce *fingerprint*. Il existe des protocoles *buyer-seller* permettant de tatouer une séquence $\mathbf{X}_j$ dans un contenu c_o sans révéler $\mathbf{X}_j$ au vendeur ni c_o à l'acheteur. Ils sont basés sur du chiffrement homomorphique et fonctionnent avec quelques implémentations spécifiques aux techniques d'étalement de spectre [Kur10] ou aux techniques de *Quantization Index Modulation* watermarking [DBPP09]. Le lecteur peut consulter [Kur10, DBPP09] pour plus de détails. Ces méthodes peuvent être adaptées pour embarquer des codes de Tardos. Nous faisons une description rapide de l'adaptation de [DBPP09].

Nous adaptons le schéma de tatouage proposé dans cet article comme suit : Soit $\mathbf{c}_i^{(0)} = (c_{i,1}^{(0)}, \ldots, c_{i,Q}^{(0)})$ les Q composants quantifiés (pixels, coefficients DCT, etc) du i-ième bloc de contenu marqué avec le symbole '0' (resp. $\mathbf{c}_i^{(1)}$ pour le symbole '1'). On note $\mathbf{d}_i = \mathbf{c}_i^{(1)} - \mathbf{c}_i^{(0)}$. On utilise, comme dans [DBPP09, Sect. 5], un système de chiffrement homomorphique additif et probabiliste $E[.]$ comme le cryptosystème de Paillier, par exemple.

- L'acheteur j génère une paire de clefs publique/privée (pk_j, sk_j) et envoie
$$(E_{pk_j}[X_{j,1}], \ldots, E_{pk_j}[X_{j,m}])$$
au vendeur.
- Le vendeur envoie en retour les chiffrés
$$E_{pk_j}[c_{i,\ell}^{(0)}].E_{pk_j}[X_{j,i}]^{d_{i,\ell}}, \forall (i,\ell) \in [m] \times [Q].$$
- Grâce à la propriété homomorphique du système, l'acheteur j déchiffre ces valeurs avec sk_j dans $c_{i,\ell}^{(0)}$ si $X_{j,i} = 0$, $c_{i,\ell}^{(1)}$ si $X_{j,i} = 1$.

Puisque $X_{j,i}$ est constant pour les Q composants du i-ième bloc, une grande part de bande passante et de puissance de calcul est sauvée avec une représentation en signal composite telle que détaillée dans [DBPP09, Sect. 3.2.2].

Une étape cruciale dans ce protocole *buyer-seller* est de prouver au vendeur que ce que l'acheteur lui envoie est bien le chiffrement de bits, et plus encore des bits correspondants au *fingerprint* de l'acheteur. Pour faire une telle preuve, il faut habituellement des protocoles zero-knowledge très compliqués [Kur10, DBPP09]. Nous pensons pouvoir éviter cette complexité en tirant avantage du fait que le vendeur connaît déjà certains bits du *fingerprint* $\mathbf{X}_j$, c'est-à-dire ceux qui appartiennent au *halfword* (voir Sect. 5.3.2), et les acheteurs ne connaissent pas les indices de ces bits. A cet effet, pour m_v indices du *halfword* pris au hasard, le vendeur demande à l'acheteur j de révéler l'engagement. Pour un tel indice i_v, l'acheteur j révèle la valeur aléatoire r_{i_v} du chiffrement probabiliste de Pallier (avec les notations de [DBPP09]). Le vendeur calcule $g^{X_{j,i_v}} h^{r_{i_v}} \mod N$ et vérifie la correspondance avec le i_v-ième chiffré, lequel vaut $E_{pk_j}[X_{j,i}]$ d'après l'acheteur.

L'inconvénient de ce simple schéma de vérification est que l'acheteur révèle m_v indices du *halfword*. Ceci peut conduire à des attaques par collusion élaborées. Par exemple, l'acheteur j, qui est un *colluder*, peut essayer d'appliquer $Y_{i_v} \neq X_{j,i_v}$ en essayant de créer une copie pirate . Une plus grande discussion la-dessus sort un peu du sujet.

Cette approche introduit aussi une menace pour l'acheteur. Un vendeur malhonnête peut demander d'ouvrir les engagements de bits qui ne font pas partie du *halfword* afin de découvrir des bits qu'il n'est pas censé connaître. Pour cette raison, le vendeur doit envoyer $\texttt{alea}_{k(i_v,j),i_v}$ comme défini dans la Sec. 5.3.2 pour prouver à l'acheteur j que la vérification a bien lieu sur un bit du *halfword*.

5.4.2 Accusation

L'accusation est simple et similaire à d'autres protocoles de *fingerprinting*. Une agence de dépistage est en charge de la capture d'un faux. Le vendeur extrait le message

embarqué et les extraits de séquence $\mathbf{Y}$ à partir du contenu piraté. Le vendeur calcule les *halfscores* en appliquant l'équation 5.1 uniquement sur les *halfwords*. Il dresse une liste de suspects, soit les utilisateurs dont le score est au-dessus d'un seuil, soit les utilisateurs avec les plus grands scores. Cette liste de suspects ne peut pas être prise pour argent comptant, car elle a été élaborée par le vendeur seul. Cependant, cette étape rend l'accusation plus pratique parce que l'étape finale est en effet assez chargée en termes de communication et de calcul.

Cette liste est donnée à un juge qui commence par vérifier les calculs des *halfscores*. Si il trouve des valeurs différentes, le vendeur est black-listé. Sinon, le juge calcule les scores sur le *fingerprint* complet. Pour ce faire, le juge a besoin de connaître le vecteur secret $\mathbf{p}$: il demande au vendeur les clefs $\{K_{k,i}\}$, $\forall(k,i) \in [N] \times [m]$ afin de pouvoir accéder à tous les objets $\{O_{k,i}\}$ qui sont dans le WORM, et ensuite aux valeurs exactes de $(p_1, \ldots, p_m)$. D'un autre côté, le juge demande aux utilisateurs j suspectés les clefs $\boxed{\text{choix}}$ qui cachent le choix ($R_{j,i}$ dans le cas du CES, $a_{i,j}$ pour l'utilisation du protocole de Chu) ce qui révèle quel objet a été pris par l'acheteur j au i-ième tour du protocole de génération du *fingerprint* Sec. 5.3.1 et d'où $X_{j,i}$. Le juge accuse finalement l'utilisateur dont le score calculé sur le *fingerprint* complet est au-dessus d'un seuil donné (qui dépend de la probabilité de fausse alarme).

5.4.3 Sécurité

Afin de contrer une éventuelle attaque d'un vendeur malhonnête, nous avons un peu modifié l'utilisation du code de Tardos. Nous nous demandons ici si ces modifications n'ont pas affecté les propriétés du code de Tardos, et si la multiplication d'échanges de données ne donne pas lieu à des attaques inexistantes jusque là.

5.4.3.1 Vendeur honnête

Supposons pour commencer que le vendeur est honnête et notons c la taille de la coalition. Une grande capacité de traçage sûr sur les *halfwords* est nécessaire afin d'éviter les fausses alarmes. Pour ceci, comme montré par G. Tardos, $m_h = O(c^2 \log n\epsilon^{-1})$, où ϵ est la probabilité de suspecter des utilisateurs innocents. De plus, des attaques par collusion efficaces sont contrées si il y a des valeurs de p_i secrètes telles que $p_i < c^{-1}$ ou $p_i > 1 - c^{-1}$([FPF09b]). La longueur des listes utilisées pour la quantification des valeurs p_i doit être assez grande pour permettre l'existence de telles valeurs de p_i. Par conséquent, N doit être suffisamment grand, autour d'une centaine, pour résister à des attaques menées par des coalitions de taille de l'ordre de dix. La figure 5.4 illustre le comportement des scores d'accusation en fonction de la longueur N des listes utilisées.

Pendant la génération du *fingerprint* présentée à la Sec. 5.3.1, la permutation $\pi_j(.)$ permet de s'assurer que l'acheteur j choisit un bit '1' de façon aléatoire avec la probabilité $p_i = L_i/N$ comme il est demandé pour le code de Tardos. En particulier, un *colluder* ne peut pas bénéficier des découvertes faites par ses complices.

On analyse maintenant pourquoi les *colluders* tricheraient pendant le tatouage de leur version du document décrit dans la Sec. 5.4.1. En comparant leurs *fingerprints*,

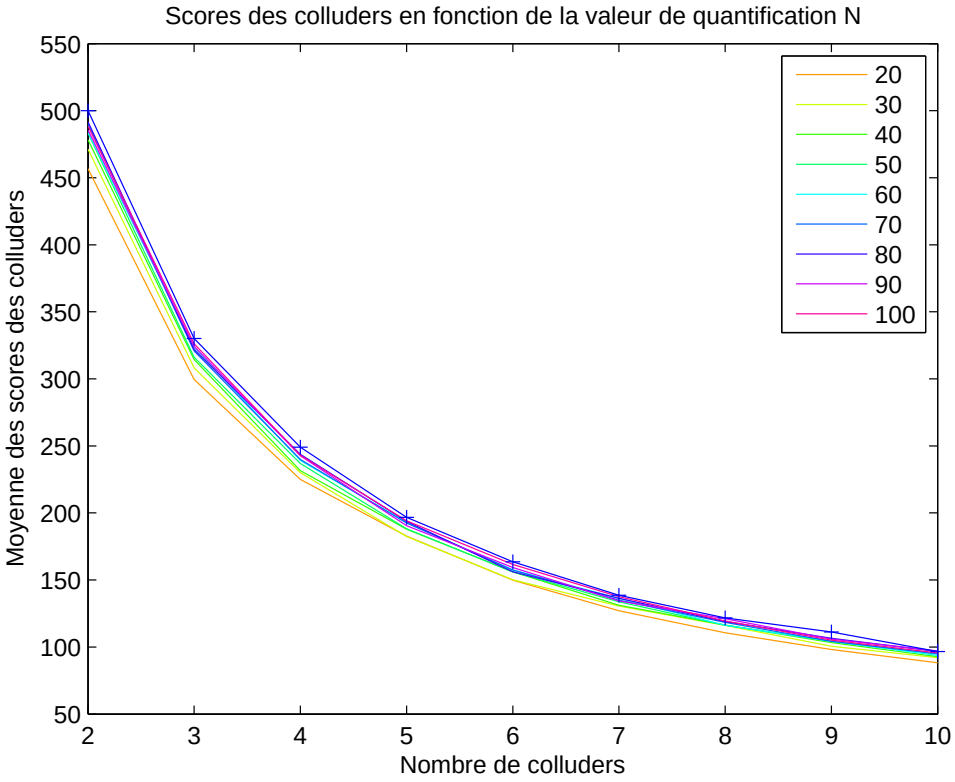


FIGURE 5.4 – Impact de la quantification sur le calcul des scores. N de 10 à 100. $m = 1500$. Avec des croix apparaissent les valeurs concernant le code de Tardos non quantifié.

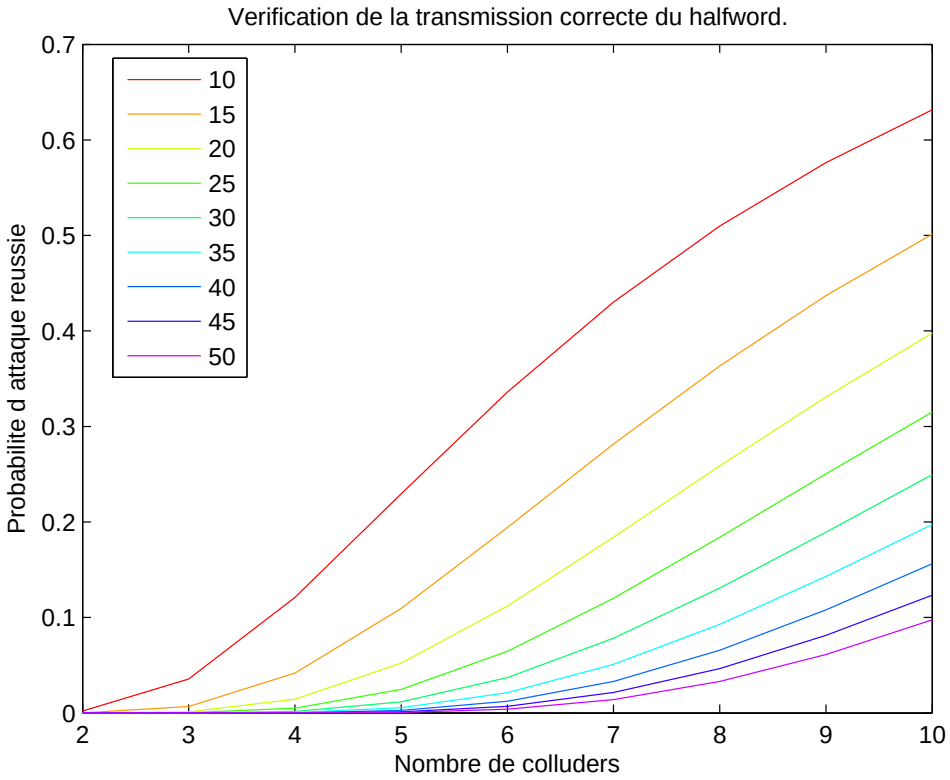


FIGURE 5.5 – Attaque pendant l'insertion : probabilité d'attaque menée à bien en fonction du nombre de vérifications m_v demandées. m_v de 10 à 50. $m = 3000$ et $m_h = 1500$.

ils détectent des indices auxquels est affecté le même symbole, que ce soit '0' ou '1'. Comme expliqué dans l'introduction, les *colluders* ne sont pas capables de changer ces bits dans le *fingerprint* inséré sauf s'ils trichent durant la phase d'insertion : si leurs bits de *fingerprint* à l'index i sont tous égaux à '1', l'un des *colluders* peut chercher à prétendre qu'il a un '0' à cette position. S'ils arrivent à faire de même pour toutes les positions, ils seront capables de créer une copie piratée avec un *fingerprint* qui vaut '0' partout, par exemple.

Combien de fois est-ce que les *colluders* ont besoin de tricher ? Avec la probabilité p_i^c (resp. $(1-p_i)^c$), ils ont tous le bit '1' (resp. '0') à l' index i. Donc, il y a en moyenne $m_c(c) = m\int_t^{1-t}(p^c+(1-p)^c)f(p)dp$ indices qui correspondent à cette situation. Le vendeur demande un bit de vérification avec la probabilité m_v/m_h. La probabilité de mener une attaque réussie pour une collusion de taille c est par conséquent $(1-m_v/m_h)^{m_c(c)}$. On peut voir sur la figure 5.5 les résultats de nos simulations numériques. Elles montrent que m_v n'a pas besoin d'être supérieur à 50 bits pour une longueur de code habituelle et une taille de coalition en dessous de dix. Ainsi, m_v est bien en-dessous de m_h.

5.4.3.2 Vendeur malhonnête

Supposons maintenant que le vendeur est malhonnête. Le fait que les m listes $\mathcal{C}_i$, $\forall i \in [m]$ soient publiques et non modifiables empêche le vendeur de les modifier pour un acheteur ciblé dans l'idée de le confondre plus tard. De plus, cela augmente la suspicion du juge si la distribution empirique des p_i ne correspond pas à la pdf f. Pourtant, des biais peuvent être introduits dans les probabilités pour les symboles des *fingerprints* des *colluders* seulement si il y a une coalition entre eux et le vendeur malhonnête. Par exemple, le vendeur peut choisir une permutation en sélectionnant le premier item (resp. le dernier) dans la liste $\mathcal{D}_{j,i}$ pour qu'un *colluder* complice soit sûr de prendre le symbole '1' (resp. '0'). Ceci annule la propriété traçante du code, mais ne permet pas au vendeur de piéger un innocent. Premièrement, il est garanti que le vecteur $\mathbf{p}$ utilisé dans l'équation 5.1 est celui qui a généré le code. Deuxièmement, le vendeur et son complice *colluder* continuent d'ignorer une part signifiante des *fingerprints* des acheteurs innocents. A cette fin, $m - m_h$ doit aussi être de l'ordre de $O(c^2 \log n\epsilon^{-1})$. Si on a cela, le juge est capable de prendre une décision solide en s'appuyant sur la partie *halfword* du *fingerprint*. Par conséquence, $m \approx 2m_h$, notre protocole a doublé la longueur habituelle, qui est toujours en $O(c^2 \log n\epsilon^{-1})$.

5.5 Conclusion

Les codes de Tardos sont actuellement les meilleurs codes anti-collusion. Cependant, des protocoles antérieurs de *fingerprinting* asymétrique ne peuvent pas leur être appliqués directement pour 2 raisons principales. Tout d'abord, l'acheteur doit générer son *fingerprint*, mais en fonction du vecteur $\mathbf{p}$, qui est gardé secret par le vendeur. Deuxièmement, le vecteur $\mathbf{p}$ utilisé dans le processus d'accusation doit être le même que celui qui a servi à générer les *fingerprints*.

Nous avons proposé un nouveau protocole de *fingerprinting* asymétrique dédié aux codes de Tardos. Nous croyons que c'est le premier protocole de ce type, et qu'il est efficace en pratique.

La construction des *fingerprints* et leur intégration dans des pièces de travail ne nécessitent pas la présence d'un tiers de confiance.

Notez, cependant, que pendant la phase accusation, un tiers de confiance est nécessaire comme dans n'importe quel schéma de *fingerprinting* asymétrique que nous connaissons. Des travaux supplémentaires sont nécessaires pour déterminer si un tel tiers peut être éliminé. En particulier, nous prévoyons qu'une certaine forme de calcul multi-partie peut être appliquée.

D'autres extensions de ce travail comprennent l'étude des codes de Tardos non binaires, et l'extension vers un protocole de *fingerprinting* anonyme.

Les protocoles anonymes sont la prolongation des protocoles asymétriques et ce sont ces protocoles qui font l'objet des recherches les plus récentes. L'absence de structure déterministe des codes de Tardos rend impossible leur intégration dans un protocole existant. Il est donc nécessaire de créer un protocole original. Nous avons commencé à réfléchir à ce travail et n'avons pas trouvé de solution. Néanmoins, nous pensons qu'il

est possible de construire un tel protocole.

Chapitre 6

Conclusion

Le *fingerprinting* est un outil permettant de tracer la redistribution illégale de documents multimédia. Le contexte du *fingerprinting* est expliqué dans le chapitre 1. Il consiste à marquer chaque copie du document que l'on souhaite tracer avec un identifiant (ou *fingerprint*) qui correspond à la personne à qui est distribuée la copie. On suppose que parmi ces utilisateurs, certains sont malhonnêtes et vont tenter de redistribuer leur copie du document sur des canaux non autorisés. Si ces utilisateurs malhonnêtes distribuent leur copie en l'état, ils s'exposent à être immédiatement reconnus. On suppose alors que ces utilisateurs malhonnêtes vont chercher à détruire la marque (donc attaquer le *watermarking*), ou vont se mettre à plusieurs pour créer une copie contenant une fausse marque. Cette attaque à plusieurs utilisateurs est dite attaque par collusion et les utilisateurs malhonnêtes qui la mettent en oeuvre sont appelés *colluders*. C'est à ce type d'attaque que nous nous sommes intéressés dans cette thèse. Pour contrer les attaques par collusion, les identifiants ne peuvent être pris au hasard. Ils sont pris parmi les mots d'un code dit code anti-collusion.

L'insertion de l'identifiant se fait au moyen d'un technique de marquage (ou *watermarking*). Nous avons présenté dans le chapitre 2 les composants principaux d'un schéma de *fingerprinting*, à savoir la technique de marquage ainsi que le code qui définit les identifiants. Il existe des schémas de *fingerprinting* qui n'utilisent pas de code, c'est alors la technique de marquage qui assure aussi la traçabilité. Dans le chapitre 2, nous décrivons aussi les environnements du schéma de *fingerprinting*. Cet environnement peut être symétrique (c'est le cas habituel de la littérature), asymétrique ou bien asymétrique anonyme. La différence principale de ces environnements concerne 'qui fait quoi'. Alors que dans un environnement symétrique c'est le distributeur de contenu (appelé vendeur) qui procède à quasiment toutes les actions ; dans un environnement asymétrique certaines tâches sont effectuées par l'acheteur, d'autres effectuées au cours d'échanges 2-partie entre acheteur et vendeur.

Les codes anti-collusion sont décrits en détail dans le chapitre 3. Notre travail porte sur une famille de codes anti-collusion particulière, les codes de Tardos. Les codes de Tardos sont les premiers codes de *fingerprinting* à atteindre la borne de longueur minimale et sont donc actuellement les plus performants. Ce sont des codes probabilistes,

ce qui les rend très différents des autres codes anti-collusion étudiés auparavant, ces codes étant pour la plupart des codes correcteurs d'erreur. Nous avons travaillé sur les codes de Tardos en étudiant leur comportement, en essayant de les améliorer, et en les incorporant dans des structures complexes de *fingerprinting* asymétrique.

Les codes de Tardos sont décrits en détail dans le chapitre 3 concernant les codes anti-collusion. Dans le chapitre 4, nous nous intéressons aux fonctions d'accusation des codes de Tardos binaires. Les fonctions d'accusations données par Tardos ont la particularité d'avoir le même comportement quelle que soit la stratégie utilisée par les *colluders* pour construire la copie pirate. Nous avons amélioré la phase d'accusation en modifiant ces fonctions afin de tenir compte de la stratégie utilisée par les *colluders*. Pour utiliser au mieux ces fonctions, il est donc nécessaire d' obtenir des informations sur cette stratégie, c'est pourquoi nous avons aussi construit une estimation de la stratégie à la volée en utilisant l'algorithme EM (*Expectation-Maximization*).

Les résultats obtenus grâce à ces nouvelles fonctions sont très intéressants, puisqu'en cas de bonne estimation de la stratégie, le nombre de *colluders* détecté augmente considérablement. La faiblesse du schéma est dans l'estimation du nombre de *colluders*. En effet, l'algorithme EM fonctionne bien lorsque les distributions à séparer sont gaussiennes et avec un grand nombre d'échantillons. Dans un scénario de *fingerprinting*, le nombre de *colluders* considéré est rarement très élevé, on considère régulièrement un nombre de *colluders* inférieur à 10. Avec un aussi petit nombre d'éléments, l'algorithme EM est très instable dans son estimation.

Nous avons commencé le même travail d'analyse des fonctions d'accusation dans le cas q-aire, ces travaux sont présentés en annexe A. La construction de fonctions d'accusation dans le cas q-aire afin de tenir compte de la stratégie est un travail que nous n'avons pas réussi à mener à bien, c'est une perspective de travail intéressante.

Dans le chapitre 5, nous avons construit un schéma de *fingerprinting* dans un environnement asymétrique en utilisant les codes de Tardos. Après avoir étudié les schémas existants, nous avons remarqué qu'aucune de ces constructions ne permettait l'utilisation des codes de Tardos, notemmment à cause de l' absence de structure déterministe de ces codes. Nous avons construit un tel schéma en nous intéressant particulièrement à l'aspect pratique. C'est pourquoi nous avons entièrement spécifié le schéma, en donnant des solutions pour toutes les étapes. Notre schéma se situe dans un modèle où le vendeur et l'acheteur sont malhonnêtes. La difficulté de la construction du schéma asymétrique avec un code de Tardos consiste à gérer le tirage des bits de l'identifiant (effectué par l'acheteur) selon une probabilité connue du vendeur seul. Nous avons choisi pour solution à ce problème l'utilisation d'une primitive cryptographique appelée *Oblivious Transfer*. Cette primitive autorise Bob à effectuer un tirage aléatoire dans une liste détenue par Alice de telle sorte qu'Alice ne sait pas ce que Bob a tiré et Bob ne sait rien sur les éléments qu'il n'a pas tirés.

La suite logique de ce travail consiste à contruire un schéma asymétrique anonyme. Là encore, l'absence de structure déterministe des codes de Tardos pose un problème que nous n'avons pas réussi à résoudre pour l'instant.

Annexe A

Etude du code de Tardos q-aire

Nous nous intéressons maintenant aux fonctions d'accusation du code de Tardos (le code est décrit en détail dans la partie 3.3). Dans [FGC08b], les auteurs cherchent à savoir si les fonctions d'accusation données par Tardos, puis Skoric dans le cas symétrique binaire, sont les meilleures qu'on puisse utiliser. Partant de l'hypothèse que la distribution des scores calculés grâce à ces fonctions doit être la même quelle que soit la stratégie utilisée par les *colluders*, ils montrent que les fonctions données dans [SKC08] sont bien les meilleures (voire les seules) qu'on peut avoir. Dans ce chapitre, nous regardons ce qu'il en est dans le cas q-aire, en se basant sur le code proposé par Skoric pour le cas q-aire

A.1 Cas q-aire du code de Tardos

Cette construction a été présentée dans [SKC08]. On considère un alphabet A de taille q : $A = (a_1, a_2, \ldots, a_q)$. Au lieu d'avoir une valeur de p, on a un vecteur $\mathbf{p} = (p_{a_1}, p_{a_2}, \ldots, p_{a_q})$, p_{a_i} représente la probabilité d'avoir le symbole a_i de l'alphabet de taille q. f est alors une fonction à $q-1$ variables, car $\sum_{i=1}^{q} p_{a_i} = 1$.
On conserve les notations utilisées dans le cas binaire : X représente le mot associé à un utilisateur, Y désigne la copie forgée. m est la longueur des mots, il y a n utilisateurs. c désigne la taille de la collusion.

Pour calculer le score, on utilise deux fonctions g_1 et g_0 comme dans le cas binaire : Cas où les deux valeurs X_i et Y_i sont égales :

$$U(X_i, Y_i, \mathbf{p}) = g_1(p_{Y_i}) \qquad si \quad X_i = Y_i$$

Cas où les valeurs X_i et Y_i sont différentes :

$$U(X_i, Y_i, \mathbf{p}) = g_0(p_{Y_i}) \qquad si \quad X_i \neq Y_i$$

Nous considérons un fonctionnement proche du code binaire, mais on pourrait imaginer que les fonctions g_0 et g_1 dépendent aussi du caractère sur lequel on est tombé.

On peut modéliser les fonctions d'accusation sous forme d'une matrice G de taille $q \times q$ construite avec les g_i. Sur la diagonale apparaissent les fonctions utilisées lorsque les valeurs X_i et Y_i sont égales, sur les autres positions, les fonctions utilisées lorsque les valeurs sont différentes : $G_{jk} = U(k, j, p_j)$.

Exemple 12. Dans le cas binaire, la matrice G est : $\begin{pmatrix} g_1(1-p) & g_0(1-p) \\ g_0(p) & g_1(p) \end{pmatrix}$ avec le vecteur $\mathbf{p}$: $\begin{pmatrix} 1-p & p \end{pmatrix}$

Dans le cas q-aire, $G = \begin{pmatrix} g_1(p_1) & g_0(p_1) & g_0(p_1) & \dots & g_0(p_1) \\ g_0(p_2) & g_1(p_2) & g_0(p_2) & \dots & g_0(p_2) \\ \vdots & g_0(p_3) & g_1(p_3) & \dots & \vdots \\ g_0(p_q) & g_0(p_q) & g_0(p_q) & \dots & g_1(p_q) \end{pmatrix}$

A.1.1 Contraintes

On se fixe les mêmes contraintes que dans le cas binaire :
- La moyenne des scores des innocents vaut 0.
- La variance des scores des innocents vaut 1.
- Les scores de deux innocents sont indépendants (covariance = 0).
- L'espérance et la variance du score d'un utilisateur (coupable ou non) sont indépendantes de la stratégie utilisée par les pirates.

A.1.2 Cas où l'utilisateur est innocent

Pour une position fixée, on modélise la stratégie des pirates par un vecteur $\mathbf{r} = (r_1, r_2, \dots, r_q)$, avec $r_j = \mathbb{P}(Y = a_j)$. On a donc $\sum_{i=1}^{q} r_i = 1$.
L'utilisateur X étant innocent, les variables X et Y sont indépendantes. On veut que le résultat soit indépendant de la tactique des pirates.

A.1.2.1 Covariance

D'après le théorème central limite, lorsque m est assez grand, le score d'un innocent suit une distribution gaussienne. Nous allons maintenant regarder la dépendance entre deux scores d'utilisateurs innocents.

$$Cov(S_j, S_k) = m\mathbb{E}(S_j S_k) = m\mathbb{E}(U(X_j, Y, P)U(X_k, Y, P)$$

$$\begin{aligned} Cov(S_j, S_k) &= m\mathbb{E}_p(\sum_{i=1}^{q} r_i p_i^2 g_1^2(p_i) + \sum_{i=1}^{q} r_i (1-p_i)^2 g_0^2(p_i) + 2\sum_{i=1}^{q} r_i p_i (1-p_i) g_1(p_i) g_0(p_i)) \\ &= m\mathbb{E}_p(\sum_{i=1}^{q} r_i [p_i g_1(p_i) + (1-p_i) g_0(p_i)]^2) \end{aligned}$$

Les coefficients r_i sont non tous nuls, donc pour que cette somme soit nulle, on doit avoir :

$$\forall i, [p_i g_1(p_i) + (1-p_i) g_0(p_i)]^2 = 0$$

ce qui implique

$$\forall i, p_i g_1(p_i) + (1-p_i) g_0(p_i) = 0$$

soit

$$(G\mathbf{p}^T) = 0 \tag{A.1}$$

Cela nous donne une relation entre g_0 et g_1 pour tous les p_i :

$$-(1-p_i) g_0(p_i) = p_i g_1(p_i) \tag{A.2}$$

A.1.2.2 Espérance

On peut modéliser l'espérance par le produit suivant : $\mu = m\mathbb{E}_p(\mathbf{r}G\mathbf{p}^T)$.

On a montré que pour avoir les scores des innocents indépendants, il faut avoir $(G\mathbf{p}^T) = 0$ (voir A.1), on a donc $\mu = 0$.

A.1.2.3 Variance

On appelle G_2 la matrice constituée par les coefficients de G au carré :

$$G_2 = \begin{pmatrix} g_1^2(p_1) & g_0^2(p_1) & \dots & g_0^2(p_1) \\ g_0^2(p_2) & g_1^2(p_2) & \dots & g_0^2(p_2) \\ \vdots & & \vdots & \\ g_0^2(p_q) & g_0^2(p_q) & \dots & g_1^2(p_q) \end{pmatrix}$$

On a $Var(S_j) = m\mathbb{E}_p(S_j^2) = m\mathbb{E}_p(\mathbf{r}G_2\mathbf{p}^T)$.

Le vecteur $\mathbf{r}$ peut se décomposer en somme de deux vecteurs $\mathbf{r} = \frac{1}{q}\mathbb{1} + \tilde{r}$, avec $\tilde{r}$ tel que $\tilde{r}\mathbb{1}^T = 0$. Cela vient du fait que $\sum_{i=1}^{q} r_i = 1$.

On a alors : $\mathbb{E}_p(\mathbf{r}G^2\mathbf{p}^T) = \mathbb{E}_p((\frac{1}{q}\mathbb{1} + \tilde{r})G^2\mathbf{p}^T) = \mathbb{E}_p(\frac{1}{q}\mathbb{1}G^2\mathbf{p}^T + \tilde{r}G^2\mathbf{p}^T)$, avec $\mathbb{1} = (1, 1, \dots, 1)$.
La partie $\frac{1}{q}\mathbb{1}G^2\mathbf{p}$ ne dépend pas de r.

Afin que la variance ne dépende pas de la stratégie des pirates, on doit avoir : $G^2\mathbf{p} = k_3\mathbb{1}$, soit

$$\begin{pmatrix} g_1^2(p_1) & g_0^2(p_1) & \dots & g_0^2(p_1) \\ g_0^2(p_2) & g_1^2(p_2) & \dots & g_0^2(p_2) \\ \vdots & & \vdots & \\ g_0^2(p_q) & g_0^2(p_q) & \dots & g_1^2(p_q) \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_q \end{pmatrix} = constante$$

Ce qui donne :

$$p_1 g_1^2(p_1) + (1-p_1) g_0^2(p_1) = p_2 g_1^2(p_2) + (1-p_2) g_0^2(p_2) = \ldots = p_q g_1^2(p_q) + (1-p_q) g_0^2(p_q)$$

En utilisant la relation trouvée précédemment, on a :

$$\begin{aligned} -(1-p_i) g_0(p_i) &= p_i g_1(p_i) \\ (1-p_i)^2 g_0^2(p_i) &= p_i^2 g_1^2(p_i) \\ (1-p_i) g_0^2(p_i) &= \frac{p_i^2 g_1^2(p_i)}{1-p_i} \end{aligned}$$

On remplace dans $F(p_i) = p_i g_1^2(p_i) + (1-p_i) g_0^2(p_i)$

$$\begin{aligned} F(p_i) &= p_i g_1^2(p_i) + \frac{p_i^2 g_1^2(p_i)}{1-p_i} \\ &= g_1^2(p_i)(p_i + \frac{p_i^2}{1-p_i}) \\ F(p_i) &= g_1^2(p_i)(\frac{p_i}{1-p_i}) \end{aligned}$$

$$g_1^2(p_1)\frac{p_1}{1-p_1} = g_1^2(p_2)\frac{p_2}{1-p_2} = \ldots = g_1^2(p_q)\frac{p_q}{1-p_q}$$

pour tous les q-uplets $\mathbf{p}$ tels que $\sum p_i = 1$

Lemme 1. *Soit* $F(p) = g_1^2(p)\frac{p}{1-p}$
La fonction F *est constante sur [0,1] si* $q \geq 2$.

preuve : par l'absurde, supposons que F n'est pas constante. Alors, il existe deux valeurs distinctes a et b telles que $F(a) \neq F(b)$. q est supérieur à 2, donc pour tout q-uplets $(a, a_1, \ldots, a_q)$ et $(b, b_1, \ldots, b_q)$ tels que :
$\sum a_i = 1$ et $\sum b_i = 1$
On a :
$F(a) = F(a_1) = \ldots = F(a_q)$
$F(b) = F(b_1) = \ldots = F(b_q)$
On peut alors toujours prendre l'un des b_i à la place d'un des a_j, ce qui nous donne $F(a) = F(b)$.
La fonction F est donc constante, $g_1^2(p)\frac{p}{1-p} = K$
ce qui donne $g_1^2(p) = K\frac{1-p}{p}$

$$\boxed{g_1(p) = \sqrt{K\frac{1-p}{p}}}$$

On retrouve alors une fonction g_1 de la forme de celle donnée par Tardos.

Grâce à la relation trouvée avec l'espérance, on peut alors en déduire g_0.

$$\begin{aligned} g_0(p) &= -\frac{p}{1-p} g_1(p) \\ &= -\frac{p}{1-p}\sqrt{K\frac{1-p}{p}} \\ &= -\sqrt{K\frac{p}{1-p}} \end{aligned}$$

$\boxed{g_0(p) = -\sqrt{K\frac{p}{1-p}}}$

Si on pose $Var(S_j) = m$, alors on a $K = 1$ et on retrouve exactement les fonctions proposées par Skoric et al [SKC08].

A.2 Conclusion

Nous avons fait une étude de la structure du code de Tardos en mode q-aire, et montré que sous certaines conditions on retrouvait bien les fonctions utilisées dans [SKC08]. Avec les conditions d'indépendance des scores des innocents entre eux, et d'indépendance par rapport à la stratégie utilisée par les *colluders*, les fonctions données dans [SKC08] sont les seules qui respectent les hypothèses. Cependant, il faudrait étudier de même le rôle joué par la fonction f de distribution des probabilités.

Annexe B

Rappels sur l'algorithme Expectation-Maximization

On utilise souvent l'algorithme d'Espérance-maximisation pour la classification de données, l'apprentissage automatique, ou la vision artificielle. Nous utilisons cet algorithme lors de l'étape S2 (voir 4.2) afin de séparer les scores appartenant à des *colluders* de ceux appartenant à des innocents.

L'algorithme d'espérance-maximisation comporte :

- une étape d'évaluation de l'espérance (E), où l'on calcule l'espérance de la vraisemblance en tenant compte des dernières variables observées,
- une étape de maximisation (M), où l'on estime le maximum de vraisemblance des paramètres en maximisant la vraisemblance trouvée à l'étape E.

On utilise ensuite les paramètres trouvés en M comme point de départ d'une nouvelle phase E, et on recommence.

On a un échantillon de variables $(x_1, \ldots, x_n)$ qui sont issues de g différents groupes (dans notre cas, 2 groupes).

On sait que chacun de ces groupes G_k suit une loi de paramètres θ_k. Dans notre cas, on sait que le premier groupe des innocents suit une loi gaussienne $f(x, \theta_1)$ de paramètres $\theta_1 = (\mu_1 = 0, var_1 = m)$, tandis que le deuxième groupe des *colluders* suit une loi gaussienne $f(x, \theta_2)$ de paramètres $\theta_2 = (\mu_2 = m * \frac{2}{\pi c}, var_2 = m * (1 - (\frac{\mu_2}{m})^2)$.

- Les proportions d'éléments de chaque groupe dans l'échantillon sont notées $(\pi_1, \ldots, \pi_g)$.
- On note $\Phi = (\pi_1, \ldots, \pi_g, \theta_1, \ldots, \theta_g)$ le paramètre du mélange.
- On note z_{ik} la grandeur qui vaut 1 si l'individu x_i appartient au groupe G_k et 0 sinon. C'est ce vecteur que l'on cherche à connaître pour séparer les données.
- On note t_{ik} l'estimation des z_{ik} au cours des itérations de l'algorithme.

Les deux étapes Expectation et Maximization sont itérées jusqu'à la convergence.

- * Etape E : calcul de t_{ik} par la règle d'inversion de Bayes :

$$t_{ik} = \frac{\pi_k^{(c)} f(x_i, \theta_k^{(c)})}{\sum_{\ell=1}^{g} \pi_\ell^{(c)} f(x_i, \theta_\ell^{(c)})}$$

- * Etape M : détermination de Φ maximisant

$$Q\left(\Phi, \Phi^{(c)}\right) = \sum_{i=1}^{n} \sum_{k=1}^{g} t_{ik} \log\left(\pi_k f(x_i, \theta_k)\right)$$

L'avantage de cette méthode est qu'on peut séparer le problème en g problèmes élémentaires qui sont, en général relativement simples. Dans tous les cas, les proportions optimales sont données par

$$\pi_k = \frac{1}{n} \sum_{i=1}^{n} t_{ik}$$

L'estimation des θ dépend par ailleurs de la fonction de probabilité f choisie. Dans le cas normal, il s'agit des moyennes μ_k et des matrices de variance-covariance σ_k. Les estimateurs optimaux sont alors donnés par

$$\mu_k = \frac{\sum_{i=1}^{n} t_{ik} x_i}{\sum_{i=1}^{n} t_{ik}}$$

$$\Sigma_k = \frac{\sum_{i=1}^{n} t_{ik} (x_i - \mu_k)(x_i - \mu_k)'}{\sum_{i=1}^{n} t_{ik}}$$

Avec M' la matrice transposée de M et en supposant que les μ_k sont des vecteurs colonnes.

Annexe C

Rappels sur les codes linéaires

C.1 Codes correcteurs d'erreurs

Définition intuitive

Les codes correcteurs d'erreurs servent habituellement à compenser la perte d'information lors de transmission de données par un canal peu fiable. Ils procèdent par ajout de redondance dans l'information transmise. Une illustration en est l'utilisation par les aviateurs de "Alpha" pour A, "Bravo" pour B, ... La personne qui entend "Alpha Bravo Charlie" sur une mauvaise transmission aura plus de chance de retrouver "ABC" que si on émet "ABC" tel quel.
Pour la transmission de données numériques, on utilise le même principe : par exemple pour transmettre "101", on peut envoyer "111100001111". Un compromis est à trouver entre la sécurité de transmission recherchée et la taille du message codé. Pour optimiser ce problème, les codes utilisés ne sont pas simplement à répétition, mais utilisent des relations plus complexes entre les bits du message (par exemple un bit de parité). Pour plus de détails sur les codes correcteurs d'erreurs, voir [PH92], [HP03].

C.1.1 Vocabulaire et définitions

Définition 17. Un **code** (m, N, d) est un ensemble de N mots de longueur m, d est la distance minimale du code.

Définition 18. Support d'un mot : si $x = (x_1, \ldots, x_n)$, alors $supp(x) = \{i | x_i \neq 0\}$.
Cela correspond aux positions du mot pour lesquelles les lettres sont différentes de 0.
Le support de 10110 est $\{1, 3, 4\}$.

Définition 19. Poids de Hamming : $w(x) = |supp(x)|$
C'est le cardinal du support du mot, soit le nombre de lettres non nulles du mot.
Le poids de Hamming de 10110 est 3.

Définition 20. Distance de Hamming : $d(x, y) = w(x - y)$.
La distance entre 10110 et 01011 est 4.

Définition 21. Distance minimale d'un code C : La *distance minimale* d du code est la distance minimale entre deux mots quelconques du code (pour la distance de Hamming).

$$d = Min_{x,y \in C}(d(x,y))$$

Définition 22. Capacité de correction : Si un code C a pour distance minimale d, alors il permet de corriger toute erreur de poids au plus $t = \lfloor (d-1)/2 \rfloor$. Le nombre t s'appelle la capacité de correction du code.

Un *encodage* est une relation ϕ entre le message et un mot du code. Cette relation est bien sûr injective, un mot du code ne doit correspondre qu'à un seul message. On note c l'encodage du message x.

C.2 Codes linéaires

C.2.1 Définitions

Les codes linéaires sont des sous-espaces-vectoriels, ce qui permet de les aborder du point de vue de l'algèbre linéaire.
Les paramètres d'un code linéaire sont alors $[m, k, d]$: longueur, dimension et distance minimale.

La **fonction d'encodage** correspond à un produit matriciel entre le message et la matrice génératrice du code.

Exemple 13. Exemple 1 : Soit le code $[5, 2, 3]$ de matrice génératrice

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Pour avoir tous les mots d'un code, il suffit de multiplier tous les mots appartenant à l'espace des messages par la matrice génératrice.

$$c = xG$$

Une matrice génératrice à k lignes correspond donc à un code ayant 2^k éléments. On voit ainsi qu'une matrice génératrice d'un code correcteur a forcément plus de colonnes que de lignes.

Pour notre exemple, on a donc :

message	encodage
00	00000
01	01011
10	10110
11	11101

La *matrice de parité* (ou matrice de contrôle) du code linéaire C est la matrice H telle que si c est un mot du code, alors $H.c^T = 0$. On a alors $HG^T = GH^T = 0$.

Exemple 14. En reprenant le code donné précédemment, la matrice de parité correspondante est

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Définition 23. Un *décodeur* est une application qui étant donné un mot y de l'ensemble d'arrivée, retrouve le mot du code le plus proche au sens de la distance de Hamming.

Définition 24. Tous les codes linéaires vérifient la **borne de Singleton** :

$$m - k \geq d - 1$$

Définition 25. Un code est appelé **MDS** (Maximum distance separable) s'il atteint la borne de Singleton $k + d = m + 1$.

C.2.2 Décodage par syndrome

Définition 26. Translatés d'un code : La relation $xRy, x - y \in C$ est une relation d'équivalence. Les *cosets* ou translatés d'un code ou classes latérales sont les classes d'équivalence. Ce sont les ensembles de la forme $x + C$.

Définition 27. Le **syndrome** d'un mot est la valeur $H.x^t$, où H est la matrice de parité du code. Les mots du code ont donc un syndrome égal à zéro.

- l'émetteur envoie c un mot du code ;
- le canal étant perturbé, le récepteur reçoit $y = c + e$ où e représente le syndrome (l'erreur) ;
- si l'erreur n'est pas trop importante (soit $e < \frac{d-1}{2}$ avec d la distance minimale du code), on peut la corriger, c'est-à-dire retrouver le mot c.

1. Dans tout translaté $y + C$, on choisit un élément de plus petit poids, $e(y)$ que l'on met dans une table indexée par le syndrome de y. Remarquons que $e(y+c) = e(y)$ pour tout $c \in C$ 2. Si y est le mot reçu, on calcule son syndrome $S(y)$, et on lit $e(y)$ dans la table. 3. l'algorithme retourne $\gamma(y) = \tilde{c} = y - e(y)$.

Exemple 15. En reprenant le code donné précédemment,
un tableau standard de ce code est C.1 :
Par exemple pour décoder le mot $y = 10111$: On calcule $H^t.y = 001$
Dans la table des translatés, on repère le translaté *leader* correspondant à ce syndrome, c'est $c(y) = 00001$.
On calcule $y - c(y) = 10110$, c'est le mot de code le plus proche de y. Le mot envoyé était donc 10.

C.3 Exemples de codes

C.3.1 Codes de Hamming

Le code de Hamming binaire H_m est un code linéaire de paramètres $(n = 2^m - 1, 2^m - m - 1, 3)$, les colonnes de sa matrice de parité sont les $2^m - 1$ vecteurs distincts

message :	00	10	01	11	syndrome
code :	00000	10110	01011	11101	000
coset :	10000	00110	11011	01101	110
coset :	01000	11110	00011	10101	011
coset :	00100	10010	01111	11001	100
coset :	00010	10100	01001	11111	010
coset :	00001	10111	01010	11100	001
coset :	00101	10011	01110	11000	101
coset :	01100	11010	00111	10001	111
	coset leaders				

TABLE C.1 – Table des translatés

non nuls de F_2^m (c'est bien une matrice $m \times n$) :

$$H_m = \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & 1 & \dots & 1 \\ 0 & 1 & 1 & 0 & \dots & 1 \\ 1 & 0 & 1 & 0 & \dots & 1 \end{pmatrix}$$

Le code H_m est l'ensemble des combinaisons linéaires nulles des colonnes de H_m.

Exemple 16. $m = 3$, le code de Hamming H_3 a pour paramètres $[7, 4, 3]$, et pour matrice de parité :

$$H_3 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

C.3.2 Codes cycliques

Définition 28. Un code linéaire C de longueur m est dit **cyclique** si chaque permutation circulaire d'un mot de code est un mot de code.

$$w = (w_0, w_1, \dots, w_{m-1}) \in C \Rightarrow w^\pi = (w_{m-1}, w_0, \dots, w_{m-2}) \in C$$

La structure de ces codes est celle d'un idéal dans l'anneau des polynômes de degré au plus $m - 1$. Pour créer le parallèle entre les deux ensembles, chaque vecteur de longueur m sera associé au polynôme de degré au plus $m - 1$ dont les coefficients sont les éléments du vecteur.

$$w = (w_0, w_1, \dots, w_{m-1}) \in C \Leftrightarrow w(X) = w_0 + w_1 X + \dots + w_{m-1} X^{m-1} \in C(X)$$

$C(X)$ est l'ensemble des polynômes associés aux mots de C.

Un code linéaire C est cyclique si et seulement si $C(X)$ est un idéal de $\mathbb{F}_q[X]/(X^m - 1)$. $\mathbb{F}_q[X]/(X^m - 1)$ est un anneau principal, donc tous ses idéaux sont principaux (par définition), ce qui signifie que $C(X)$ est engendré par un polynôme unitaire $g(X)$.

On peut donc définir la **matrice génératrice** du code C à partir du polynôme générateur unitaire $g(X)$.

Théorème 5. *Pour chaque diviseur unitaire $g(X)$ de $X^m - 1$, on associe un code cyclique de longeur m et de dimension $k = m - r - 1$ (r est le degré de $g(X)$) dont la matrice génératrice est :*

$$\begin{pmatrix} g_0 & g_1 & \dots & g_{r-1} & 1 & 0 & 0 & \dots & 0 \\ 0 & g_0 & g_1 & \dots & g_{r-1} & 1 & 0 & \dots & 0 \\ 0 & 0 & g_0 & g_1 & \dots & g_{r-1} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & g_0 & g_1 & \dots & g_{r-1} & 1 \end{pmatrix}$$

Pour définir la matrice de parité, on considère le polynôme $h(X) = (X^m - 1)/g(X)$. $h(X)$ est le générateur de l'idéal de dimension $k = n - r$, c'est un polynôme unitaire de degré k.

$$H = \begin{pmatrix} 1 & h_{k-1} & h_{k-2} & \dots & h_1 & h_0 & 0 & 0 & \dots & 0 \\ 0 & 1 & h_{k-1} & h_{k-2} & \dots & h_1 & h_0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & \dots & 1 & h_{k-1} & h_{k-2} & \dots & \dots & h_1 & h_0 \end{pmatrix}$$

Exemple 17. Le polynôme $g(X) = X^3 + X + 1$ qui divise $X^7 - 1$ sur $\mathbb{F}_2$ engendre un code cyclique dont voici la matrice génératrice :

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

C.3.3 Codes BCH

On considère ici les racines d'un polynôme $g(X)$ dont les coefficients sont dans un corps fini $\mathbb{F}_q$, avec q qui est la puissance d'un nombre premier p.

Soit α un élément primitif de $\mathbb{F}_{q^m}$.

$q' = q^m - 1$. Les racines de $g(X)$ sont des puissances d'une racine n-ième de l'unité β qui vaut $\beta = \alpha^{q'/m}$.

Théorème 6. *Soit $g(X)$ le polynôme générateur d'un code cyclique $[m, k]$ $(k = m - r)etr = degr(g(X))$. Si $\beta^{i_1}, \beta^{i_2}, \dots, \beta^{i_r}$ sont les r racines de $g(X)$, alors une matrice*

de parité du code est :

$$H = \begin{pmatrix} 1 & \beta^{i_1} & (\beta^{i_1})^2 & \dots & (\beta^{i_1})^{m-1} \\ 1 & \beta^{i_2} & (\beta^{i_2})^2 & \dots & (\beta^{i_2})^{m-1} \\ 1 & \beta^{i_3} & (\beta^{i_3})^2 & \dots & (\beta^{i_3})^{m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \beta^{i_r} & (\beta^{i_r})^2 & \dots & (\beta^{i_r})^{m-1} \end{pmatrix}$$

Théorème 7. *Soit C un code cyclique $[m, k]$ et β une racine de l'unité. Si son polynôme générateur $g(X)$ admet $\delta - 1$ puissances consécutives de β, $\beta^b, \beta^{b+1}, \dots, \beta^{b+\delta-2}$ comme racines, alors la distance minimale du code est au moins δ. δ est appelée la distance construite.*

On définit le code BCH primitif au sens strict sur $\mathbb{F}_q$ de distance construite δ comme l'ensemble des polynômes de $F_q[X]/(X^m - 1)$ s'annulant en $\alpha, \alpha^2, ..., \alpha^{\delta-1}$. Ce code a pour matrice de parité :

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{m-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(m-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{\delta-1} & \alpha^{2(\delta-1)} & \dots & \alpha^{(m-1)(\delta-1)} \end{pmatrix}$$

C.3.4 Codes de Reed-Solomon

Un code de Reed-Solomon est un code BCH défini sur $K = F_{p^l}$ et de longueur $m = p^l - 1$. Le code $RS(d)$ est alors le code ayant pour ensemble de définition $\{1, ..., d-1\}$. Son polynôme unitaire générateur de plus petit degré est :

$$g(X) = (X - \alpha^a)(X - \alpha^{a+1}) \dots (X - \alpha^{a+d-2})$$

Le degré de g est $d - 1$.

Théorème 8. *Les codes de Reed-Solomon sont des codes MDS. Le code $RS(d)$ a pour distance minimale $d = m - k + 1$ qui est la plus grande distance minimale pour m et k fixés.*

Exemple 18. Dans $\mathbb{F}_7$, $\alpha = 3$ est un élément primitif. Le code de Reed-Solomon de paramètres $m = 6$, $d = 4$, $k = 3$ déterminé par $\alpha = 3$ et $a = 1$ a pour polynôme générateur :

$$g(X) = (X - 3)(X - 2)(X - 6) = X^3 + 3X^2 + X + 6$$

Sa matrice génératrice est :

$$G = \begin{pmatrix} 6 & 1 & 3 & 1 & 0 & 0 \\ 0 & 6 & 1 & 3 & 1 & 0 \\ 0 & 0 & 6 & 1 & 3 & 1 \end{pmatrix}$$

et sa matrice de parité est :

$$H = \begin{pmatrix} 1 & 3 & 2 & 6 & 4 & 5 \\ 1 & 2 & 4 & 1 & 2 & 4 \\ 1 & 6 & 1 & 6 & 1 & 6 \end{pmatrix}$$

Annexe D

Primitives cryptographiques

Dans cette annexe, nous présentons quelques-unes des principales primitives cryptographiques. Nous ne les avons pas toutes utilisées dans nos contributions, mais certains protocoles présentés dans l'état de l'art les utilisent, c'est pourquoi nous trouvons intéressant de les décrire.

D.1 Sécurité des chiffrements, quelques balises

Pour étudier la sécurité d'une primitive cryptographique, on commence par lister les types d'adversaires et d'attaques auxquels elle doit faire face.

D.1.1 Sécurité inconditionnelle

La sécurité des schémas de chiffrement a été formalisée pour la première fois par Shannon, qui a introduit la notion de *secret parfait/ sécurité inconditionnelle*, qui caractérise les schémas de chiffrement pour lesquels la connaissance d'un message chiffré ne donne aucune information que ce soit sur le clair correspondant ou sur la clef. Il a prouvé que le One-Time Pad est parfaitement sûr sous certaines conditions : Véritable hasard dans la génération du flot de chiffrement, de la même longueur que le texte clair, et à usage unique. En fait, aucun autre schéma, qu'il soit symétrique ou asymétrique, a été prouvé comme ayant une sécurité inconditionnnelle. Mais, si on oublie le chiffrement One-Time-Pad, la sécurité de n'importe quelle primitive cryptographique est évaluée en observant le pouvoir de calcul de l'attaquant.

D.1.2 Modèle de l'oracle aléatoire

Dans le cas des schémas asymétriques, on ne peut pas se baser sur leur structure mathématique pour estimer leur niveau de sécurité avec une méthode formelle. Ils sont basés sur des problèmes mathématiques identifiés comme étant très difficiles à résoudre, mais faciles à résoudre pour la personne qui connaît les clefs. C'est-à-dire que c'est facile pour le possesseur des clefs de calculer sa clef secrète, mais personne d'autre ne doit être capable de le faire, puisque la connaissance de la clef publique ne doit rien révéler sur la

clef privée. Par réduction, on peut comparer le niveau de sécurité de ces schémas avec la difficulté de résoudre ces problèmes mathématiques (factorisation de grands entiers, ou calcul de logarithme discret dans un groupe fini de grands taille) qui sont connus pour leurs difficulté. En procédant de cette façon, on obtient une estimation du niveau de sécurité, qui s'avère parfois optimiste. cette estimation peut s'avérer insuffisante pour certaines raisons. Premièrement, il peut y avoir d'autres façons de casser le système que de résoudre le problème mathématique correspondant. Ensuite, la plupart des preuves de sécurité sont calculées dans un modèle idéal appelé *random oracle model*, dans lequel les primitives concernées, *e.g.*, fonctions de hachage, sont considérées comme ayant véritablement des valeurs prises au hasard. Ce modèle a permis l'étude du niveau de sécurité pour des chiffrements asymétriques numériques. Des travaux récents ont montré que nous sommes maintenant capables de calculer des preuves dans un modèle plus réaliste appelé *standard model*. Un grand nombre d'articles compare ces deux modèles, en discutant les différences entre les deux. En parallèle de cette estimation formelle du niveau de sécurité, une estimation empirique est calculée dans chaque cas, et de nouveaux schémas symétriques et asymétriques sont évalués en tenant compte des attaques publiées.

D.1.3 Sécurité sémantique

En parallèle, Goldwasser a introduit la notion de *sécurité semantique*, formalisant ce qui peut être une sécurité sous-optimale mais réaliste, comparée avec la *sécurité inconditionelle*. Un cryptosystème est dit *semantically secure* si il est impossible pour un adversaire limité dans sa puissance de calcul de déduire des informations significantes sur un message clair (plaintext) alors qu'il ne possède que le chiffré (et la clef publique de chiffrement dans le cas d'un système asymétrique). Il a été prouvé que cette notion est équivalente à la *polynomial security*. Cette équivalence amène à affirmer qu'un système de chiffrement asymétrique doit être probabiliste pour atteindre la sécurité sémantique (c'est une condition nécessaire, pas suffisante). Une conséquence est que pour qu'un schéma atteigne la sécurité sémantique, l'espace occupé par le chiffré est plus grand que celui du clair, ce qui mène souvent à ce que les chiffrés soient plus longs que les clairs correspondants. L'*expansion* est définie comme le quotient de ces deux longueurs.

D.1.4 Niveaux de sécurité

La méthode pour l'évaluation de la sécurité a été établie par Shannon en 1949 : tous les messages considérés sont chiffrés avec la même clef – c'est-à-dire, pour le même receveur – et le but de l'attaquant est de tirer un avantage de toutes ces observations pour découvrir la clef secrète/privée. En général, pour évaluer la capacité d'attaque de l'attaquant, on distingue plusieurs contextes : *ciphertext-only attacks* (dans lesquels l'attaquant n'a accès qu'à un certain nombre de chiffrés), *known-plaintext attacks* (dans lesquels l'attaquant a accès à des paires clairs/chiffrés qui se correspondent), *chosen-plaintext attacks* (même conditions que précédemment, mais l'attaquantbut peut choisir les textes clairs et avoir les chiffrés correspondants) et *chosen-ciphertext attacks* (l'atta-

quant a accès à un oracle de déchiffrement, qui se conduit comme une boîte noire, qui prend en entrée un texte chiffré et retourne le clair correspondant en sortie). Le premier contexte est le plus fréquent dans la vie réelle, c'est le pire cas pour un attaquant. Les autres cas ont l'air difficiles à réaliser, et peuvent arriver lorsque l'attaquant a une position de pouvoir ; il peut, par exemple, avoir volé des textes clairs, ou une machine à chiffrer. Les cas 'choisis' existent dans une version *adaptive*, où l'attaquant attend de recevoir le résultat d'un calcul avant de choisir la prochaine entrée.

Pour les schémas asymétriques, tout ceci amène à la définition des classes de sécurité sémantique suivantes, de la plus faible à la plus forte. le système peut fournir *INDistinguishability against Chosen Plaintext Attack* (IND-CPA), *INDistinguishability against Chosen Ciphertext Attack* (IND-CCA1) ou *INDistinguishability against adaptive Chosen Ciphertext Attack* (IND-CCA2).

Pour des exemples, nous nous référons à [Gol04, vT05, FG07] pour de plus amples détails sur les niveaux de sécurité. Ces niveaux de sécurité concernent la primitive que nous allons utiliser dans des protocoles plus compliqués.

D.2 Signature

D.2.1 Définitions

La signature numérique est l'analogue de la signature manuscrite, que l'on appose par exemple au bas d'un contrat ou sur un chèque. Elle engage la responsabilité du signataire, qui ne peut pas la renier. La signature ne peut pas être imitée (sauf avec une probabilité très faible), et peut être vérifiée par n'importe qui.

Définition 29. Un procédé de signature est la donnée de :
- Un ensemble fini P, appelé l'espace des messages
- Un ensemble fini S, appelé l'espace des signatures
- Un ensemble fini K, appelé l'espace des clefs
- Pour chaque clef $k \in K$, une fonction de signature $s_k : P \to S$ et une fonction de vérification $v_k : (P, S) \to \{oui, non\}$ telles que $v_k(x, s_k(x)) = oui$ pour tout $x \in P$.

Dans la pratique, il existe deux types de schémas de signature.
- Les schémas de signature avec *restauration du message* : dans ce cas, la signature est une donnée dont le traitement permet de retrouver le message et de garantir son authenticité. Le message clair n'a donc pas à être transmis en plus de la signature. Par contre, la signature est nécessairement au moins aussi longue que le message.
- Les schémas de signature *en appendice* : la signature est alors en général une valeur beaucoup plus courte que la donnée qu'elle identifie. Pour la vérifier, il faut disposer de la signature et du message signé (qui ont été obtenus ensemble ou séparement)

D.2.2 Quelques exemples de schémas de signature

D.2.2.1 Signature RSA

Le cryptosystème RSA, décrit par Rivest, Shamir et Adleman en 1977 [RSA78] fonctionne comme suit :
Le destinataire Bob choisit deux entiers N, E tels que

$$N \text{ est le produit de deux grands nombres premiers distincts } p \text{ et } q$$
$$0 \leq E \leq \varphi(N) \text{ et } pgcd(E, \varphi(N)) = 1, \text{ où } \varphi(N) = (p-1)(q-1)$$

D'autre part, il calcule l'entier d tel que

$$0 \leq d \leq \varphi(N)$$

et $Ed \equiv 1(mod\varphi(N))$ Le module N et l'exposant public E sont diffusés, p, q, et l'exposant secret d sont gardés secrets.
Le message à transmettre est converti en éléments de Z_N. Pour chiffrer l'élément $m \in Z_N$, Alice calcule $c = m^E modN$ qu'elle transmet. Bob reçoit c et déchiffre en calculant $c^d \bmod N$.
En effet, le groupe $(Z/NZ)^*$ est d'ordre $\varphi(N)$ ce qui implique :

$$c^d \equiv m^{Ed} \equiv m(modN)$$

Pour la signature, le choix des paramètres p, q, N, E se fait de la même façon que pour chiffrer mais il est fait par l'expéditeur Alice, qui diffuse sa clef publique (N,E). Pour signer un message $m \in Z_N$ qu'elle veut envoyer à Bob, Alice calcule $s = m^d \bmod N$. Dans son envoi à Bob elle fait accompagner m par sa signature s. A la réception, Bob peut contrôler la signature d'Alice en vérifiant que $s^E \bmod N = m$. Si l'égalité est vérifiée, Bob est assuré que le message m provient bien d'Alice et qu'il n'a pas été altéré puisqu'elle est la seule à posséder la clef secrète d nécessaire au calcul de s.

D.2.2.2 Signature El Gamal

La sécurité de ce procédé de signature repose sur le problème du logarithme discret. Le procédé de chiffrement d'El-Gamal a été proposé par Taher El-Gamal en 1984 [ElG85]

L'expéditeur Alice choisit un (grand) nombre premier p ainsi qu'une racine primitive g modulo p. Elle choisit aussi un entier a dans $[1, p-2]$ et calcule $A = g^a \bmod p$. Elle publie les entiers p, g, et A et garde a secret.
Pour signer chaque message $m \in Z_p$, Alice choisit un entier k dans $[1, p-2]$ tel que $pgcd(k, p-1) = 1$ et calcule $K = g^k \bmod p$. Elle calcule aussi $s = (m - aK)k^{-1} \bmod p - 1$ et dans son envoi à Bob, elle fait accompagner le message m par sa signature (K, s).
A la réception, Bob s'assure que la signature est bien celle d'Alice en vérifiant que $1 \leq K \leq p-1$ et $A^K K^s \equiv g^m$ modulo p.
en effet :

$$A^K K^s \equiv (g^a)^K (g^k)^{m-aK} k^{-1} \equiv g^{aK} g^{m-aK} \equiv g^m (modp)$$

D.3 Engagement

D.3.1 Principe

Cette primitive peut être décrite de la façon suivante :

La partie qui s'engage écrit une information (par exemple un bit au hasard) sur un bulletin et place le bulletin dans une boîte opaque. Après que le protocole ait fait intervenir une autre partie, la boîte est ouverte et son contenu dévoilé. Cette primitive a deux propriétés :

1. L'information est dissimulée (*concealing property*) dans un premier temps, jusqu'au moment où elle est révélée.
2. L'information ne peut pas être modifiée (*binding property*) entre le début de l'engagement (le moment où on ferme la boîte) et sa révélation (le moment où on ouvre la boîte).

D.3.2 Exemple : engagement d'un bit reposant sur les résidus quadratiques

Soit $N = pq$ un entier RSA. On note :

$$Z_N^+ = \{x \in Z_N | (x/N) = 1\}$$
$$Q = \left\{x^2 mod N | x \in Z_N, pgcd(x, N) = 1\right\} \subseteq Z_N^+$$

Alice génère un élément m de Z_N^+/Q qu'elle rend public. Pour engager un bit $b \in \{0, 1\}$, Alice transmet $e = m^b r^2 \mod N$ où r est un élément aléatoire de Z_N^*. Pour dévoiler b à Bob, elle lui indiquera les valeurs de b et de r et celui-ci pourra vérifier que $e \equiv m^b r^2$ modulo N.

D.4 Protocoles sans divulgation d'information (Zero-knowledge)

D.4.1 Protocole de Fiat-Shamir

Ce protocole à divulgation nulle de connaissance a été proposé en 1987 par Amos Fiat et Adi Shamir [FS87]. Ce protocole utilise un entier $n = pq$ tel ceux utilisés pour RSA fourni par un tiers de confiance. Le secret détenu par Alice est la racine carrée a modulo n d'un entier public $A \in [1, n-2]$. Alice est identifiée auprès de Bob au moyen des trois échanges suivants :

1. Données publiques Le prouveur choisit un nombre S et calcule $V = S^2 mod(N)$. Il publie V
2. Authentification : Le vérifieur demande au prouveur de s'authentifier : Le prouveur a choisi un nombre aléatoire R.
3. Phase d'enchère : Le prouveur calcule $X = R^2 \mod (N)$. Il envoie X au vérifieur

4. Phase de défi : Le vérifieur met le prouveur au défi : Il choisit un nombre aléatoire binaire D et l'envoie au prouveur
5. Phase de preuve : Le prouveur répond en envoyant Y au vérifieur Si $D = 0$ $Y = R$ Si $D = 1$ $Y = R * S \bmod (N)$
6. Phase de vérification : Le vérifieur calcule Y^2. Il doit trouver : Si $D = 0$ $Y^2 = X$, Si $D = 1$ $Y^2 = X * V \bmod (N)$

ANALYSE : Si le fraudeur connaissait, dès la phase 1, la question posée en 3, il pourrait toujours tromper le valideur : Si $D = 0$, il choisit R quelconque et calcule $X = R^2$. Si $D = 1$, il choisit un nombre K arbitraire et pose $X = K^2 * V \bmod (N)$ $Y = K * V \bmod (N)$. Ceci vérifie donc $Y^2 = X * V \bmod (N)$ (mais il ne connaît pas R, la racine de X) Il est donc indispensable de procéder dans cet ordre. Donc ne connaissant pas la question le fraudeur doit a priori choisir entre les deux stratégies et a donc une chance sur deux d'être capable de répondre.

D.4.2 Protocole de Schnorr

Ce protocole de signature numérique a été proposé par C P Schnorr en 1990. [Sch90] Paramètres :

1. Un nombre premier p
2. Un deuxième nombre premier q diviseur de $p - 1$
3. un générateur g du groupe cyclique d'ordre q de Z_p^*
4. un entier $x < q$. $y = g^x$ modolo p
5. Les valeurs (p, q, g, y) sont publiques.
6. La valeur x est le secret.

étant donné y, prouver que l'on connaît x

Prouveur		Vérifieur
tire au hasard $r \bmod q$		
$t = g^r \bmod p$	$\longrightarrow$	
	$\longleftarrow$	tire au hasard $e < q$
$s = r - ex \bmod q$	$\longrightarrow$	
		Vérifie que $t = g^s y^e \bmod p$

D.5 *Oblivious Transfer*

D.5.1 Description

Un *Oblivious Transfer* (transfert inconscient, en français) ou OT est un protocole cryptographique qui permet à Bob de choisir k éléments dans une liste de n détenue par Alice sous les conditions suivantes :

1. Bob obtient des éléments qui sont bien dans la liste concernée ;
2. Bob n'obtient aucune information sur les éléments qu'il n'a pas choisis ;

3. Alice ne sait pas quels sont les éléments qui ont été choisis.

Il existe de très nombreux protocoles d'*OT* dans la littérature. Nous décrivons 4 protocoles adaptifs qui atteignent les meilleurs niveaux de sécurité étudiés jusqu'à maintenant.

D.5.2 Quelques exemples de protocoles OT

D.5.2.1 Protocole $OT^N_{k\times 1}$ proposé par Chu et Tzeng

Ce protocole a été proposé en 2005 par Chu et Tzeng [CT05].

- Les paramètres du système sont (g, H_1, H_2, G_q). Les H_i sont des fonctions de hachage.
- S possède les messages $m_1, m_2, \ldots, m_N$
- R veut choisir les messages : $\sigma_1, \sigma_2, \ldots, \sigma_k$

Phase d'engagement :

1. S calcule $c_i = m_i \oplus H_2(w_i^x)$ et $y = g^x$. $x \in_R \mathbb{Z}_q^*$, $w_i = H_1(i)$
2. S envoie à R $y, c_1, c_2, \ldots, c_N$

Phase de transfert :

Pour chaque σ_i, faire les actions suivantes :

1. R choisit aléatoirement $a_j \in \mathbb{Z}_q^*$ et calcule $w_{\sigma_j} = H_1(\sigma_j)$, $A_j = w_{\sigma_j} g^{a_j}$.
2. R envoie A_j à S.
3. S renvoie $D_j = (A_j)^x$.
4. R calcule $K_j = D_j / y^{a_j}$ et obtient $m_{\sigma_j} = c_{\sigma_j} \oplus H_2(K_j)$.

D.5.2.2 Camenish 1

Ce protocole décrit dans l'article [CNS07] a été proposé par Camenisch, Neven et Shelat, il est basé sur un schéma de Boneh et Boyen faiblement sûr [BB04].
e est une table admissible bilinéaire $e : \mathbb{G}_1 X \mathbb{G}_1 \to \mathbb{G}_T$, $e(g^a, g^b) = e(g, g)^{ab}$, $e(g, g) \neq 1$.
Le schéma utilise la table bilinéaire comme suit : la clef secrète du signeur est $x \leftarrow \mathbb{Z}_p$, la clef publique correspondante est $y = g^x$. La signature sur un message M est $s = g^{1/(x+M)}$. La vérification est faite en vérifiant que $e(s, y.g^M) = e(g, g)$

Initialisation :

S possède les messages $m_1, m_2, \ldots, m_N$
R veut choisir les messages : $\sigma_1, \sigma_2, \ldots, \sigma_k$

- $g, h \leftarrow \mathbb{G}_1^*$, $H = e(g, h)$
  $x \in \mathbb{Z}_p$; $y = g^x$; $pk = (g, H, y)$
  Pour $i = 1, \ldots, N$ faire
  $A_i = g^{1/(x+i)}$
  $B_i = e(h, A_i).M_i)$
  $C_i = (A_i, B_i)$

$S_0 = (h, pk)$

- S envoie pk, C_i à R.

Phase de transfert :

Pour chaque σ_i, faire les actions suivantes :

1. R choisit aléatoirement $v \in \mathbb{Z}_p$ et calcule $V = (A^v_{\sigma_i}$
2. R envoie V à S.
3. S calcule $W = e(h, V)$ et le renvoie.
4. R obtient $M = B_{\sigma_i}/(W^{1/v})$.

D.5.2.3 Camenish 2

Ce deuxième protocole $OTk \times 1^N$ proposé par J. Camenish dans [CNS07], utilise une signature aveugle (*Blind Signature*) BS = (Kg, Sign, User, Vf). Ces 4 algorithmes sont utilisés comme suit : Kg$\rightarrow (pk, sk)$ est l'algorithme de génération de clé, Sign(sk) utilisé avec l'algorithme User(pk, m) permet de construire un protocole de signature du message m à l'issu duquel User retourne une signature s, l'algorithme de vérification Vf(pk, m, s) retourne 1 si la signature est valide, 0 sinon.

- H est un oracle aléatoire.
- S possède les messages $m_1, m_2, \ldots, m_N$
- R veut choisir les messages : $\sigma_1, \sigma_2, \ldots, \sigma_k$
- $(pk, sk \leftarrow Kg$
  Pour $i = 1, \ldots, N$ faire
  $s_i = Sign(sk, i)$
  $C_i = (A_i, B_i)$
  $S_0 = H(i, s_i) \oplus M_i$

- S envoie pk, C_i à R. Cet ensemble est noté R_0.
- $S_0 = sk$.

Phase de transfert :

Pour chaque σ_i, faire les actions suivantes :

1. S lance le protocole $Sign(sk)$ et envoie
2. R calcule $s_i = User(pk, \sigma_i)$
3. R vérifie si $Vf(pk, \sigma_i, s_i) = 0$ alors $M_{\sigma_i} = vide$
   sinon $M_{\sigma_i} = C_{\sigma_i} \oplus H(i, s_i)$

La sécurité de ce protocole est en fonction de la sécurité du schéma de signature aveugle utilisé.

D.5.2.4 Green 1

Cet algorithme est proposé par Matthew Green et Susan Hohenberger dans [GH07]
$H : M \longrightarrow \{0,1\}^n$

(Setup, Extract, Encrypt, Decrypt) est IND-sID-CPA-secure.

- S possède les messages $m_1, m_2, \ldots, m_N$
- R veut choisir les messages : $\sigma_1, \sigma_2, \ldots, \sigma_k$

Initialisation :

1. S choisit $(params, msk) \longleftarrow \mathsf{Setup}\ (1^\kappa, c(\kappa))$.
2. S choisit aléatoirement $(W_1, \ldots, W_N) \in M$, et pour $j = 1, \ldots, N$ ensembles :
   - $A_j \longleftarrow \mathsf{Encrypt}\ (params, j, W_j)$
   - $B_j \longleftarrow H(W_j) \oplus M_j$
   - $C_j = (A_j, B_j)$
3. S lance $PoK\{msk : (params, msk) \in \mathsf{Setup}\ (1^\kappa, c(\kappa)\}$
4. S envoie $(params, C_1, \ldots, C_N)$ à R.
5. R vérifie si $\mathsf{isValid}\ (params, i, C_i)$

Phase de transfert :

Pendant le i^{th} transfert, lancer Blindextract sur la valeur σ_i pour que le receveur obtienne sk_{σ_i}.

1. Si Blindextract échoue, alors R place m'_{σ_i} sur $\perp$.
2. sinon R lance $t \longleftarrow \mathsf{decrypt}\ (params, \sigma_i, sk_{\sigma_i}, A_{\sigma_i})$ and set $m'_{\sigma_i} \longleftarrow B_{\sigma_i} \oplus H(t)$.

D.5.2.5 Green 2

Ce protocole est décrit dans l'article [GH08] rédigé par Matthew Green et Susan Hohenberger.

OTGenCRS, OTInitialize, OTRequest, OTRespond, OTComplete)

CRS signifie *Common Reference String*

- OTGenCRS génère crs.
- OTInitialize est exécuté par S, la sortie est un engagement sur les données, T, et une clé secrète pour S sk.
- OTRequest est exécuté par R. Ce protocole génère une requête Q sur la liste T et l'index σ.
- OTRespond Executé par S. La sortie est 'reject' si l'envoyeur ne veut pas répondre à la requête du receveur, sinon, la sortie est (s, δ), où δ est une preuve, et $s = d_1^{x_1}.d_2^{x_2}$ (d_1 et d_2 viennent de Q, x_1 et x_2 font partie de la clé secrète sk).
- OTComplete est exécuté par R. La sortie est un message m.

S est sollicité avec (sid, sender, $\langle M_1, \ldots, M_N \in \{0,1\}^l \rangle$) :

1. S demande F_{CRS} avec (sid, S, R) et reçoit (sid, crs).
2. S calcule $(T, sk) \leftarrow$ OTInitialize (crs, $M_1, \ldots, M_N$), envoie (sid, T) à R et conserve (sid, T, sk).

Lorsque R est sollicité avec (sid,receiver, σ), et que R a reçu auparavant (sid, T) et (sid, crs) :

1. R lance (Q, Q_{priv})$\leftarrow$ OTRequest(crs,T, σ), envoie (sid, Q) à S et conserve (sid, Q_{priv}).

	Initialisation		Transfert	
	rounds	opérations	rounds	opérations
Naor-Pinkas [NP99b] using OT_1^2	1	N exp	$logN$	OT_1^2 + 2 exp
Naor-Pinkas [NP99b] Sum consistent	1	$2\sqrt{N}$ exp	$2t$	$OT_1^{\sqrt{N}}$
Chu [CT05]	1	$N+1$ exp	2	2+1 exp
Camenish2 [CNS07]	1+1PoK	$N+1$ exp	2 + 2 PoK	1 exp
Green2 [GH08]	1	$4+7N$	N	10 exp

TABLE D.1 – Comparaison de divers schémas $OT_{k\times 1}^N$. PoK signifie '*Proof of knowledge*', elle peut être faite en 4 rounds et un coût $O(\kappa)$ de communication (κ est un paramètre de connaissance d'erreur $\in [0,1]$). t est un paramètre supplémentaire ($t = 3$ or 4). Les autres protocoles proposés par Camenish [CNS07] et celui proposé par Green-Hohenberger [GH07] sont sous des formes génériques, leur coût dépend donc des sous-protocoles utilisés dans le protocole principal.

2. S lance R $\leftarrow$ OTRespond(crs,T, sk, Q) et envoie (sid, R) à R.
3. R reçoit (sid, R) de S, et calcule (sid, OtComplete(crs, T, R, Q_{priv})).

Chaque algorithme est détaillé dans [GH08].

Bibliographie

[AGC10] W. Abdul, P. Gaborit et P. Carré : Private anonymous fingerprinting for color images in the wavelet domain. *In SPIE 2010*, page 10.1117/12.838996, 2010.

[BB04] D. Boneh et X. Boyen : Short signatures without random oracles. *Advances in Cryptology-EUROCRYPT 2004*, 3027:56–73, 2004.

[BBK03] A. Barg, GR Blakley et G.A. Kabatiansky : Digital fingerprinting codes : Problem statements, constructions, identification of traitors. *Information Theory, IEEE Transactions on*, 49(4):852–865, 2003.

[BD01] F. Bao et R.H. Deng : Privacy Protection for Transactions of Digital Goods. *In ICICS 2001*, volume 2229 de *LNCS*, pages 202–213. Springer-Verlag, 2001.

[BM97] I. Biehl et B. Meyer : Protocols for collusion-secure asymmandric fingerprinting. *In STACS 97*, pages 399–412. Springer, 1997.

[BMP86] G. Blakley, C. Meadows et G. Purdy : Fingerprinting long forgiving messages. *In Advances in Cryptology, CRYPTO 85 Proceedings*, pages 180–189. Springer, 1986.

[BS95] D. Boneh et J. Shaw : Collusion-secure fingerprinting for digital data. *Advances in Cryptology-CRYPTO 95*, 963:452–465, 1995.

[BS98] D. Boneh et J. Shaw : Collusion-secure fingerprinting for digital data. *IEEE Trans. Inform. Theory*, 1998.

[BS02] D. Boneh et J Shaw : Collusion-secure fingerprinting for digital data. *Information Theory, IEEE Transactions on*, 44(5):1897–1905, 2002.

[BT08] O. Blayer et T. Tassa : Improved versions of tardos'fingerprinting scheme. *Des. Codes cryptography*, 48:79–103, 2008.

[Cam00] J. Camenisch : Efficient anonymous fingerprinting with group signatures. *Advances in Cryptology-ASIACRYPT 2000*, pages 415–428, 2000.

[Can02] R. Canandti : Universally composable security : A new paradigm for cryptographic protocols. *In Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2002.

[CFF05] F. Cayre, C. Fontaine et T. Furon : Watermarking security : Theory and practice. *Signal Processing, IEEE Transactions on*, 53(10):3976–3987, 2005.

[CFF10] A. Charpentier, C. Fontaine et T. Furon : Décodage EM du code de Tardos pour le fingerprinting. *TS. Traitement du signal*, 27(2):127–146, 2010.

[CFFC11] A. Charpentier, C. Fontaine, T. Furon et I Cox : An asymmetric fingerprinting scheme using tardos codes. *IH Information Hiding*, 2011.

[CFNP00] B. Chor, A. Fiat, M. Naor et B. Pinkas : Tracing traitors. *IEEE Trans. Inform. Theory*, 46:893–910, 2000.

[CKLS97] I.J. Cox, J. Kilian, F.T. Leighton et T. Shamoon : Secure spread spectrum watermarking for multimedia. *Image Processing, IEEE Transactions on*, 6(12):1673–1687, 1997.

[CNS07] J. Camenisch, G. Neven et A. Shelat : Simulatable adaptive oblivious transfer. *Advances in Cryptology-EUROCRYPT 2007*, pages 573–590, 2007.

[CT05] C.K. Chu et W.G. Tzeng : Efficient k-out-of-n oblivious transfer schemes with adaptive and non-adaptive queries. *Public Key Cryptography-PKC 2005*, pages 172–183, 2005.

[CXFF09] A. Charpentier, F. Xie, C. Fontaine et T. Furon : Expectation maximization decoding of Tardos probabilistic fingerprinting code (Proceedings Paper). *SPIE*, 2009.

[DBPP09] M. Deng, T. Bianchi, A. Piva et B. Preneel : An efficient Buyer-Seller watermarking protocol based on composite signal representation. *In ACM MM&Sec'09*, pages 9–18, 2009.

[Del10] B. Delyon : Régression, 2010.

[DF02] J. Domingo-Ferrer : Anonymous fingerprinting of electronic information with automatic identification of redistributers. *Electronics Landters*, 34(13): 1303–1304, 2002.

[DFHJ00] J. Domingo-Ferrer et J. Herrera-Joancomartí : Efficient smartcard based anonymous fingerprinting. *In Smart Card. Research and Applications*, pages 221–228. Springer, 2000.

[ElG85] T. ElGamal : A public key cryptosystem and a signature scheme based on discrete logarithms. *In Advances in Cryptology*, pages 10–18. Springer, 1985.

[FFG06] C. Fontaine, T. Furon et F. Galand : Etat de l'art sur le fingerprinting. 2006.

[FG07] C. Fontaine et F. Galand : A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007:15, 2007.

[FGC08a] T. Furon, A. Guyader et F. Cérou : Experimental assessment of the reliability for watermarking and fingerprinting schemes. *EURASIP Journal on Inf. Security*, 2008.

[FGC08b] T. Furon, A. Guyader et F. Cerou : On the design and optimization of Tardos probabilistic fingerprinting codes. *In IH 2008*, 2008.

[FPF09a] T. Furon et L. Pérez-Freire : EM decoding of Tardos traitor tracing codes. *In Proceedings of the 11th ACM workshop on Multimedia and security*, pages 99–106. ACM, 2009.

[FPF09b] T. Furon et L. Pérez-Freire : Worst case attack against binary probabilistic traitor tracing codes. *In IEEE WIFS 2009*, pages 46–50, 2009.

[FS87] A. Fiat et A. Shamir : How to prove yourself : Practical solutions to identification and signature problems. *In Advances in Cryptology. Crypto 86*, pages 186–194. Springer, 1987.

[Fur09] T. Furon : Le traçage de traîtres. *In SSTIC 2009*, 2009.

[GH07] M. Green et S. Hohenberger : Blind identity-based encryption and simulatable oblivious transfer. *In Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security*, pages 265–282. Springer-Verlag, 2007.

[GH08] M. Green et S. Hohenberger : Universally composable adaptive oblivious transfer. *Advances in Cryptology-ASIACRYPT 2008*, pages 179–197, 2008.

[Gol04] O. Goldreich : *Foundations of cryptography : Basic applications*. Cambridge Univ Pr, 2004.

[GS98] V. Guruswami et M. Sudan : Improved decoding of Reed-Solomon and algebraic-geomandric codes. *In focs*, page 28. Published by the IEEE Computer Sociandy, 1998.

[HC05] H.F. Huang et C.C. Chang : A new design for efficient t-out-n oblivious transfer scheme. 2005.

[HP03] W.C. Huffman et V. Pless : *Fundamentals of error-correcting codes*. Cambridge Univ Pr, 2003.

[HVLLT98] H.D.L. Hollmann, J.H. Van Lint, J.P. Linnartz et L.M.G.M. Tolhuizen : On Codes with the Identifiable Parent Property . *Journal of Combinatorial Theory, Series A*, 82(2):121–133, 1998.

[HW05a] S. He et M. Wu : Improving collusion resistance of error correcting code based multimedia fingerprinting. *In Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 2, pages ii–1029. IEEE, 2005.

[HW05b] S. He et M. Wu : Performance study on multimedia fingerprinting employing traceability codes. *Digital Watermarking*, pages 84–96, 2005.

[HW06] S. He et M. Wu : Joint coding and embedding techniques for MultimediaFingerprinting. *Information Forensics and Security, IEEE Transactions on*, 1(2):231–247, 2006.

[Kur10] M. Kuribayashi : On the Implementation of Spread Spectrum Fingerprinting in Asymmandric Cryptographic Protocol. *EURASIP Journal on Inf. Security*, 2010.

[Lin08] Y. Lindell : Efficient fully-simulatable oblivious transfer. *CT-RSA 2008*, 4964:52–70, 2008.

[NP99a] M. Naor et B. Pinkas : Oblivious transfer and polynomial evaluation. *In Proceedings of the thirty-first annual ACM symposium on Theory of computing*, page 254. ACM, 1999.

[NP99b] M. Naor et B. Pinkas : Oblivious transfer with adaptive queries. *In Advances in Cryptology-CRYPTO 99*, pages 791–791. Springer, 1999.

[NP05] M. Naor et B. Pinkas : Computationally secure oblivious transfer. *Journal of Cryptology*, 18(1):1–35, 2005.

[OB09] A. Oprea et K. D. Bowers : Authentic Time-Stamps for Archival Storage. *In ESORICS 2009*, volume 5789 de *LNCS*, pages 136–151. Springer-Verlag, 2009.

[OK04] W. Ogata et K. Kurosawa : Oblivious keyword search. *Journal of Complexity*, 20(2-3):356–371, 2004.

[OU98] T. Okamoto et S. Uchiyama : A new public-key cryptosystem as secure as factoring. *In EUROCRYPT 98*, volume 1403 de *LNCS*, pages 308–318, 1998.

[Pai99] P. Paillier : Public key cryptosystems based on degree residuosity classes. *In EUROCRYPT 99*, volume 1592 de *LNCS*, pages 223–238. Springer-Verlag, 1999.

[PH92] A. Poli et L. Huguand : *Error correcting codes : Theory and applications.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1992.

[PS96] B. Pfitzmann et M. Schunter : Asymmetric fingerprinting. *In EUROCRYPT 96*, volume 1070 de *LNCS*, pages 84–95. Springer-Verlag, 1996.

[PS99] B. Pfitzmann et A.R. Sadeghi : Coin-based Anonymous fingerprinting. *In EUROCRYPT 99*, volume 1592 de *LNCS*, pages 150–164. Springer-Verlag, 1999.

[PSS03] C. Peikert, A. Shelat et A. Smith : Lower bounds for collusion-secure fingerprinting. pages 472–478, 2003.

[PW97a] B. Pfitzmann et M. Waidner : Anonymous fingerprinting. *In EUROCRYPT 97*, volume 1233 de *LNCS*, pages 88–102. Springer-Verlag, 1997.

[PW97b] B. Pfitzmann et M. Waidner : Asymmetric fingerprinting for larger collusions. *In Proceedings of the 4th ACM conference on Computer and communications security*, pages 151–160. ACM, 1997.

[Rab81] M. Rabin : How to exchange secrets by oblivious transfer. Rapport technique, Harvard Aiken Computation Laboratory, 1981.

[RSA78] R. Rivest, A. Shamir et L. Adleman : A method for obtaining digital signatures and public-key cryptosystems. volume 21, pages 120–126. ACM, 1978.

[Sch90] C. SCHNORR : Efficient identification and signatures for smart cards. *In Advances in Cryptology-CRYPTO 89 Proceedings*, pages 239–252. Springer, 1990.

[Sch04] H.G. SCHAATHUN : Binary collusion-secure codes : comparison and improvements. Rapport technique, 2004.

[Sch06] H.G. SCHAATHUN : The boneh-shaw fingerprinting scheme is better than we thought. *Information Forensics and Security, IEEE Transactions on*, 1(2):248–255, 2006.

[Sch08] H.G. SCHAATHUN : On error-correcting fingerprinting codes for use with watermarking. *Multimedia Systems*, 13(5):331–344, 2008.

[SKC08] B. SKORIC, S. KATZENBEISSER et M. CELIK : Symmandric Tardos fingerprinting codes for arbitrary alphaband sizes. *Designs, Codes and Cryptography*, 46(2):137–166, 2008.

[SSW01] J.R. STADDON, D.R. STINSON et R. WEI : Combinatorial properties of frameproof and traceability codes. *IEEE Trans.Inform. Theory*, 47:1042–1049, 2001.

[Tar03] G. TARDOS : Optimal probabilistic fingerprint codes. *In STOC 2003*, pages 116–125. ACM, 2003.

[vT05] H.C.A. van TILBORG : *Encyclopedia of cryptography and security*. Springer Verlag, 2005.

[WWZ+03] Z.J. WANG, M. WU, H. ZHAO, K.J.R. LIU et W. TRAPPE : Resistance of orthogonal Gaussian fingerprints to collusion attacks. *In icme*, pages 617–620. IEEE, 2003.

[WZW03] Q.H. WU, J.H. ZHANG et Y.M. WANG : Practical t-out-n oblivious transfer and its applications. *Information and Communications Security*, pages 226–237, 2003.

[ZWFB04] B. ZHANG, H. WU, D. FENG et F. BAO : Cryptanalysis of a knapsack based two-lock cryptosystem. *In Applied Cryptography and Nandwork Security*, pages 303–309. Springer, 2004.

[ZWWL03] H. ZHAO, M. WU, Z.J. WANG et K.J.R. LIU : Nonlinear collusion attacks on independent fingerprints for multimedia. *In icme*, pages 613–616. IEEE, 2003.

Table des figures

Résumé

Les travaux présentés dans cette thèse se situent dans le contexte du fingerprinting. Un distributeur de documents multimédia souhaite se prémunir contre la redistribution illégale des données en insérant dans chaque copie distribuée un identifiant propre à chaque utilisateur. En cas de redistribution de cette copie, il est donc possible de retrouver l'utilisateur indiscret. Afin de contrer les attaques par collusion, qui surviennent lorsque les utilisateurs se mettent à plusieurs pour créer une copie pirate, les identifiants doivent être pris dans un code anti-collusion. Cette thèse étudie une famille de codes anti-collusion particulière, les codes de Tardos. Ces codes probabilistes sont particulièrement intéressants, car leur longueur est optimale. Ils sont de plus faciles à implémenter, et remarquablement efficaces. Dans cette thèse,nous présentons une amélioration de la phase d'accusation des codes de Tardos. Plus spécifiquement nous montrons comment l'optimiser en fonction de la stratégie d'attaque des pirates. Nous proposons également des moyens d'estimer à partir d'une copie pirate le nombre d'attaquants qui se sont rassemblés pour la créer, ainsi que la stratégie qu'ils ont employée. Notre solution s'appuie sur un algorithme itératif a la EM(Expectation-Maximization). Une autre contribution est l'étude d'un environnement asymétrique.Dans un tel environnement, seul l'utilisateur est en possession de la copie marquée avec son identifiant. L'identifiant doit être partiellement inconnu du distributeur tout en assurant sa fonction de traçage. Nous présentons un schéma de fingerprinting asymétrique entièrement spécifié intégrant les codes de Tardos, en utilisant une primitive cryptographique appelée Oblivious Transfer.

Abstract

The context of the work presented in this thesis is fingerprinting. A multimedia content distributor wants to be protected against illegal redistribution of data by embedding in each of the distributed copies a different identifier for each user. In the case of an illegal redistribution of the copy, it will be possible to find out the indiscreet user. In order to block collusion attacks, which happen when several users are getting together in order to create a pirated copy, identifiers must be part of an anti-collusion code. This thesis is about a particular anti-collusion codes family, called Tardos codes. These probabilistic codes are particularly interesting, because their length is optimal. They are easier to implement, and remarkably effective. We present an improvement of the accusation phases of Tardos codes. More specifically we show how to optimize this phase according to the attack's strategy of the pirates. We also propose a way to estimate from a pirated copy the number of pirates who participate in the creation and the strategy that was used. Our solution is based on a Expectation-Maximization iterative algorithm. Another contribution is the study of the asymmetric environment. In such an environment, the user only must possess the copy fingerprinted with his identifier. The identifier is partially unknown from the distributor while maintaining its function of tracking. We present an entirely specified fingerprinting asymmetric scheme including Tardos codes, using a cryptographic primitive called Oblivious Transfer.