France Télécom – Technologie et Innovation
France Télécom R&D
38-40, rue du Général-Leclerc
92794 Issy-Les-Moulineaux Cedex 9

UFR Mathématiques Informatique
Université de Bordeaux 1
351, cours de la Libération
33404 Talence

Code Collusion-secure et Watermarking

Sous la responsabilité de :
Stanislas FRANCFORT (FT R&D)
Alain YGER (Bordeaux 1)

Julien ZAK
Master Cryptologie et Sécurité Informatique
2004 / 2005

Remerciements

Je suis reconnaissant à Thierry BARITAUD de m'avoir accueilli à France Télécom Recherche & Développement au sein du laboratoire de sécurité MAPS/NSS et je tiens à remercier tout le département pour son accueil.

Mes remerciements vont tout particulièrement à Stanislas Francfort qui m'a encadré pendant ce stage. Je le remercie pour les conseils qu'il m'a donnés tout au long de mon stage, ainsi que pour l'autonomie et la confiance qu'il m'a accordé.

Je veux aussi remercier mon responsable de la faculté de Bordeaux 1 Alain Yger.

Mes remerciements vont également aux stagiaires de France Télécom qui ont participé à la bonne ambiance du laboratoire.

Résumé du stage

Titre : Watermarking et codes collusion-secure

Mots clés : watermarking, code collusion-secure, collusion, Costa Scalaire, Boneh & Shaw.

Nous sommes à l'heure ou des millions de fichiers s'échangent à travers le monde, ceci étant facilité par la performance des ordinateurs en constante augmentation et surtout par les offres des débits, des fournisseurs d'accès internet, qui ne cessent de croître. Par conséquent, le piratage devient de plus en plus facile, comme nous pouvons le voir avec l'industrie du disque, et la protection des contenus (images, films par exemple) de plus en plus difficile. Le problème du piratage devient une priorité pour toutes les industries qui veulent protéger des contenus.

On a donc vu apparaître les DRM (Digital Rights Management) qui sont des technologies qui permettent aux propriétaires de contenu de protéger et de contrôler la diffusion de leurs produits. Cette protection est assurée par le cryptage du contenu multimédia et n'autorise l'accès qu'aux personnes disposant d'une licence pour lire ce contenu. Mon stage s'inscrit dans cette lignée, alliant le watermarking et les codes collusion-secure pour une étude sur la traçabilité des pirates.

Rappelons que le watermarking consiste à rajouter sur un médium, en l'occurrence une image, une marque qui doit être suffisamment imperceptible pour ne pas détériorer le médium et suffisamment robuste pour être décelée même après une ou plusieurs attaques.

Rappelons qu'un code collusion-secure est un code qui a un « bon » algorithme de traçage. Dans la suite un mot du code désignera une suite de bits. Si l'on prend plusieurs mots du code (ils sont tous différents deux à deux) et que l'on effectue des opérations avec ces mots (exemple la moyenne des bits) pour forger un nouveau mot, l'algorithme de traçage permettra de retrouver les mots ayant servis à le forger.

On se place dans la situation du distributeur qui veut identifier des pirates. On va donc générer un code collusion-secure, et insérer chaque mot du code à l'aide du watermarking dans une image. Par ce processus, à chaque personne détenant une image, on lui associe un mot du code (unique) qui lui est propre. On appelle collusion un ensemble de personnes malveillantes c'est-à-dire un groupe de pirates. On va donc étudier les différentes attaques liées à la collusion sur des images, ainsi que le traçage des pirates appartenant à cette collusion.

On va mettre en œuvre un algorithme de watermarking, Costa Scalaire et le code de Boneh & Shaw qui est collusion-secure. On va donc se servir de l'algorithme de Costa Scalaire pour insérer les mots du code de Boneh & Shaw dans des images. Ensuite on formera des collusions et on essaiera de retrouver les pirates.

SOMMAIRE

1. Présentation générale

S'il est une leçon essentielle à tirer de ce stage, c'est bel et bien de voir loin devant soi et de bien cerner les tenants et les aboutissants d'une mission. Or cerner les objectifs d'une mission ne peut se faire sans comprendre la stratégie du groupe. Ainsi nous brosserons à grands traits le portrait d'une entité, France Télécom R&D, qui, plus que jamais, doit faire figure de proue au sein du groupe France Télécom dans un contexte de concurrence acharnée entre les opérateurs téléphoniques.

1.1. Un peu d'histoire : le CNET

Fondé en 1944, le CNET (Centre National d'Études des Télécommunications) a contribué de façon déterminante aux progrès des services et des réseaux de télécommunications français, ainsi qu'à l'édification d'une industrie forte dans ce secteur.

Initiateurs de paris technologiques audacieux, ses travaux ont abouti, en 1970, à l'installation du premier commutateur temporel en Europe. En 1978, eut lieu l'ouverture commerciale du premier réseau de données utilisant la commutation par parquets (Transpac), suivie, en 1981, par l'ouverture du service vidéotex Télétel célèbre par son terminal le Minitel. Ces réalisations ont connu un succès de réputation mondiale.

Le CNET s'est aussi illustré parmi les pionniers des télécommunications spatiales. Il fut, en particulier, le concepteur et l'architecte des satellites Télécom 1 et Télécom 2. Il eut un rôle déterminant dans la définition et l'introduction du réseau numérique à intégration de services (Numéris). Grâce aux innovations du CNET et des GIE associés, la carte à mémoire a pu atteindre un niveau de maturité permettant la généralisation, en France, des Publiphones à carte. Ces innovations ont conduit au développement des systèmes ouverts de contrôle d'accès pour la télévision à péage et des nouveaux systèmes d'identification des utilisateurs de services numériques radiomobiles.

Depuis le 1er mars 2000, le CNET poursuit ses activités sous le nom de France Télécom Recherche & Développement.

1.2. Missions

Les laboratoires de France Télécom Recherche & Développement sont à l'origine d'environ 70% des produits et services commercialisés par le Groupe. Avec près de 3700 chercheurs et ingénieurs, France Télécom Recherche & Développement est un atout majeur pour France Télécom. Ses missions consistent à anticiper les évolutions technologiques et les changements d'usages, innover pour offrir aux clients le meilleur des technologies, et explorer de nouvelles sources de croissance pour fournir au Groupe les assises d'un développement durable. Peu d'opérateurs dans le monde peuvent justifier d'un tel atout, pensé et organisé pour servir la stratégie globale de développement.

Avec ses 8 sites en France, plus deux aux États-Unis (San Francisco, Boston), un au Japon (Tokyo), un en Angleterre (Londres) et un en Chine (Pékin), il accompagne également le développement international de France Télécom en mettant son expertise au service de ses filiales étrangères, en s'impliquant auprès de grands groupes industriels, de la communauté scientifique mondiale et des instances internationales.

Pour répondre à ses ambitions, France Télécom Recherche & Développement a accéléré sa réorganisation dès 1995 en se recentrant sur les domaines de recherche liés aux stratégies de marché de France Télécom, et en donnant la priorité à la recherche appliquée. En effet, lorsqu'on a détecté une opportunité d'innovation et déterminé sa pertinence, priorité est donnée à la recherche d'applications concrètes et à la réduction du cycle de développement. France Télécom Recherche & Développement poursuit son évolution aujourd'hui et améliore en permanence sa capacité d'innovation à la fois technologique, marketing, managériale et organisationnelle.

Ainsi, le centre de recherche contribue également à développer le capital humain du Groupe, en jouant un rôle de vivier de compétences et en aidant les Ressources Humaines à anticiper les évolutions de métiers et les besoins en compétences.

1.3. Quelques chiffres

Voici un résumé des chiffres les plus significatifs, mettant en évidence l'importance du laboratoire de Recherche & Développement pour le groupe France Télécom :

- Plus de **3 700 collaborateurs**, dont 3 000 ingénieurs, chercheurs et techniciens.
- **14 sites d'implantation** en France, aux USA (San Francisco et Boston), au Japon (Tokyo), en Angleterre (Londres) et en Chine (Pékin). (Voir Fig. 1)
- **7 094 brevets** détenus au niveau mondial.
- **418 nouveaux brevets** et **349 logiciels** déposés sur les douze derniers mois. Un nombre de contrats de licences et de transferts de savoir-faire en augmentation permanente.
- **80% des clients satisfaits** de la qualité des relations et de l'écoute.
- **200 recrutements** chaque année.
- **150 thésards** français et étrangers.
- **500 stages-terrain** réalisés par les cadres de France Télécom Recherche & Développement dans les services opérationnels de France Télécom.
- **Une quinzaine de start-ups** créées depuis fin 1998 sur l'initiative d'ingénieurs de France Télécom Recherche & Développement, générant presque un millier d'emplois nouveaux.

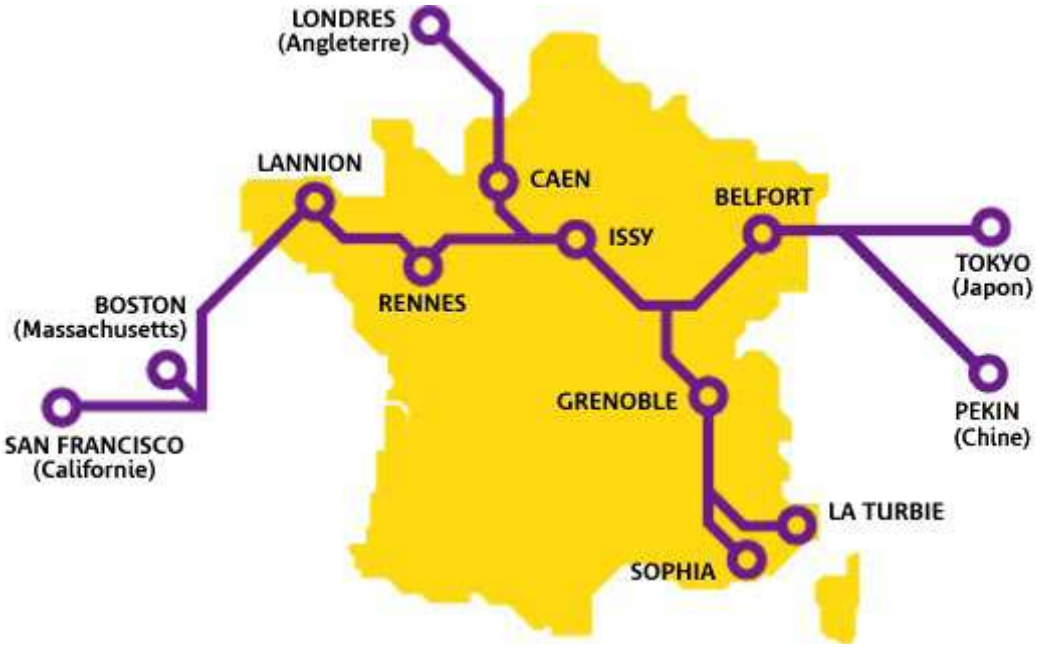


Fig. 1 – Sites d'implantation

1.4. Structure

Une structure de la taille de France Télécom Recherche & Développement nécessite une organisation stricte est un découpage fin des différents pôles de recherche. Récemment la division Recherche & Développement a d'ailleurs subi une refonte complète de sa structure.

Ainsi, au plus haut niveau les Recherches ne sont plus gérées par sept directions mais par six Centres de Recherche & Développement (CRD), regroupés autour de trois axes de recherches différents.

Deux CRDs sont voués aux services pour les clients :

- **SIRP** : Services Intégrés Résidentiels et Personnels
- **BIZZ** : Services aux Entreprises

Deux CRDs sont responsables d'activités d'intérêt commun :

- **MAPS** : Middleware et Plates-formes Avancées
- **TECH** : Technologies

Enfin, deux CRDs sont axés sur l'intégration des réseaux :

- **CORE** : Coeur de Réseau
- **RESA** : Réseaux d'Accès

Chaque CRD est ensuite découpé en laboratoires, eux même découpés en Unités de Recherche et Développement (URD). C'est à Issy les Moulineaux, au sein du laboratoire de Sécurité des Services et des Réseaux (NSS), dépendant du CRD Middleware et Plates-formes

Avancées que j'ai eu l'opportunité d'effectuer mon stage. Nous allons donc présenter brièvement le laboratoire, les différentes URDs qui y sont rattachées.

1.5. Une expertise technique de pointe

Dans le cadre de l'enrichissement de son offre de services, France Télécom doit ouvrir ses réseaux aux clients, aux prestataires et autres opérateurs, tant pour améliorer son offre que pour respecter les nouvelles législations. Le laboratoire de Sécurité des Services et des Réseaux a pour mission d'apporter son expertise technique en matière de sécurité ainsi que les briques de sécurité nécessaires afin d'assurer la pérennité des produits proposés par le groupe.

Le laboratoire se divise en trois URDs à savoir :

- **RMC**, Sécurité pour les Réseaux, la Mobilité et la Confidentialité : cette URD étudie et améliore la sécurité du réseau sémaphore, du réseau intelligent, de tous les aspects de la mobilité, ainsi que des services liés à la confidentialité,

- **SPR**, Sécurité des Services, du Paiement et des systèmes Réparties : cette URD conçoit et met en place la sécurité des services offerts aux clients de France Télécom, la sécurité des paiements électroniques et évalue et met en oeuvre la sécurité des systèmes répartis,

- **SII**, Sécurité Intranet/Internet, URD qui évalue et améliore la sécurité des réseaux de type Internet ou Intranet pour les besoins de France Télécom et de ses clients.

Mon stage s'est précisément déroulé dans l'URD RMC dont les missions ont détaillées ci-dessus.

2. Sujet et problématique du stage

2.1. Intérêt du stage

Ce stage s'inscrit dans l'étude de la protection de contenu. Nous allons donc nous placer à la fois dans la position du distributeur qui veut protéger ses contenus et des pirates qui vont attaquer les médiums.

Pour protéger ses contenus, le distributeur va insérer dans chacun, une marque qui sera unique afin d'identifier les personnes ayant droit de le détenir. Par exemple, un distributeur délivrant 1 million d'images à 1 million de personnes. En apparence les images ont l'air identiques mais chacune comporte une marque unique, typiquement une suite de bits. A chaque personne ayant un contenu, on lui associe une marque. Ainsi lorsque le distributeur veut savoir à qui appartient une image, il extrait de l'image la suite de bits, la marque, qui lui identifie la personne à qui l'image appartient. Ce processus permet au distributeur de lutter contre les redistributions naïves de leurs contenus. On entend par 'redistribution naïve ', le fait de redistribuer un contenu sans l'altérer, c'est-à-dire sans effectuer d'attaques dessus. Ainsi une personne redistribuant naïvement une image se fera identifier facilement par le distributeur et pourra selon le cas encourir des suite pénales.

Mais quand est-il, si plusieurs personnes se regroupent mettant en commun leurs contenus afin de « fabriquer » un nouveau contenu. Peut-on les retrouver à partir du contenu « fabriqué » ?

2.2. Le watermarking

Le **watermarking** consiste à rajouter sur un médium, en l'occurrence une image, une marque qui doit être suffisamment imperceptible pour ne pas détériorer le médium et suffisamment robuste pour être décelée même après une ou plusieurs attaques. Nous reviendrons plus en détails sur la définition du watermarking dans la suite.

2.3. Les codes collusion-secure

Un code **collusion-secure** est un code qui a un « bon » algorithme de traçage.
Dans la suite, un mot du code désignera une suite de bits. Si l'on prend plusieurs mots du code (ils sont tous différents deux à deux) et que l'on effectue des opérations avec ces mots (exemple la majorité des bits) pour forger un nouveau mot, l'algorithme de traçage permettra de retrouver les mots ayant servis à le forger.

2.4. Objectif du stage

On appelle **collusion** un ensemble de personnes malveillantes c'est-à-dire un groupe de pirates.

Notre étude va porter sur la **traçabilité** de pirates appartenant à une **collusion** ainsi que sur les **fonctions de forge**.

On est dans le scénario ou le distributeur délivre à plusieurs personnes le même contenu, une image par exemple. Dans chaque contenu on insère un mot du code C, qui est collusion-secure, à l'aide du watermarking. Chaque personne a donc un mot du code C, Mi (i∈ I avec I ensemble fini) unique qui sert à l'identifier. Ensuite, un groupe de personnes ayant chacune un contenu, se réunissent. Ils ont donc un ensemble de mots du code C, Mj (j ∈ J avec J un sous ensemble de I), qui est à leur disposition et forme un nouveau mot Mf à l'aide d'une fonction de forge.

Le but est donc de retrouver les Mj , donc les pirates, à partir de Mf.
L'objectif de ce stage est donc de choisir un médium. De chercher, et développer un « bon » code collusion-secure, un « bon » algorithme de watermarking, et d'étudier différentes fonctions de forge que peuvent utiliser les pirates.

Le schéma suivant est un résumé de toutes les phases que nous venons d'évoquer. On peut voir les termes de fingerprint et Customer's ID apparaître, mais pour l'instant on peut les interpréter comme synonyme de « mot du code collusion-secure ».

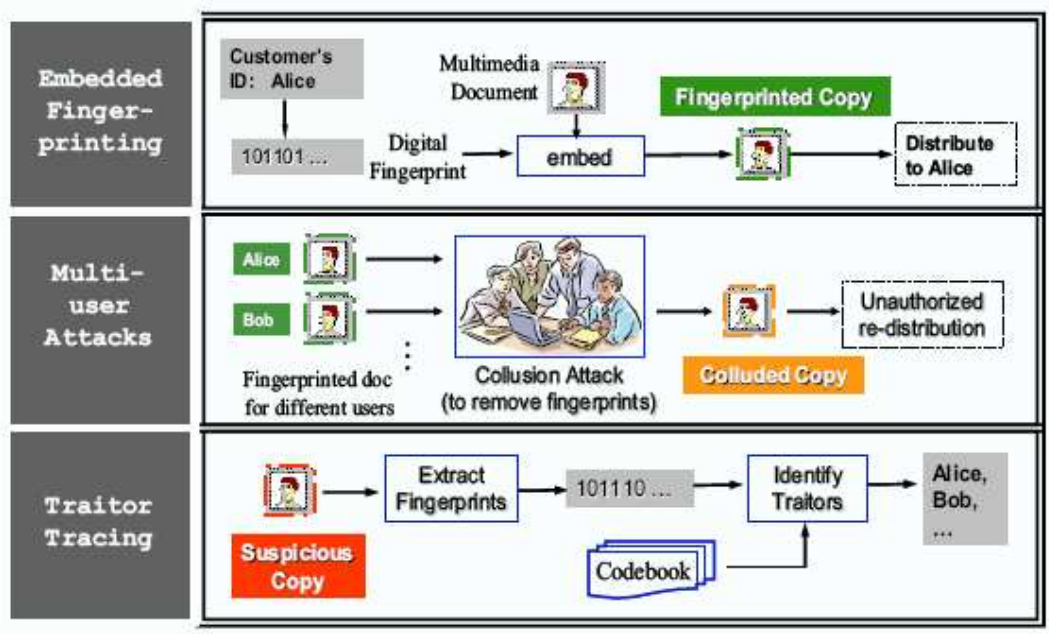


Figure 1. Using Embedded Fingerprinting for Tracing Users.

2.5. Cadre du stage

Pour cette étude nous avons sélectionné :
- Type de contenu : les **images**
- Algorithme de watermarking : **Costa Scalaire**
- Code anti-collusion : **Boneh &Shaw**

Le type de contenu n'est pas important étant donné que l'on pourra adapter notre étude à n'importe quel autre type de contenu. Notre étude porte sur les attaques par collusion et non sur un type de contenu.

En revanche, les images entraînent une contrainte sur la taille des données que l'on peut insérer à l'aide du watermarking. En effet, notre algorithme de watermarking, Costa Scalaire, permet d'insérer un bit par bloc de 8x8. Et nous allons insérer des messages de l'ordre de 250 000 bits, ce qui implique que nous allons devoir utiliser la concaténation d'images pour pouvoir faire cela.

L'algorithme de watermarking choisi est celui de Costa Scalaire, en raison de sa simplicité d'implémentation, afin de pas compliquer ou fausser les tests.

Le code anti-collusion développé est celui de Boneh & Shaw. Nous avons opté pour ce code en raison de sa simplicité d'implémentation et de sa taille. La taille de ce code est « relativement » courte, même si elle nous amène a des mots de code de 250 000 bits. Pour n utilisateurs on construit un code de taille : $O(\log(n)^4 \log(1/\varepsilon)^2)$ avec $\varepsilon > 0$, ou ε détermine l'erreur de probabilité de l'algorithme de traçage qui donnerait un innocent comme coupable.

Pour lutter contre les attaques par collusion, nous allons donc insérer dans chaque image distribuée, à l'aide du watermarking, les mots du code de Boneh & Shaw.

Nous allons donc présenter le watermarking et en particulier celui de Costa Scalaire. Ensuite nous allons nous attacher à comprendre la construction du code de Boneh & Shaw, et finir par la combinaison des deux techniques dans les parties Tests effectués et travail réalisé.

LE WATERMARKING
Costa Scalaire

3. Le watermarking

3.1. Présentation générale

Le watermarking consiste à rajouter, sur un medium (qui peut être une image, une chanson, un film vidéo), une **marque** (en anglais, watermark signifie filigrane) qui doit être suffisamment **imperceptible** pour ne pas détériorer le medium et suffisamment **robuste** pour pouvoir être décelée même après traitement du medium que ce soit un traitement usuel ou celui résultant d'une attaque du système de marquage.

Le contenu d'une marque, typiquement quelques bits d'information, peut être de différentes natures.

1. Elle contient des informations sur les **permissions attachées au document**. Prenons l'exemple d'un film vidéo : il pourrait être marqué en copie illimitée, copie interdite (dans le cas où il est acheté dans le commerce), copie une fois seulement (film diffusé à la télé) charge au "magnétoscope" de transformer cette marque en une marque de copie interdite. Dans ce type de marquage, c'est au matériel de copie que revient la charge de détecter la marque.
2. La marque peut **indiquer qui est propriétaire du document**. Ainsi, toute personne qui s'en réclamera propriétaire illégalement pourra être condamnée légalement, la marque faisant office de preuve devant un tribunal.
3. On peut également **marquer l'ayant droit du document**, c'est-à-dire la personne à qui le propriétaire a donné une copie. Ainsi, chaque copie du document contient une marque différente (qu'on appelle empreinte) permettant d'identifier son utilisateur. Cette technique est appelée le "fingerprinting". En plus des attaques usuelles du watermarking, le fingerprinting doit résister également aux collusions entre un nombre limité d'ayant droit qui comparent leur document dans le but d'enlever toute empreinte permettant d'identifier l'un d'eux.

3.2. Les caractéristiques d'un bon marquage

3.2.1 Imperceptibilité :

La marque doit être imperceptible pour tout être humain, c'est-à-dire qu'il est impossible au non-expert d'entendre ou de voir la marque. Dans le cas du tatouage d'images, on parle d'invisibilité plutôt que d'imperceptibilité.
Ceci pose d'emblée le problème suivant : si on trouve un algorithme de compression du medium qui ne garde que les informations perceptibles de celui-ci, alors cette compression fait disparaître la marque. On peut, si l'on craint une telle compression, imposer une notion de déformation minimale du medium par la marque plutôt que celle d'imperceptibilité.
Un autre bémol à la notion d'imperceptibilité est apporté par une technique employée pour le fingerprinting de séquences vidéo. On marque une séquence différemment selon le point de vue de la scène qu'on choisit (i.e. selon la position de la caméra), étant entendu que le problème de transformer une image en la même image vue d'un autre point est un problème difficile, voire impossible. La marque est alors perceptible. Toutefois dans la mesure où elle ne nuit pas à l'information véhiculée par le medium, on considère que c'est une bonne marque pour ce critère.

3.2.2 Spécificité :

Quoique imperceptible, la marque doit être suffisamment spécifique pour être clairement identifiable lors de son extraction. Une marque trop peu perceptible serait peu robuste et, plus grave, pourrait être détectée à tort. Si les techniques de marquage veulent conduire à l'élaboration de preuves légales, il faut que les marques soient assez spécifiques pour ne jamais condamner un innocent.

3.2.3 Robustesse :

Le medium marqué va subir des transformations de nature très variées, comme le passage dans un canal analogique et rééchantillionnage (impression/scannérisation par exemple pour les images), compression avec perte d'information (telle la compression jpeg pour les images ou mp3 pour les sons), déformations non linéaires, bruits de canal additifs ... Il va sans dire que la marque doit être assez robuste pour rester décelable tant que la dégradation du medium par ces transformations naturelles reste peu signifiante.

3.2.4 Résistance aux attaques :

Plus que des transformations naturelles, le medium va subir l'attaque de pirates voulant effacer cette marque. Ces pirates sont supposés, d'après le principe de Kerckhoffs, connaître l'algorithme de marquage et donc être en mesure de développer des attaques spécifiques à cet algorithme. Là encore, il faut que la destruction de la marque ne puisse se faire sans détérioration significative du medium. Il faut aussi, dans le cas où le marquage s'est fait grâce à la connaissance d'un secret (comme une clé secrète de marquage), que les pirates ne puissent avoir accès à ce secret par l'étude du medium.

3.3. Les modèles de marquage

De même qu'il existe différents types de cryptographie selon qu'on utilise une clé symétrique identique pour les deux interlocuteurs ou des paires de clés différentes pour chaque interlocuteur, de même il existe plusieurs formes de tatouage/extraction de marques. La figure ci-dessous illustre une forme générique de ce procédé :

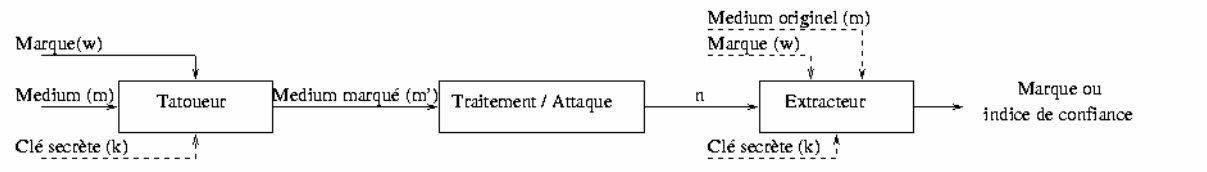


Fig 1 : Processus générique d'insertion/attaque/extraction d'une marque.

Par convention, nous désignons par algorithme de tatouage (ou tatoueur) l'algorithme qui incruste la marque dans l'image et par algorithme d'extraction (ou extracteur) l'algorithme qui retrouve cette marque. Quand nous parlerons d'algorithme de marquage, ce sera pour désigner la conjonction d'un tatoueur et d'un extracteur associé.

On distingue d'abord deux types d'extracteurs de marques :

- Si on passe à l'extracteur la marque m qu'on suppose avoir marqué le support et que celui-ci répond par oui ou par non (ou par un indice de confiance compris entre 0 et 1 dans le cas d'une extraction souple), on dit que l'algorithme de marquage est de type II. Bien entendu, ce type d'algorithme nécessite de savoir à l'avance à quelle marque on aurait affaire ;

- Un cas plus agréable serait celui où on ne passe pas de marque supposée à l'extracteur, charge à lui de déterminer la ou les marques éventuelles tatouées sur le medium n. Les algorithmes répondant à cette spécification sont dits de type I.

D'autres caractéristiques que le besoin ou non de fournir une marque à l'extracteur entrent en jeu. C'est ainsi qu'on distingue les algorithmes de marquage suivants :

- Le marquage privé est celui où le support originel est donné à l'extracteur. Dans ce type de marquage, on compare l'original au medium m récupéré pour extraire la marque ;

- Le marquage aveugle est celui où l'extracteur n'a pas connaissance du medium originel m. Seule la clé secrète de chiffrement lui est nécessaire pour extraire la marque ;

- Le marquage asymétrique est celui où l'extraction de la marque ne nécessite pas la connaissance d'un secret. Ceci implique que tout le monde est capable de lire la ou les marques du medium sans pouvoir les effacer. Cela pourrait se faire par un marquage sans clé ou alors par un tatouage avec clé secrète et une extraction avec la clé publique correspondante (dans un schéma analogue à celui de la cryptologie asymétrique).

Les deux premiers modèles d'algorithmes de marquage nécessitent donc la connaissance d'un secret pour l'extraction de la marque. Les protocoles mis en place au-dessus de ces algorithmes nécessitent l'existence d'un tiers de confiance dépositaire de ce secret.
Le marquage privé reste très lourd à manier puisque le tiers de confiance doit alors posséder l'original de tous les mediums marqués. La base de données peut alors être gigantesque et peu pratique à utiliser de façon sécurisée.
Dans le cas du marquage aveugle, la même clé peut servir au marquage d'un grand nombre de mediums (à condition que l'algorithme de marquage soit solide cryptologiquement). De plus la taille de la clé est beaucoup plus petite que celle du medium qu'elle marque. C'est donc une solution plus satisfaisante que celle du marquage privé qui n'a que peu de chance de trouver des applications viables économiquement.

Le marquage asymétrique représente la panacée. Plus besoin de tiers de confiance, la marque est une propriété du médium que tout le monde peut lire. Et pourtant, personne ne peut l'enlever. Toutefois, on n'a pas trouvé d'algorithme valable de marquage asymétrique et on doute de jamais en trouver.

3.4. Quelques algorithmes de marquage

Nous ne nous intéressons dorénavant qu'au marquage d'images fixes et planes. Le lecteur, cherchant une explication plus détaillée pourra se rapporter à la thèse de Caroline Fontaine.

3.4.1 Modification des bits de poids faibles

Pour s'assurer de l'invisibilité de la marque, les premiers algorithmes allaient inscrire la marque dans les bits de poids faible de la luminance de l'image. Voir par exemple le rapport de recherche de Dautzenberg et Boland.
Cette marque est très rudimentaire et relève de la stéganographie plutôt que du watermarking. Cette marque est très facile à modifier ou à enlever. On peut décider de cacher les emplacements des bits marqués avec une clé secrète, ce qui empêchera l'écriture d'une autre marque à la place (voir par exemple, la technique du Patchwork). Par contre, la marque ne résistera pas à une compression jpeg où à un bruit blanc gaussien additif.
Ce type de marquage n'est donc absolument pas robuste.

3.4.2 Technique du "Patchwork"

Pour donner à cette dernière technique un peu de solidité, on peut décider de répéter un grand nombre de fois le même bit pour qu'une étude statistique nous donne le bit marqué.
Bien qu'étant une amélioration, cet algorithme est aussi très faible ; en effet, une étude statistique des bits de poids faible de l'image renseigne le pirate sur l'existence du marquage qu'il peut ensuite enlever à loisir.
Le raffinement proposé par la technique du patchwork (voir encore le rapport de recherche de Dautzenberg et Boland) est double :

1. Pour chaque bit forcé à 1 par le tatouage, on force un autre bit à 0. Ainsi, les propriétés globales statistiques de l'image sont inchangées ;

2. De plus, pour rendre cette marque invisible localement, on utilise une clé qui va coder l'emplacement des bits à 0 et des bits à 1.

L'extraction de la marque se fait alors par un calcul de la somme des différences entre les positions des bits donnés par la clé.
Cette technique est plus robuste que la première ; en effet, l'ajout d'une forte redondance permet de compenser les effets du bruit blanc additif. Toutefois, ce marquage ne résiste pas à de petites déformations géométriques, ni même à la compression JPEG.

3.4.3 Algorithme de Koch et Zhao

Constatant la mauvaise performance des algorithmes de marquage dans le domaine spatial vis-à-vis de la compression JPEG, Koch et Zhao ont proposé un nouvel algorithme se fondant sur le marquage dans le domaine fréquentiel.
L'idée de base est d'extraire un certain nombre de carrés de 8x8 pixels de l'image, de calculer la transformée discrète en cosinus (DCT) de ces blocs et d'aller marquer un bit sur les moyennes fréquences correspondantes, sachant que la modification des basses fréquences de l'image la changerait trop et que les hautes fréquences sont enlevées par la compression JPEG.

3.4.4 Watermarking par étalement de spectre

Utilisé dans les télécommunications militaires, l'étalement de spectre consiste à envoyer un message sur un grand spectre de fréquences de telle manière que, à toute fréquence, la puissance du signal émis soit inférieure au bruit. Ainsi, localement, l'émission est toujours imperceptible et ce n'est qu'en écoutant sur l'ensemble du spectre d'émission et avec la connaissance du procédé utilisé (qu'on peut paramétrer par une clé) que l'on pourra ``entendre'' le message émis.
Certains algorithmes de marquage d'images suivent ce principe. Un exemple est de sélectionner des blocs de bits, tous de la même taille, de se donner une suite pseudo-aléatoire de la taille des blocs et de rajouter à tout bit d'un bloc la marque "xor" le bit correspondant de la suite pseudo-aléatoire. Cet algorithme est expliqué dans l'article de Hernández et Pérez-González.
Cette technique d'étalement de spectre rappelle beaucoup celle du Patchwork. Elle perd toute sa robustesse face à des déformations géométriques de l'image. Pour pallier ces carences, une méthode en vogue est d'introduire des suites auto-corrélées dans l'image.

3.4.5 Watermarking fractal

Une technique initiale est présentée par Puate et Jordan en 96, fondée sur la compression fractale décrite par Fisher. La compression fractale repose sur la détermination d'un *IFS* (Iterated Functions System) qui permet de représenter l'image comme un *attracteur*. Un IFS est constitué d'un ensemble de fonctions w_i : K -> K, contractantes et définies sur un compact (K, d). L'application W qui, à une partie A de K, définie par $W(A) = \cup_{i=1}^{n} w_i(A)$ est contractante pour la métrique de Hausdorff et admet un unique point fixe, l'attracteur de l'IFS.

Le principe de la compression fractale est de déterminer les w_i dont l'attracteur est l'image qu'on souhaite compresser et qui, pour simplifier le problème, sont des fonctions affines. On partitionne alors l'image en carrés R_i de taille n par n, appelés *range blocks*, et on recherche des carrés de taille égale ou différente, appelés *domain blocks*, transformables par les w_i en range blocks via une transformation spatiale et en niveaux de gris. En compression, les domain blocks sont en général plus gros que les range blocks, mais cette contrainte n'intervient plus dans le cadre du watermarking.

Le procédé de marquage élaboré par Puate et Jordan associe à chaque range block deux ensembles de domain blocks possibles. La marque, composée sur un alphabet binaire, détermine l'ensemble dans lequel on doit chercher le domain block qui minimise l'erreur au sens des moindres carrés.

Une autre technique, proposée dans Bas, Chassery et Davoine, met en oeuvre l'étude des similarités dans l'image. La méthode consiste à rechercher un IFS sur l'image pour y dissimuler de nouvelles similarités servant de marque. En forçant ainsi les similarités qui définissent la marque, on conserve le contrôle sur le code fractal (i.e. l'IFS). Pour la détection, il suffit alors de rechercher le range block qui minimise l'erreur au sens des moindres carrés pour un domain block donné et de s'assurer qu'il correspond bien a celui désiré.

Pour plus d'information, vous pouvez consulter le site de l'INRIA d'où a été tiré cette présentation : www-rocq.inria.fr/codes/Watermarking

3.5. Algorithme de watermarking utilisé : Costa Scalaire

3.5.1 Petit rappel sur les différents codages de couleurs

Afin de pouvoir manipuler correctement des couleurs et échanger des informations colorimétriques, on fait appelle a différent codage tel que :
- le RGB (Rouge, Vert, Bleu, en anglais Red, Green , Blue)
- le YUV
- le YIQ

Il en existe d'autre comme par exemple le codage CMYK, CIE , etc…

Le RGB est s'en doute le plus connu, il a été mis au point en 1931 par la Commission Internationale de l'Eclairage (CIE), et consiste à représenter l'espace des couleurs à partir de trois rayonnements monochromatiques de couleurs :
- rouge (de longueur d'onde égale à 700,0 nm)
- vert (de longueur d'onde égale à 546,1 nm)
- bleu (de longueur d'onde égale à 435,8 nm)

Cet espace de couleur correspond à la façon dont les couleurs sont généralement codées informatiquement. Le modèle RGB propose de coder sur un octet chaque composante de couleur, ce qui correspond à 256 intensité de rouge (28), 256 intensité de vert et 256 intensité de bleu soit 16 777 216 possibilités théoriques de couleurs différentes, c'est-à-dire plus que ne peut discerner l'œil humain (environ 2 millions). Toutefois cette valeur n'est que théorique car elle dépend fortement du matériel d'affichage utilisé.

Le YUV (appelé CCIR 601) est un modèle de représentation de la couleur dédié à la vidéo analogique. Il s'agit du format dans les standards PAL (Phase Alternation Line) et SECAM (Séquentiel Couleur avec Mémoire). Le paramètre Y représente la luminance (c'est-à-dire l'information sur le noir et blanc) tandis que U et V permettent de représenter la chrominance, c'est-à-dire l'information sur la couleur. Ce modèle a été mis au point afin de permettre de transmettre des informations colorées aux téléviseurs couleurs, tout en s'assurant que les téléviseurs noir et blanc existant continuent d'afficher une image en tons de gris.
Voici les relations liant Y à R, G et B, U à R et à la luminance, et enfin V à B et à la luminance :
- $Y = 0,299\ R + 0,587\ G + 0,114\ B$
- $U = -0,147\ R - 0,289\ G + 0,436\ B = 0,492\ (B - Y)$
- $V = 0,615\ R - 0,515\ G - 0,100\ B = 0,877\ (R - Y)$

Ainsi U est parfois noté Cb, et V noté Cr, d'où la notation YCbCr.

Le YIQ est très proche du modèle YUV. Il est notamment utilisé dans les standards vidéo NTSC (utilisé entre autres aux Etat Unis et au Japon).
Le paramètre Y représente la luminance, I et Q représente l'Interpolation et la Quadrature.
Les relations entre ces paramètres et le modèle RGB sont les suivants :
- $Y = 0,299\ R + 0,587\ G + 0,114\ B$
- $I = 0,596\ R - 0,275\ G - 0,321\ B$
- $Q = 0,212\ R - 0,523\ G + 0,311\ B$

Nous avons introduit ces petits rappels sur le codage des couleurs. Dans la suite, on va s'intéresser en particulier à la luminance, c'est-à-dire à Y, dans les images. En effet, on va insérer des bits juste sur la composante Y. Par conséquent, on prend une image I, on va lui extraire sa composante Y, on lui insère des bits à l'aide de l'algorithme que l'on va présenter juste après, puis on réinsérer la composante Y codé afin d'obtenir notre image I codée. La phase de décodage se fera donc en conséquence, on décodera que la composante Y de l'image pour obtenir le message inséré.

3.5.1 Cadre de l'algorithme

Tout d'abord, à l'aide de la figure 1, nous allons essayer de voir ou s'intègre notre algorithme dans le panorama des différentes techniques pour cacher de l'information. Il est évident que cette figure est incomplète, on pourrait rajouter d'autres watermarkings aveugle par exemple, mais elle donne une idée de l'échelle à laquelle se situe l'algorithme Costa Scalaire.

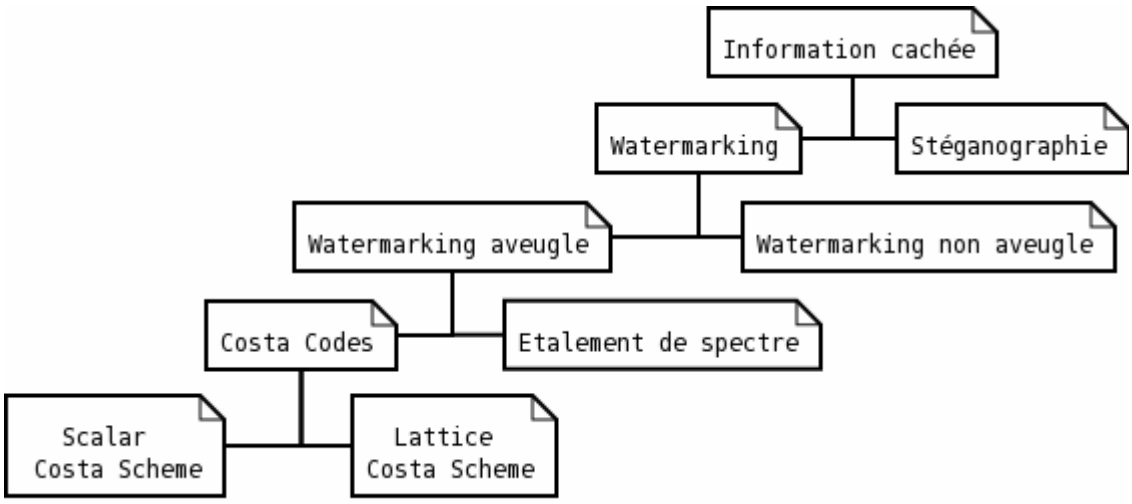


Fig 1. Les différentes méthodes pour cacher de l'information

On est dans la situation ou l'on a un message M, que l'on veut insérer dans une image I. Typiquement, M est une suite de bits, mais pour la suite de l'explication on supposera que M est égal soit à 0 soit à 1. Dans toute la suite, on se place au niveau de la composante Y. C'est-à-dire que lorsque on parlera de bloc, il s'agira des blocs de la composante Y de l'image. Les phases de code et décodage de notre algorithme s'effectuent au niveau de la composante Y.

Costa Scalaire est un algorithme de watermarking qui permet d'insérer 1 bit d'information par bloc. C'est-à-dire que l'on extrait de l'image I sa composante Y que l'on va découper par blocs ; blocs dans lesquelles on code 1 bit du message. La taille des blocs doit être fixée au début de l'algorithme, mais elle est peut être variable. Cependant on retrouve souvent dans la littérature des blocs de taille 8x8, nous avons donc opté pour ce choix. De plus cet algorithme est basé sur la quantification (discrétisation) des coefficients des blocs, à l'aide de deux quantificateurs selon si l'on code un 0 ou un 1. La phase de décodage se fait donc en conséquence, on extrait de l'image I codée, sa composante Y que l'on va découper en blocs 8x8, d'où l'on va extraire les bits d'informations codés. On obtiendra donc le message inséré.

3.5.3 Phase de Codage :

3.5.3.1 Théorie

• On construit un premier quantificateur q(.).
• Chaque message m module un signal de décalage d[m] différent, et ainsi à chaque message correspond un quantificateur Qm.
Ainsi, si s est le signal résultant de l'insertion d'un message m dans un signal hôte x, on aura :

$$s(x ; m) = q(x+d[m]) - d[m]$$

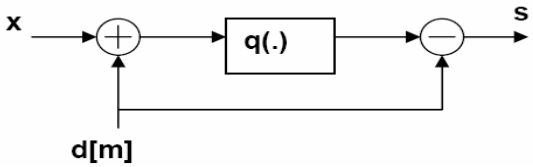


Fig1. Insertion d'un message m dans un signal hôte x

A partir d'un quantificateur scalaire de pas Δ, on construit deux quantificateurs Q0 et Q1 en considérant deux signaux de décalage d[0] et d[1] :

• d[0] est choisi arbitrairement (généralement d[0] = 0).
• d[1] est construit à partir de d[0] de telle façon que les points de construction de Q0 et Q1 soient le plus possible éloignés les uns des autres :

$d[1] = d[0] + \Delta/2$ si $d[0] \geq 0$

$d[1] = d[0] - \Delta/2$ si $d[0] < 0$

On considère un coefficient donné d'un bloc 8x8. On le quantifie par Q0 si le bit à insérer est 0, et on le quantifie par Q1 si le bit est 1.

3.5.2.2 Notre phase de codage

On pose x comme étant le coefficient du bloc examiné.
d[0] = 0 ;
d[1]= Δ/2 ;
Δ = 10

1- Calcul du quantificateur Q(x) :

Le quantificateur va permettre de quantifier les blocs. C'est-à-dire que l'on va discrétiser l'ensemble des blocs à partir du quantificateur. La quantification va s'effectuer en fonction du pas Δ.
On pose $Y(x)= (x / \Delta) - E(x / \Delta)$,
Avec E(.) notant la partie entière inférieure, (fonction floor() de Scilab)
Si $Y(x) \leq ½$, $Q(x) = \Delta * (E (x / \Delta))$
Si $Y(x) > ½$, $Q(x) = \Delta * (E (x / \Delta + 1)$

2- Codage du message M (0 ou 1) dans le bloc :
On se place dans le bloc B
M = 1 ou 0 ; selon que le bloc B code un 1 ou un 0
Pour toutes les valeurs x du bloc B :
Si M = 0 : s = Q (x)
Si M = 1 : s = Q (x + Δ / 2) - (Δ / 2)
s remplace le coefficient x.

Nous allons prendre un petit exemple. On se place dans un bloc de taille 8x8 et on pose Δ=10.

Dans ce bloc on va analyser comment on va coder un coefficient de luminance x, avec x =7, selon si l'on veut coder un 1 ou un 0.
On a donc :

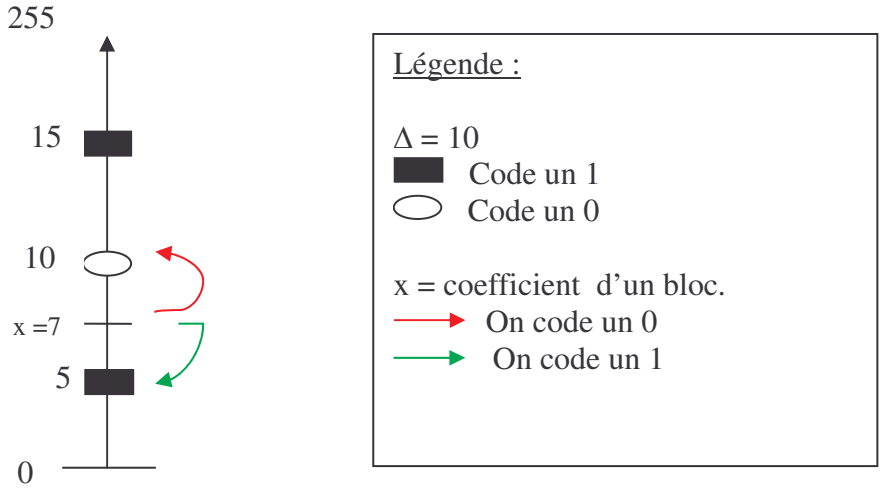
Y= (7/ 10) - E(7 / 10)
Y = 0,7
Donc Y > ½ d'où Q(x) = Q(7) = 10*(0+1)
Donc Q(x) = 10

1ier Cas : M = 0 :

On veut coder un O et on a donc :
s = Q(x) = 10

2ieme Cas : M= 1:

On veut coder un 1 et on a donc :
s = Q(x+5) – 5 = Q(12) -5
Q(12) = 10 (On obtient ce résultat à l'aide des formules du (1) avec Y(12))
D'où
s = 5
Tout ceci se résume à la figure ci-dessous.



Echelle de la luminance (0→ 255)

On va réitérer cette opération sur tous les coefficients de luminance de chaque bloc. Ainsi on va insérer 1 bit d'information par bloc, et finalement une suite de bit dans une image.

3.5.4 Phase de Décodage :

3.5.4.1 Théorie

Le décodage se fait en aveugle, à partir de l'image tatouée seule. On décompose l'image en blocs de taille 8x8.

On rappelle que dans chaque bloc, un bit d'information est inséré. Pour un bloc donné B, le décodage se fait comme suit :

• On effectue une quantification de tous les coefficients de B à l'aide des deux quantificateurs Q0 et Q1.

• On arrange les coefficients quantifiés de B suivant deux ensembles M1 et M2 tels que :

Soit un coefficient quelconque $c \in B$, c1 dénote le point résultant de la quantification de c par Q0, c2 dénote le point résultant de la quantification de c par Q1. Dans la suite la distance entre deux points désigne la distance usuelle. C'est-à-dire que la distance entre a et b est définit par la valeur absolue de leur différence.

Si (distance(c,c1) < distance(c,c2)) alors $c \in M1$ sinon $c \in M2$.

• Si (cardinal M1 >> cardinal M2) on conclut que le bit inséré dans B est égal à 0.

• Si (cardinal M2 >> cardinal M1) on conclut que le bit inséré dans B est égal à 1.

• Si (cardinal M1 ≈ cardinal M2), le bloc n'est pas tatoué.

On effectue le décodage sur tous les blocs, et on compose un message binaire. Afin d'éviter le cas ou les blocs ne sont pas tatoués on a effectué du bourrage dans le message à insérer. Ce bourrage est donc substitué lors du décodage, et on retrouve bien le message inséré. Ce qui implique que nous devons connaître la taille du message à insérer avant de décoder. La phase de bourrage n'est pas obligatoire mais elle permet de simplifier la phase des tests. Par conséquent, il est tout à fait possible d'effectuer cet algorithme sans connaître la taille du message à insérer.

On peut résumé la phase de décodage avec la figure ci-dessous.

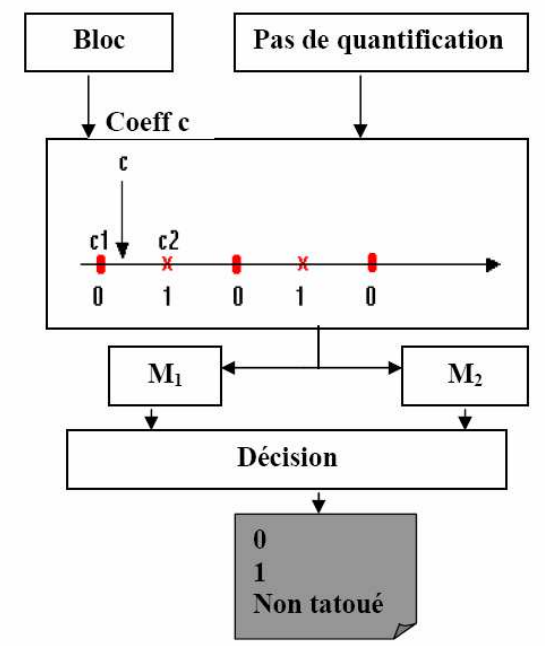


Fig 1 : Extraction d'un bit inséré dans un bloc 8x8

3.5.4.2 Notre décodage :

On a fixé le seuil de décodage à 40. C'est-à-dire qu'il faut que le cardinal de M1 ou M2 soit supérieur ou égal à 40 pour qu'un bloc soit marqué. Un bloc est de taille 8x8, il comporte donc 64 coefficients, ce qui veut dire qu'il faut que 62,5 % des coefficients du bloc dans M1 ou M2 pour que celui-ci soit marqué.

Décodage du message M dans le bloc B :
Seuil =40 ;
Pour toutes les valeurs s du bloc B :
s1 = Q (s)
s2 = Q (s + Δ / 2) - (Δ / 2)
si (distance (s , s1) < distance (s , s2)) alors s est un élément de M1, sinon, s est un élément de M2.
si cardinal (M1) > seuil, alors le bloc B code un 0.
si cardinal (M2) > seuil, alors le bloc B code un 1.

Prenons un exemple simple. On se place dans les hypothèses de l'exemple du codage, vu précédemment. On a donc :
Δ = 10, et s = 15 qui code un 1 (cf schéma du codage, la valeur 15 code un 1)
On a :
s1 = Q (15) = 10
s2 = Q (20) – 5 = 15
D'ou distance(s , s2) = 0 < 5 = distance(s , s1).
On a donc s ∈ M2, et s code bien un 1.

Commentaire :

Dans notre exemple, notre s était déjà positionné sur une valeur qui codait 1. Il est donc logique que l'on retrouve 1 lors du décodage. En réalité, notre image, contenant le bloc, a pu subir des transformations, et par conséquent la valeur de s a pu être décalée un peu comme on peut le voir sur la Figure 1 ci-dessus. Ainsi la valeur de s se trouve entre deux valeurs de la quantification, « proche » de sa valeur initiale et « loin » de la valeur de quantification suivante. On évalue la distance de s par rapport a ces deux valeurs et on en déduit, en prenant la valeur la plus proche, ce que code s. Si notre image après avoir été codé ne subit aucune transformation, les blocs sont déjà quantifiés et on trouvera alors que le cardinal de M1 (ou respectivement M 2) est égal à 64 (bloc de 8x8 d'ou 64 valeurs).

Nous avons voulu évaluer le pourcentage d'erreurs commises lors de la phase de décodage. L'algorithme de décodage va faire des erreurs, il va extraire un bit d'un bloc qui à l'origine n'en contient pas. On parle de faux positifs. Pour notre algorithme, on a fixé le seuil à 40. C'est-à-dire qu'il faut 40 valeurs de 1 ou 0 pour que l'algorithme de décodage donne en sortie un bit. Pour avoir un faux positif dans un bloc, il faut donc que l'on ait plus de 40 ou moins de 24 valeurs dans les ensembles M1 ou M2. En effet, si l'on le cardinal de M1 est de 40 alors le cardinal de M2 est de 24 étant donné que l'on a 64 valeurs. Pour évaluer les faux positifs, on pose X une variable aléatoire qui suit une loi de Bernoulli de paramètre p = ½. On a donc :

$$P(X = 1) = p \quad (= 1/2)$$
$$P(X = 0) = 1-p \quad (= 1/2)$$

Soit Y une variable aléatoire telle que : $Y = \sum_{i=1}^{n} X_i$

Alors Y suit une loi binomiale de paramètre n et p (= ½).
D'où :

$$P(Y = k) = C_n^k p^k (1-p)^{n-k} \text{ pour } 0 \le k \le n$$

En remplaçant, avec :

n = 64 (nombre de coefficients d'un bloc),

p = 1/2,

k = 40 (seuil de décodage),

On obtient une erreur de l'ordre de 6 %.

Notons, qu'une erreur se produit si le un bloc non marqué donne le cardinal d'un ensemble M1 ou M2, supérieur à 40 ou inférieur à 24. Lorsque nous avons remplacé les paramètres dans la formule, nous avons multiplié le résultat par 2, pour les deux cas de figure, le cardinal supérieur à 40 et inférieur à 24.

Hors en pratique on peut constater que ce taux est bien plus important, autour de 25 % pouvant monter jusqu'à 70 %, ceci étant peut être dû à la corrélation des pixels. Notre calcul d'erreur ne tient pas compte de cette corrélation qui joue un rôle important dans la présence de faux positifs.

CODE COLLUSION-SECURE
Boneh & Shaw

4. Les codes collusion-secure : Boneh & Shaw

4.1 Introduction

Après avoir vu l'algorithme de watermarking utilisé, Costa Scalaire, nous allons nous attacher aux codes collusion-secure.

Tout d'abord nous allons évoquer le « fingerprinting ». Le fingerprinting est une technique qui consiste à marquer et enregistrer chaque contenu distribué. Cette marque va permettre au distributeur du contenu de détecter tout copie non autorisée et de tracer l'utilisateur qui l'a diffusé. Le traçage s'effectue avec un algorithme de traçage qui prend en entrée un contenu ou une marque et donne en sortie l'utilisateur associé. Cependant, un problème apparaît, quant plusieurs utilisateurs se regroupent et tentent d'attaquer le contenu. Ce regroupement de pirates, se nomme collusion. Ces pirates vont mettre en commun leurs contenus et les comparer afin de connaître l'emplacement de leur marque et ainsi tenter de l'effacer. Ils pourront ainsi altérer le fingerprint pour masquer leurs identités. Ils pourront donc diffuser illégalement le contenu modifié. C'est afin de lutter contre ce type d'attaque que nous allons étudier les codes collusion-secure.

Tout le long de ce chapitre, nous allons expliquer la démarche qui nous a amenée aux codes collusion-secure. C'est-à-dire que nous allons partir des codes c-frameproof en passant par les codes c-secure pour arriver aux codes collusion-secure.

4.2 Algorithme de Boneh & Shaw

On désignera par Σ un alphabet de taille S, qui représente les S différents états de la marque. Σ peut être représenté par les entiers de 1 à S.

Etant donné un mot de L bits $w \in \Sigma^L$ et l'ensemble $I = \{i1,i2,i3,\ldots,ir\} \subset \{1,\ldots,L\}$, on notera par $w|_I$ le mot $w_{i1}\, w_{i2}\, w_{i3} \ldots w_{ir}$ ou w_i est la ième lettre de w. Donc $w|_I$ est une restriction de w pour les positions appartenant à I.

Définition 1 : (L-n)-code :

Un ensemble $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\} \subset \Sigma^L$ est dit (L-n)-code. Le $w^{(i)}$ est fournit a l'utilisateur u_i , pour $1 \leq i \leq n$. On dira que l'ensemble des mots Γ forme un dictionnaire. $1 \leq i \leq n$

Définition 2 : Marque indétectable :

Un ensemble $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\}$ et C une coalition d'utilisateurs.
Pour $1 \leq i \leq n$, on dira que la ième position est indétectable pour C si les mots fournis aux utilisateurs de C sont identiques à la ième position.

Par exemple, supposons $C = \{ u_1, \ldots, u_c\}$. Alors la position i est indétectable si :
$W_i^{(u_1)} = W_i^{(u_2)} = \ldots = W_i^{(u_c)}$

Remarquons que si la coalition C peut détecter la ième position de la marque, elle peut générer un contenu avec une marque dont la ième position n'est dans aucun des S états. La coalition C peut donc rendre illisible une marque. On notera une marque illisible par « ? ».
On voit apparaître la notion d'ensemble faisable.

Définition 3 : Ensemble faisable

Soit $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\}$ un (L-n)-code et C une coalition d'utilisateurs. Soit R l'ensemble des positions indétectables pour C. On définit alors un ensemble faisable pour C par :
$F(C ; \Sigma) = \{ w \in (\Sigma \cup \{ ?\})^L$ i.e $w|_R = w^{(u)}|_R \}$
Ceci pour chaque utilisateur u de C.

On note que cet ensemble contient aussi les bits indétectables. Dans la suite on notera $F(C ; \Sigma)$ par F(C). De plus, on notera des fois $\Sigma \cup \{ ?\}$ par Σ'.

Exemple : Si deux utilisateurs A et B ont pour codes :

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

Leur ensemble faisable sera $F(AB) = \Sigma'.2\ .\Sigma'.1\ .2.$

Si le code utilisé pour marquer un contenu est caché aux utilisateurs, le distributeur des codes d'identifications n'a presque plus rien à faire. Pour chacun des n utilisateurs, il leur attribue un code unique choisi au hasard. Une coalition ne peut compromettre un utilisateur innocent. En revanche ce que nous cherchons, c'est construire un code c-frameproof même si les utilisateurs connaissent le dictionnaire.

Définition 3 : Code c-frameproof :
Un code Γ est dit c-frameproof si pour tout ensemble $W \subset \Gamma$, de taille c, on a :

$$F(W) \cap \Gamma = W$$

C'est-à-dire qu'avec un code c-frameproof, l'ensemble faisable par la coalition, d'au plus c utilisateurs, est composé des mots des membres de la coalition. Il n'existe donc pas de coalition d'au plus c utilisateurs qui peut compromettre un innocent.

A- Construction d'un code c-frameproof

On va construire un code c-frameproof à partir d'un alphabet binaire $\Sigma = \{0,1\}$.

On va introduire un simple (n,n)-code qui est n-frameproof. On définit le code $\Gamma_0(n)$ par un (n,n)-code ayant n mots binaires ayant chacun exactement un 1.Par exemple, le code $\Gamma_0(3)$ pour trois utilisateurs est {100,010,001}.

Corollaire :
$\Gamma_0(n)$ est un (n,n)-code qui est n-frameproof.

Il n'est pas difficile de voir que tout code n-frameproof doit être de taille au moins égale à n. Ce qui signifie que toute coalition de n-1 utilisateurs ne doit pas être capable de détecter au moins un bit de position, autrement dit ils peuvent compromettre un utilisateur qui n'est pas dans la coalition. La taille de Γ_0 est linéaire par rapport au nombre d'utilisateurs et c'est pour cela que c'est impraticable. Nous allons utiliser le code Γ_0 pour construire un code plus court.

Définition : Code détecteur d'erreur
Un ensemble C de N mots de taille L construit sur un alphabet de p lettres est dit $(L,N,D)_p$ –Error-Correcting Code noté $(L,N,D)_p$-ECC si la distance de Hamming entre chaque paire de mots de C est d'au moins D.(Annexe [1])

L'idée de cette construction repose sur la combinaison du code $\Gamma_0(n)$ avec un Code détecteur d'erreur. Soit $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\}$ un (l ,p)-code et C un (L,N,D)-ECC. On note la combinaison de Γ et C par Γ'. Le code Γ' est un (lL, N)-code définit par :
Soit $v = v_1 v_2 \ldots v_L \in C$ on a :
$W_v = w^{(v_1)} \| w^{(v_2)} \| \ldots \| w^{(v_L)}$
Ou || désigne la concaténation de chaînes de caractères.

Γ' est l'ensemble de tous les mots W_v , c'est-à-dire :
$\Gamma' = \{ W_v \mid v \in C \}$

Lemme A 1 :
Soit Γ un (l,p)-code c-frameproof et C un $(L,N,D)_p$-ECC. Soit Γ' la combinaison de Γ et C. Alors Γ' est un code c-frameproof avec $D > L(1-(l/c))$.

Preuve :
Posons C une coalition de c utilisateurs .Nous allons montrer que $F(C ; \Gamma')$ ne contient pas des mots de Γ' à part ceux de C.
Soit $v^1, \ldots, v^c \in C$ les mots de C à partir des quels ils vont forger un nouveau mot.
Supposons par l'absurde qu'un mot $W \in \{0,1\}^{lL}$ de $F(C ; \Gamma')$ appartient à un utilisateur $u \notin C$.
Supposons $z \in C$ le mot du code à partir du quel W a été forgé. Pour tout k = 1…c le mot z et v^k sont identiques dans moins de L/c positions. Ce qui implique que la distance de Hamming minimale de C est supérieur à que $L(1-(l/c))$. Par conséquent il existe une position j tel que : $1 \le j \le L$ et $z_j \neq v_j^k$ pour tout k = 1,…,c.
On pose $C_j = \{ w^{v^1{}_j}, \ldots, w^{v^c{}_j} \}$. De plus, Γ est un code c-frameproof , nous savons que w^{z_j} n'est pas dans l'ensemble $F(C_j ; \Gamma')$. On a w^{z_j} qui est la partie d'un mot de W, ce qui implique que $W \notin F(C ; \Gamma')$. Ceci prouve la contradiction.

On peut noter que les conditions sur la distance minimale de C peuvent être moindre mais on ne s'en préoccupera pas. Pour plus d'information on peut se référer a [2] de l'annexe.

Lemme 2 :
Pour tout entiers positifs p et N, on pose L= 8p log(N).
Alors il existe un $(L,N,D)_{2p}$ –ECC avec $D > L(1-(l/c))$.

Théorème :
Pour tout entiers n et c > 0 on pose $l = 16c^2 \log(n)$.
Il existe un (l,n)-code qui est c-frameproof.

Preuve :
Avec le Lemme 2 on sait qu'il existe un Code détecteur d'erreur de paramètres $(L,n, L(1-l/c)_{2c}$ avec L = 8c log(n). En combinant ceci avec le code $\Gamma_0(2c)$ et le Lemme 1 on obtient un code c-frameproof pour n utilisateurs de taille $2cL = 16c^2 \log(n)$

Une construction explicite de ce code est décrite dans [3] de l'annexe.

B- Construction d'un code c-Secure :

Recentrons le problème sur les codes collision-secure. On suppose qu'un distributeur marque un objet avec un code Γ. On suppose qu'une coalition d'utilisateurs C, génère un contenu à partir de quelques mots x et distribue ce contenu non enregistré. Quand ce contenu est trouvé par le distributeur, le distributeur souhaiterait trouver un sous ensemble de la coalition qui a permis de le forger. C'est à dire qu'il faut qu'il existe un algorithme de traçage. Celui ci prendrait en entrée x et donnerait en sortie un membre de la coalition. On peut voir cela comme une fonction $A : \{0,1\}^n \rightarrow \{1,\ldots,n\}$ ou n est le nombre d'utilisateurs.

Définition : Code totalement c-secure
Un code Γ est dit totalement c-secure si il existe un algorithme de traçage A tel que :
Si une coalition d'au plus c utilisateurs forge un mot x alors $A(x) \in C$.

L'algorithme A prend en entrée x et donne un membre de la coalition. Par conséquent une copie illégale peut fournir au moins un membre de la coalition coupable. Nous ne cherchons pas à retrouver tous les membres mais au moins un.

Nous allons donner une condition nécessaire sur le code pour qu'il soit totalement c-secure. On considère le scénario suivant, Γ un code. Soit C_1 et C_2 être deux coalition de c utilisateurs chacune, avec $C_1 \cap C_2 = \Phi$. On suppose qu'un contenu est marqué par le mot x qui appartient à l'ensemble faisable par C_1 et C_2. Donc les deux coalitions sont suspectes mais leur intersection est vide. Il est donc impossible d'être sûr de savoir qui a crée le contenu. C'est pour cela que Γ est dit totalement c-secure si on a que l'intersection de C_1 et C_2 entraîne que les ensembles $F(C_1)$ et $F(C_2)$ sont vides.

Lemme 1 :
Si Γ est un code totalement c-secure, alors :
$C_1 \cap \ldots \cap C_r = \phi \Rightarrow F(C_1) \cap \ldots \cap F(C_r) = \phi$ pour r fixé.
Pour toutes coalitions $C_1, \ldots, C_r$ d'au plus c utilisateurs chacune.

Les codes totalement c-secure sont donc une bonne solution pour le problème des collusion. Malheureusement, quand c > 1, un code totalement c-secure n'existe pas.

Théorème 1:
Pour $c \geq 2$ et $n \geq 3$ il n'existe pas de (l,n)-code totalement c-secure.

Preuve :
On va juste montrer qu'il n'existe pas de code totalement 2-secure.
Soit Γ un (l,n)-code. Soit $w^{(1)}$, $w^{(2)}$,$w^{(3)}$ trois marques distinctes fournies aux utilisateurs u_1, u_2, et u_3 respectivement. On définit la fonction de la majorité par :
$M = MAJ (w^{(1)} , w^{(2)} ,w^{(3)})$

$$M_i = \begin{cases} w^{(1)}_i & \text{si } w^{(1)}_i = w^{(2)}_i \text{ ou } w^{(1)}_i = w^{(3)}_i \\ w^{(2)}_i & \text{si } w^{(2)}_i = w^{(3)}_i \\ ? & \text{sinon} \end{cases}$$

On peut lire que le mot M forger par la fonction MAJ est faisable par les trois coalitions $\{u_1,u_2\}$, $\{u_1,u_3\}$ et $\{u_2,u_3\}$. Cependant, l'intersection est vide. Par conséquent, on peut conclure à l'aide du Lemme 1, que le code Γ n'est pas totalement 2-secure.

Ce que nous venons de voir montre que si une coalition utilise comme fonction de forge la fonction « majorité » elle est sûr de déjouer tous les codes fingerprint. Le technique de fingerprint n'est donc pas possible pour lutter contre les collusions. Heureusement, il existe un moyen pour remédier à ce problème, il faut utiliser l'aléatoire.

Nous allons affaiblir nos exigences du Théorème 1 pour la phase de marquage. Le distributeur va utiliser un choix aléatoire lors de la phase d'insertion de la marque. Ce choix aléatoire devra être gardé secret par le distributeur. Ainsi, nous allons construire un code qui va permettre de capturer les pirates avec une probabilité forte.
Un (l,n)-fingerprinting scheme est une fonction $\Gamma(u, r)$ avec le nombre u d'utilisateurs $1 \leq u \leq n$ et une partie aléatoire de bits r tel que $r \in \{0,1\}^*$ avec un code Σ^l. La partie aléatoire r est un ensemble de bits aléatoires choisis par le distributeur et gardés secrets. On notera ce plan de fingerprinting par Γ_r.

Supposons qu'une coalition C de c utilisateurs crée une copie illégale d'un contenu. L'algorithme de fingerprint capable de capturer un membre de la coalition C avec une probabilité d'au moins 1- ε est appelé **code c-secure** avec une erreur ε. Ici, la probabilité d'erreur est prise par rapport au choix aléatoire utilisé par le distributeur et le choix aléatoire de la forge utilisé par la coalition.

Définition : Code c-secure avec une erreur ε
Un algorithme de fingerprinting Γ_r est c-secure avec une erreur ε, si il existe un algorithme de traçage A qui satisfait la condition suivante :
Si une coalition C d'au plus c utilisateurs, forge un mot x alors :

$$Pr[A(x) \in C] > 1- \varepsilon$$

Avec la probabilité basée sur le choix aléatoire des bits r et du choix aléatoire induit par la coalition.

L'algorithme de traçage A prend en entrée x, et donne en sortie un membre de la coalition C qui a permis de forger le mot x, avec une forte probabilité.

C- Construction d'un code collusion-Secure :

Nous allons d'abord construire un (l,n)-code qui est n-secure. Même pour une taille importante de coalition on sera capable de tracer les membres de la coalition avec une probabilité forte. Cependant la taille du code sera en $n^{O(1)}$ et par conséquent, trop grand pour être praticable. Nous allons donc montrer comment construire un code c-secure pour n utilisateurs avec une taille en $\log(n)^{O(1)}$ avec c = O(log(n)).

Nous allons présenter un (l,n)-code qui est n-secure avec une erreur ε, pour tout ε > 0. Soit c_m la colonne de longueur n dont les m premiers bits sont 1 et le reste 0. Le code $\Gamma_0(n,d)$ est fait de toute les colonnes $c_1,\ldots,c_{n-1}$ chacune dupliquée d fois. La valeur de duplication détermine la probabilité d'erreur ε .
Par exemple : $\Gamma_0(4,3)$ pour quatre utilisateurs A,B,C,D est

A : 111 111 111
B : 000 111 111
C : 000 000 111
D : 000 000 000

Soit $w^{(1)},\ldots, w^{(n)}$ les mots du codes de $\Gamma_0(n,d)$. Avant que le distributeur insère un mot du code $\Gamma_0(n,d)$ dans un contenu, il doit faire le choix aléatoire suivant :
Le distributeur prend au hasard une permutation $\Pi \in S_l$.
Ou S_l désigne l'ensemble des groupes de permutation symétrique de l lettres.
Pour $x \in \{0,1\}^l$, soit une permutation $\Pi \in S_l$, on notera par
Πx le mot de l bits : $x_{\Pi(1)}\ x_{\Pi(2)} \ldots\ x_{\Pi(l)}$
Ensuite le distributeur insèrera pour chaque utilisateur u_i le mot $\Pi x^{(i)}$.
On remarque que c'est la même permutation qui est utilisé pour chaque utilisateur. Le distributeur doit garder secret la permutation utilisée.

Théorème C.1 :
Pour n > 3 et ε > 0 et $d = 2\,n^2 \log(2n/\varepsilon)$ on a :
L'algorithme de fingerprinting $\Gamma_0(n,d)$ est n-secure avec une erreur ε.
La taille de ce code est égale à $d(n-1) = O(n^3\log(n/\varepsilon))$

Pour prouver ce théorème, nous allons décrire un algorithme qui prend en entrée un mot x forgé par la coalition C, et qui donne en sortie un membre de la coalition avec une probabilité de 1- ε.
Nous allons introduire quelques notations :

1- Soit B_m l'ensemble des positions des bits dans lesquelles les utilisateurs ont des colonnes de type c_m. C'est-à-dire que B_m est l'ensemble de toutes les positions des bits dans lesquelles les premiers m utilisateurs ont un 1 et le reste à 0. Le nombre d'éléments de B_m est d.
2- Pour $2 \le s \le n-1$ on pose $R_s = B_{s-1} \bigcup B_s$.
3- Pour tout mot binaire x, on pose w(x) comme étant le nombre de 1 dans x.

Avant de décrire l'algorithme, on va en donner une idée intuitive. On suppose que l'utilisateur s n'appartient pas à la coalition. Le fait que la permutation Π soit gardée secrète permet d'éviter que la coalition ne sache quelle marque représente quel bit dans le code $\Gamma_0(n,d)$. La seule information que peut avoir la coalition, c'est la valeur des marques

qu'elle peut détecter. On remarque, que sans l'utilisateur s une coalition va avoir la même valeur pour toutes les positions des bits $i \in R_s$. Par exemple, avec le code $\Gamma_0(4,3)$, la coalition A, C, D peut voir exactement toutes les positions des bits dans R_2. Par conséquent, pour une position de bit $i \in R_s$, la coalition C ne peut dire si i ment dans B_s ou B_{s-1}. Ce qui signifie qu'avec n'importe quelle stratégie employée pour fixer les bits de $x|_{Rs}$, le nombre de 1 de $x|_{Rs}$ va être également distribué entre $x|_{Bs}$ et $x|_{Bs-1}$, avec une forte probabilité. Par conséquent, si le nombre de 1 de $x|_{Bs}$ n'est pas également distribué alors, avec une forte probabilité, l'utilisateur s est membre de la coalition qui a permis de forger le mot x.

4.2.1 Algorithme 1 :

Etant donné $x \in \{0,1\}^1$, on va trouver un sous ensemble de la coalition qui l'a forgé.

1- Si $w(x|_{B_1}) > 0$ alors « l'utilisateur 1 est coupable »

2- Si $w(x|_{B_{n-1}}) > 0$ alors « l'utilisateur n est coupable »

3- Pour tout s = 2 à n-1 faire :

Soit $k = w(x|_{R_s})$, si

$$w(x|_{B_{s-1}}) < \frac{k}{2} - \sqrt{\frac{k}{2}\log(\frac{2n}{\varepsilon})}$$

Alors « l'utilisateur s est coupable ».

On peut noter que le mot x trouvé dans la copie illégale peut contenir des marques illisibles « ? ». Par convention, nous allons fixer ces bits à 0 avant de fournir le mot x à l'algorithme 1. Maintenant, nous allons prouver l'algorithme 1 avec ce qui suit.

Lemme 1 :
On considère le code $\Gamma_0(n,d)$ ou $d = 2n^2\log(2n/\varepsilon)$
Soit S l'ensemble d'utilisateurs que l'Algorithme 1 a donné coupable en prenant comme entrée le mot x. Alors, avec une probabilité de $1-\varepsilon$, l'ensemble S est un sous ensemble de la coalition C qui a forgé x.

Preuve :
Supposons que l'utilisateur 1 est donné coupable, c'est-à-dire que $1 \in S$. Alors $w(x|_{B_1}) > 0$. Ceci implique que l'utilisateur 1 doit être un membre de la coalition C.

Sinon les bits de B_1 auraient été indétectable pour C ce qui implique que $w(x|_{B_1}) = 0$.

De façon similaire, si $n \in S$ alors $n \in C$.

Supposons que l'algorithme 1 prononce l'utilisateur $1 < s < n$ comme coupable. Nous allons montrer que la probabilité que s ne soit pas coupable est au plus ε/n.
Ce qui va montrer que la probabilité qu'il existe un utilisateur dans S qui n'est pas coupable est d'au plus ε.
Soit s un utilisateur innocent, c'est-à-dire que $s \notin C$. Ce qui veut dire que la coalition C ne peut détecter la position des bits de R_s. De plus, la permutation Π a été choisie au hasard parmi l'ensemble de toutes les permutations, les positions des 1 de $x|_{R_s}$ peuvent être vu

comme des positions aléatoires de $x|_{R_s}$. On pose k = w($x|_{R_s}$). On définit Y comme une variable aléatoire qui compte le nombre de 1 de $x|_{B_{s-1}}$ sachant que $x|_{R_s}$ contient k 1.
Pour tout entier r dans l'intervalle approprié :

$$\Pr[Y = r] = \frac{\binom{M}{r}\binom{M}{k-r}}{\binom{2M}{k}}$$

Ou $M = 2n^2\log(2n/\varepsilon)$ qui est la taille de B_{s-1} .Il est évident que l'on attend que Y soit égale à k/2. Pour borner la probabilité que s soit donné coupable, nous avons besoin de borner :

$$\Pr\left[Y < \frac{k}{2} - \sqrt{\frac{k}{2}\log(\frac{2n}{\varepsilon})} \right]$$

Ceci va être fait en comparant Y à une variable aléatoire appropriée.
Soit X une variable aléatoire représentant k expériences avec une probabilité de succè de ½. Une fonction qui calcule ceci montre que pour tout r on a : Pr[Y = r] ≤ 2Pr[X = r].
Ce qui signifie que pour tout a > 0 on a :

$$\Pr\left[Y - \frac{k}{2} < a \right] \le 2\Pr\left[X - \frac{k}{2} \right] \le 2e^{-2a^2/k}$$

Ou la dernière inégalité vient de la borne de Chernoff . En prenant a = $\sqrt{\frac{k}{2}\log(\frac{2n}{\varepsilon})}$ on obtient :

$$\Pr\left[Y < \frac{k}{2} - \sqrt{\frac{k}{2}\log(\frac{2n}{\varepsilon})} \right] \le 2e^{-\log(2n/\varepsilon)} = \frac{\varepsilon}{n}$$

(Annexe [4])
Par conséquent la probabilité que des innocents soit donné coupables est d'au plus ε .

Lemme 2 :
On considère le code $\Gamma_0(n,d)$ avec $d = 2n^2\log(2n/\varepsilon)$
Soit S être l'ensemble des utilisateurs que l'Algorithme 1 a donné coupables en prenant en entrée le mot x. Alors l'ensemble S n'est pas vide.

Pour prouver ce Lemme on va d'abord introduire une proposition.

Proposition 1:
Supposons que S est vide. Alors pour tout s on a :

$$w(x|_{B_s}) \le 2\, s^2\log(2n/\varepsilon).$$

Preuve (proposition 1) :
On va prouver la proposition 1 par récurrence sur s.
Pour s =1, la proposition est vérifiée étant donné que si l'utilisateur 1 n'est pas coupable alors $w(x|_{B_1}) = 0$.

Maintenant, on suppose que la proposition est vraie pour tout s < n-1 et on va démonter qu'elle reste vraie pour s + 1.
On pose :

$$k = w(x|_{B_s})$$
$$k' = w(x|_{B_{s+1}})$$
$$t = w(x|_{R_{s+1}})$$

Ce qui découle de ces 3 conditions est :

$$t = k + k'$$
$$k \le 2s^2 \log(2n/\varepsilon)$$
$$k \le \frac{t}{2} - \sqrt{\frac{t}{2}\log(\frac{2n}{\varepsilon})}$$

La première condition vient du fait que $R_{s+1} = B_s \cup B_{s+1}$. La seconde condition vient de l'hypothèse. Et la troisième vient du fait que l'utilisateur a été donné coupable. Nous allons montrer que ces trois conditions implique que $k' \le 2(s+1)^2 \log(2n/\varepsilon)$ ce qui prouvera la proposition.

$$k' = t - k \le \frac{t}{2} + \sqrt{\frac{t}{2}\log\frac{2n}{\varepsilon}}$$
$$= \frac{k+k'}{2} + \sqrt{\frac{1}{2}(k+k')\log(\frac{2n}{\varepsilon})}$$
$$\le \frac{2s^2\log(2n/\varepsilon)+k'}{2} + \sqrt{\frac{1}{2}(2s^2\log(\frac{2n}{\varepsilon}+k')\log(\frac{2n}{\varepsilon})}$$

D'où

$$k' \le 2s^2\log(\frac{2n}{\varepsilon}) + \sqrt{2(2s^2\log(\frac{2n}{\varepsilon}+k')\log(\frac{2n}{\varepsilon})}$$

Supposons que $k' = 2r^2 \log(2n/\varepsilon)$ pour une constante r fixée. On substitue cette expression pour k' et on divise par $2\log(2n/\varepsilon)$ on obtient :

$$r^2 \le s^2 + \sqrt{s^2 + r^2}$$

Ce n'est pas difficile de voir que cette inégalité est satisfaite pour :
$s \ge 1$ et que l'on doit avoir $r \le s+1$.
Par conséquent :

$$k' \le 2(s+1)^2 \log(2n/\varepsilon)$$

Preuve du Lemme 2 :
On suppose S est vide. Ce qui veut dire que l'utilisateur n n'est pas donné coupable. De plus, on a :

$$w(x|_{B_{n-1}}) = d = 2\, n^2 \log(2n/\varepsilon).$$

Or d'après la proposition précédente pour s = n-1 on a

$$w(x|_{B_{n-1}}) \leq 2\,(n-1)^2 \log(2n/\varepsilon).$$

D'ou la contradiction, ce qui prouve le Théorème C.1.

D- Construction d'un code c-Secure de taille logarithmique :

Le code n-secure construit auparavant va nous servir à construire un (n,l)-code c-secure de taille $l = c^{O(1)}\log(n)$. Nous allons montrer à partir de l'article [8] une technique relativement simple, pour construire un code c-secure assez court.

Soit C un (L,N)-code construit sur un alphabet de taille n ou les mots du code sont choisis aléatoirement indépendamment et uniformément. L'idée est de combiner notre code n-secure $\Gamma_0(n,d)$ avec le code C comme nous l'avons fait dans le Lemme A .1. On appelle le résultat $\Gamma'(L,N,n,d)$. Ce code contient N mots de taille Ld(n-1). Il se compose de L copie de $\Gamma_0(n,d)$. Nous nous réfèrerons à ces copies comme composantes de $\Gamma'(L,N,n,d)$. Le point important est que les mots du code C resteront cachés aux utilisateurs. On va donc cacher les L permutations utilisées quand nous insèrerons les L copies de $\Gamma_0(n,d)$ dans le contenu.

Théorème D.1 :

Etant donné N, c et $\varepsilon > 0$, on pose n = 2c, $L = 2c \log(2N/\varepsilon)$ et $d = 2n^2\log(4nL/\varepsilon)$. Alors $\Gamma'(L,N,n,d)$ est code c-secure avec une erreur ε. Le code contient N mots de taille

$$l = O(Ldn) = O(c^4 \log(N/\varepsilon)\log(1/\varepsilon))$$

Pour prouver ce théorème, nous allons exposer un algorithme qui trouve les coupables d'une coalition et nous allons le prouver.

4.2.2 Algorithme 2 :

Le but est, étant donner $x \in \{0,1\}^l$, trouver la coalition qui a forgée x.

1- Appliquer l'Algorithme 1 pour chaque L composante de x.
   Pour chaque composante i = 1,…,L on choisit arbitrairement une sortie de l'Algorithme 1. Soit y_i ce choix. On peut noter que y_i est un nombre entre 1 et n.
   Ensuite on forme le mot $y = y_1 \ldots y_L$
2- Trouver le mot $w \in C$ qui correspond le plus à y dans le plus de positions possibles.
3- Soit u l'utilisateur dont le mot du code est dérivé de $w \in C$ et donner en sortie « l'utilisateur u est coupable ».

Lemme D 1:
Soit x le mot produit par la coalition C d'au plus c utilisateurs. Alors avec les paramètres du Théorème D.1, l'Algorithme 2 donne un membre de la coalition avec une probabilité d'au moins $1-\varepsilon$.

Preuve

Soit W l'ensemble des mots de C qui correspond aux utilisateurs dans la coalition C. Pour tout $1 \le i \le L$, l'Algorithme 1 garantit que y_i correspond à w_i pour un w∈ W avec une probabilité de 1- ε/2L. Ceci vient du choix de d et du fait de la composante i des utilisateurs de C qui ont les mots de Γ_0(n,d). Avec les conditions ci dessus satisfaites, on trouve chaque composante avec une probabilité au moins égale à 1- ε/2. Nous allons appeler cela l'évènement A.

On rappelle que la taille de W est au plus c. Par conséquent, quant l'évènement A arrive, il existe un mot w∈ W qui correspond à y dans L/c positions. Cependant, les mots de C sont choisis aléatoirement, et cachés aux utilisateurs, donc tout mot de C qui n'est pas dans W est attendu avec une correspondance avec y de L/n = L/2c positions. En utilisant les bornes de Chernoff, il est possible de montrer que la probabilité qu'un mot aléatoire puisse correspondre à y dans L/c positions est inférieur à ε/2N. Par conséquent, la probabilité qu'un mot appartienne à C \ W qui correspond à y dans L/c positions est au plus ε/2. Ce qui montre que quand l'évènement A arrive, l'algorithme donne un membre de C avec une probabilité d'au moins 1- ε/2. En combinant cela avec le fait que l'évènement A arrive avec une probabilité d'au moins 1- ε/2 on prouve le lemme.

Remarque :

Théorème : Borne inférieure

Soit Γ un (l,n)-code de fingerprinting sur un alphabet binaire. On suppose Γ c-secure avec une erreur ε. Alors la taille du code est d'au moins $l \ge \frac{1}{2}(c-3)\log(\frac{1}{\varepsilon c})$.

La démonstration de ce théorème n'apporte rien de fondamentale à la compréhension du sujet, c'est pour cela que nous ne la détaillerons pas ici. En revanche, vous pouvez consulter l'article de Bone&Shaw [8] pour tous les détails de la démonstration.

5 Travail Réalisé et difficultés rencontrées

5.1 Travail Réalisé

En premier lieu, nous avons dû choisir un type de contenu pour élaborer notre étude. Nous avons opté pour les **images**, après une étude de la norme MPEG 4 qui aurait permis de travailler sur des films. Pour des questions de temps, nous avons choisi les images qui sont un type de contenu plus facile à manipuler que la vidéo. Cependant, nous pouvons élargir nos travaux aux films, en retouchant quelques parties de l'algorithme pour le passage de l'image au film. Il faut savoir que la norme MPEG 4 est orientée objet. D'où une petite difficulté au niveau du watermarking. Cette difficulté n'est pas insurmontable mais elle nous aurait fait perdre du temps étant donné que le type de contenu n'était pas primordial dans le sujet du stage.
Ensuite, nous nous sommes intéressé au choix d'un algorithme de watermarking. Au début, nous avons étudié l'algorithme basé sur la décomposition en ondelettes de Caroline Fontaine. Cependant, trop peu d'études sur cet algorithme ont été faite, et on ne connaît mal ses réactions face à des attaques. Nous avons donc cherché un autre algorithme éprouvé dont on connaît les failles. Nous avons donc choisi l'algorithme de **Costa Scalaire,** qui est bien connu dans le domaine du watermarking. De plus, cet algorithme est relativement simple dans son fonctionnement c'est aussi pour cela que nous l'avons sélectionné.
De plus, nous avons dû chercher une classe de code qui répond bien à nos exigences, à savoir qu'elle permette le traçage de pirates. Nous avons donc logiquement opté pour les codes collusion-secure et en particulier le code de Boneh & Shaw.

Lorsque j'ai commencé à développer, j'ai d'abord implémenté l'algorithme de Boneh & Shaw étant donné que nous cherchions encore un algorithme de watermarking. Durant cette période, j'ai donc développé l'algorithme de Boneh & Shaw en Java. J'ai opté pour Java pour pouvoir facilement passer de Linux à Windows sans problème pour avoir un produit multi plateformes. J'ai développé avec Scilab qui est l'équivalent de Matlab mais en version gratuite. Scilab permet le traitement d'image avec notamment sa bibliothèque SIP (Signal Image Processing).

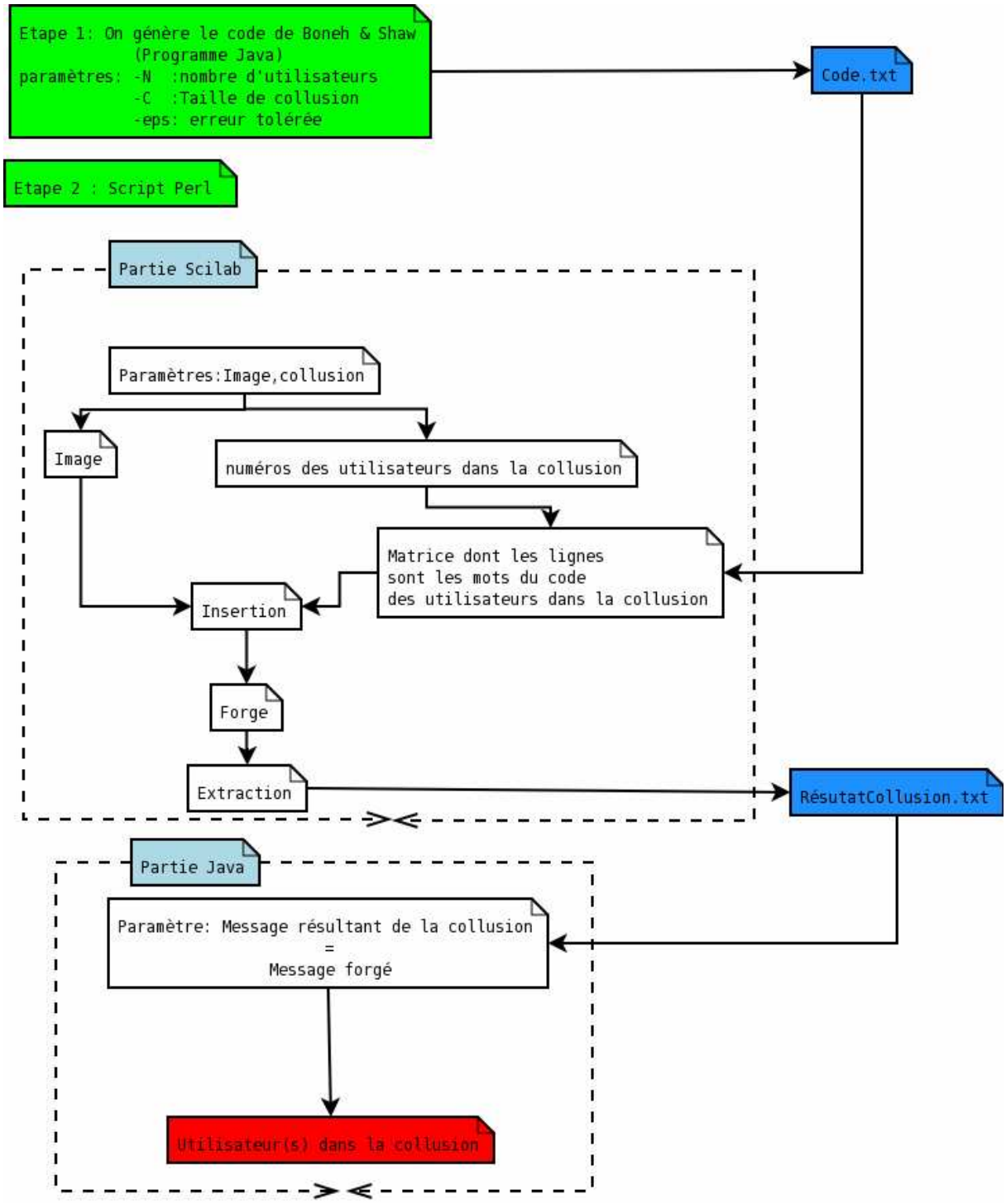
En ce qui concerne Java, j'ai développé plusieurs programmes comme CréationCode, Collusion et Traçage.

- Le programme **CreationCode** génère le code de Boneh & Shaw en prenant en paramètre le nombre d'utilisateurs, la taille des collusions et l'erreur que l'on tolère. Le résultat est stocké dans un fichier que l'on nommera *Code.txt*.
- Le programme **Collusion** génère des collusions entre les mots du code. Il existe plusieurs types de collusion, par choix aléatoire, en utilisant la Majorité, en fixant des bits à 0 ou 1, et en fonction du nombre de 0 et de 1 que l'on trouve. Ce programme Collusion, prend en entrée le type de collusion que l'on souhaite et les numéros des utilisateurs qui vont participer à la collusion. Il va chercher les utilisateurs demandés dans le fichier *Code.txt* et produit la collusion demandée. En sortie on a le résultat de la collusion qui est stocké dans le fichier *Collusion.txt.*
- Et bien sûr, le programme **Traçage** qui prend en entrée le mot forgé par une coalition et donne en sortie au moins le numéro d'une personne ayant participé à la collusion.

En ce qui concerne Scilab j'ai développé plusieurs programmes dont plusieurs fonctions qui m'ont servies afin de me fabriquer une boîte à outils.

- Le programme **CoderCosta**, permet d'insérer un message dans une image à l'aide de l'algorithme de Costa Scalaire. Ce programme prend en paramètre une image et un message pour donner en sortie l'image codée. Ce programme va avant tout vérifier si l'on peut insérer le message dans l'image. C'est-à-dire si la taille du message n'est pas trop longue. Nous avons ici, une contrainte sur la taille du message étant donné que l'on insère un bit par bloc de 8x8 de l'image.
- Le programme **DecoderCosta**, prend en paramètre l'image codée et donne en sortie le message décodé plus des bits. C'est-à-dire que si l'on a une image dans laquelle on peut insérer 160 bits et que le message originel n'est que de 150 bits les dix derniers bits vont poser problème. Normalement on a rien codé sur les dix derniers blocs, et on a choisi de coder par 2 si un bloc n'est pas marqué. On devrait donc avoir une suite de 150 bits qui représente le message décodé et 10 fois le nombre 2. Cependant il arrive qu'il y ait des faux positifs, c'est-à-dire des blocs non marqués qui donne un 1 ou 0 lors du décodage.(C'est pour cela que dans la suite nous avons effectué un bourrage de 1)
- Le programme **Forger** contient plusieurs fonctions de forges ainsi que plusieurs petites fonctions annexes. On peut trouver des fonctions de forges basées sur la FFT, la moyenne, le choix aléatoire de bloc… Les fonction annexes étaient là pour facilité la programmation, et corriger des bugs liés à Scilab lors de l'utilisation de fonctions qui lui sont propres (erreurs d'arrondis par exemple).
- Le programme **InsererCode** prend en paramètre une image et un message. Il donne en sortie la concaténation d'images codées dont la dernière image subit un bourrage de 1 si cela est nécessaire. Par exemple, je prends une image I et un message M. Supposons que dans I on peut coder 160 bits et que M est de 630 bits (630 = 3*160 + 150). Alors on aura en sortie une concaténation de 4 images dont la dernière subira un bourrage de 1 sur les 10 derniers bits (160 – 150 = 10).
- Le programme **ExtraireCode** prend en paramètre la concaténation d'images, que l'on peut considérer comme une seule image, ainsi que la taille du code à extraire. En sortie on obtient le message qui avait été inséré.
- Le programme **TraitementScilab** va permettre d'automatiser l'insertion, la forge, et l'extraction d'un message. Il prend en paramètre une matrice dont les lignes sont les mots du code des utilisateurs participant à la forge et une Image dans laquelle on va insérer ces bits. En sortie, on a le message décodé après l'utilisation d'une fonction de forge de notre choix.
- Le programme **AlgorithmeScilab** prend en paramètre une image, les numéros des utilisateurs dans la collusion et va prendre dans le fichier *Code.txt* , crée par java avec CréationCode, les utilisateurs que l'on veut, forme une matrice avec les lignes correspondantes et transmettre tout cela au programme TraitementScilab.

Je n'ai pas cité tous les programmes mais seulement les plus importants.
Ensuite j'ai développé un script Perl afin d'automatiser la communication entre les programmes écrits en Java et Scilab. Etant donné que la création du code est relativement longue, nous avons préféré ne pas l'inclure dans le script. Il faut savoir que j'ai sauté quelques étapes dans ma description des programmes pour éviter d'alourdir mon explication et la rendre plus compréhensive. Nous allons donc présenter notre développement sous forme d'un schéma qui résume le développement effectué.



Il est bien évident qu'il reste toute la partie de tests à décrire. Nous ferons cela dans le chapitre suivant.

5.2 Difficultés rencontrées

Tout d'abord, ce qui a pris du temps, le choix du contenu. Il a fallu lire la norme MPEG4 pour s'apercevoir qu'elle pourrait poser des problèmes lors de l'implémentation. Ensuite la difficulté a été de ne pas savoir sur quelle système d'exploitation j'allais développer. C'est pour cela que j'ai opté pour développer en Java qui est reconnu souple et indépendant du système d'exploitation. En revanche, Java est connu pour sa lenteur ce qui n'a pas facilité la phase de test.

En ce qui concerne l'algorithme de watermarking, comme j'avais déjà pu le constater lors du projet réalisé durant ce Master, il existe plusieurs versions de l'algorithme de Costa. Il a donc fallu choisir celle qui paraissait le plus adapté pour notre étude. Nous avons donc choisi l'algorithme de Costa Scalaire.

En ce qui concerne le choix du code anti-collusion, la recherche fût difficile étant donné que c'est un domaine qui est relativement récent. En effet, ce domaine à réellement émergé en 1998 avec l'article de Boneh & Shaw « Collusion-secure Fingerprinting for Digital Data » IEEE Transaction on Information Théory même si cet article s'inscrit dans la problématique de sujet ouvert de 1995. Il a donc était long et difficile de se procurer des articles de mathématiques traitant des problèmes liés aux collusions afin d'avoir une vu d'ensemble. Nous avons choisi l'algorithme de Boneh & Shaw pour notre implémentation, pour sa simplicité et ses caractéristiques dont la taille relativement courte de son code.

Notre contenu, à savoir une image, a imposé des contraintes sur la taille des messages que l'on peut insérer. On a donc opté pour la concaténation d'images afin d'avoir un espace suffisant pour l'insertion de message de taille arbitraire. Cependant on peut voir cette concaténation comme une seule image. Notre étude n'a donc nullement été faussée.

La création du code de Boneh & Shaw génère des mots de taille importante. Suivant les paramètres N (le nombre d'utilisateurs), c (la taille des collusions), ε (l'erreur tolérée) la taille des mots du code augmente rapidement. Lors de la création de ce code on a un facteur de duplication et la taille des collusions qui jouent beaucoup sur la taille du code.

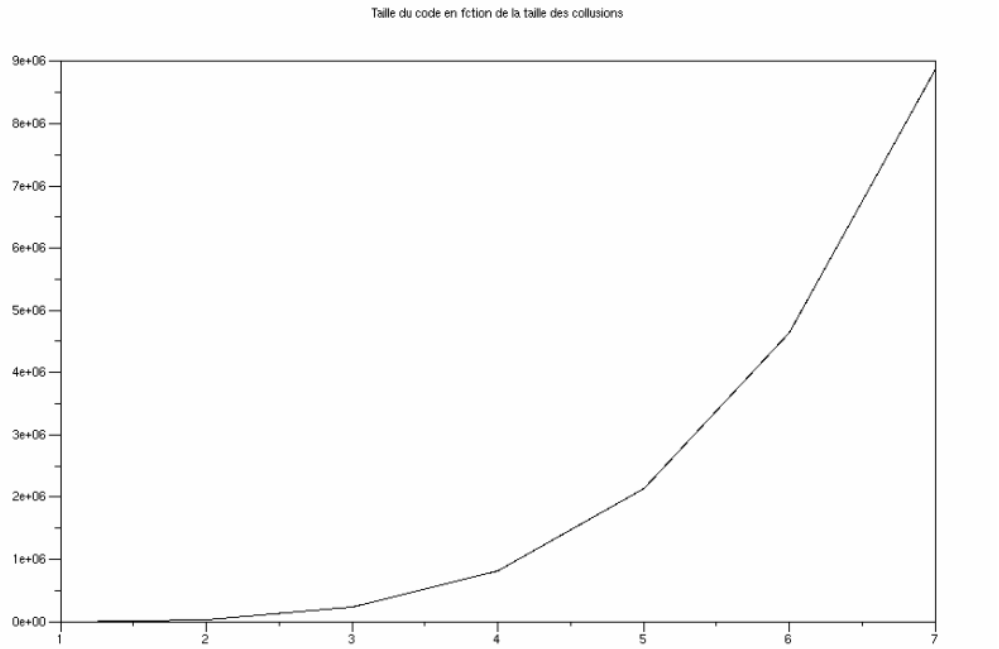


Fig 1 : Taille des mots du code en fonction de la taille des collusions

On remarque, que la taille des mots du code augmente de façon exponentielle par rapport à la taille des collusions. De plus on a aussi la figure suivante :

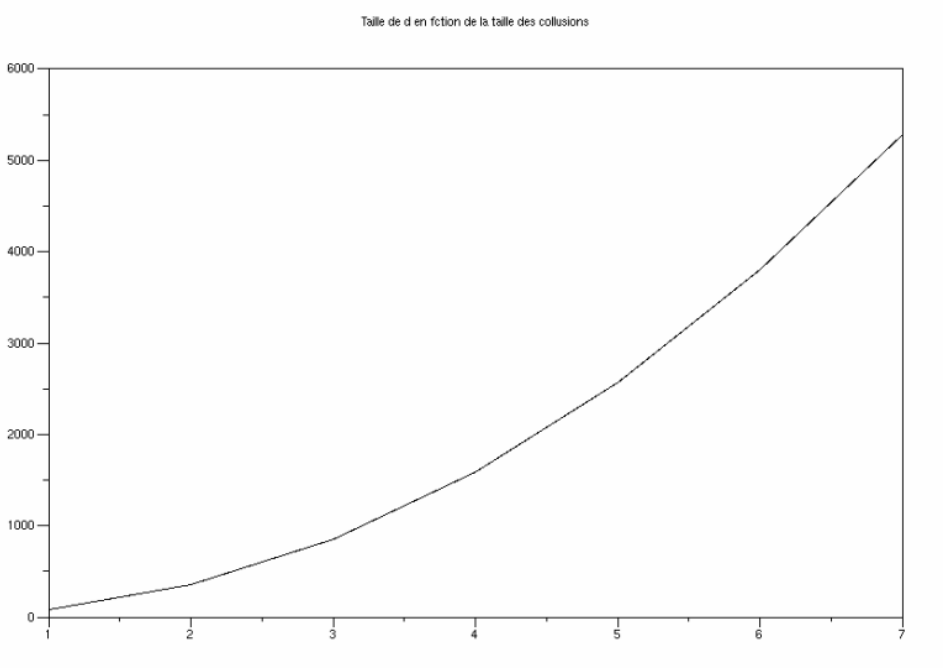


Fig 2 : Taille de d en fonction de la taille des collusions.

On rappelle que $d = 8\ c^2 \log(8cL/\varepsilon)$ (avec les notations vues au dessus Théorème D1 p 39). On remarque à l'aide des deux figures que la taille des mots du code augmente rapidement en fonction de la taille des collusions.

La taille des mots du code augmente rapidement ce qui a provoqué lors de la phase d'insertion dans les images, avec Scilab, des problèmes d'espace mémoires. On a donc dû traiter les phases d'insertion, forge, extraction, par tranche de plusieurs images étant donné la taille des mots du code. Par exemple en prenant des paramètres comme 100 utilisateurs, 3 comme taille de collusion, et 1% d'erreur on obtient des mots de taille d'environ 250 000 bits. Ce qui pose problème lors de du traitement des données, vu que l'on ne peut insérer que 1 bit par bloc de 8 x 8. Il faudrait donc des images de 2 millions par 2 millions ce qui dépasse la capacité de mémoire de Scilab lorsque l'on doit charger plusieurs images pour effectuer des collusions.

6 Tests et résultats

Nous allons décrire les divers tests réalisés et les résultats que nous avons obtenus.
Nous avons effectué deux types de tests. Le premier type sur les mots des codes afin de voir la différence théorique et pratique sur le traçage des pirates. Et le deuxième type de test, sur la collusion des images.

Premier type de test :

On génère le code de Boneh & Shaw en java avec différents paramètres :

- Le nombre d'utilisateur : N
- Taille de la collusion : c
- Erreur tolérée : ε

Ensuite on lance un programme qui génère des collusions entre les utilisateurs, il prend en paramètre :

- Le type de collusion que l'on veut
- Les utilisateurs participant à la collusion

Et pour finir on lance le programme de traçage afin de déterminer les personnes ayant participé à la collusion. Il prend en paramètre :

- mot forgé

On a donc écrit un script Perl qui génère toutes les combinaisons possibles des utilisateurs Afin de le transmettre comme paramètre au programme de collusion. Ensuite , un autre script Perl trie les résultats et affiche les faux positif si il y en a.

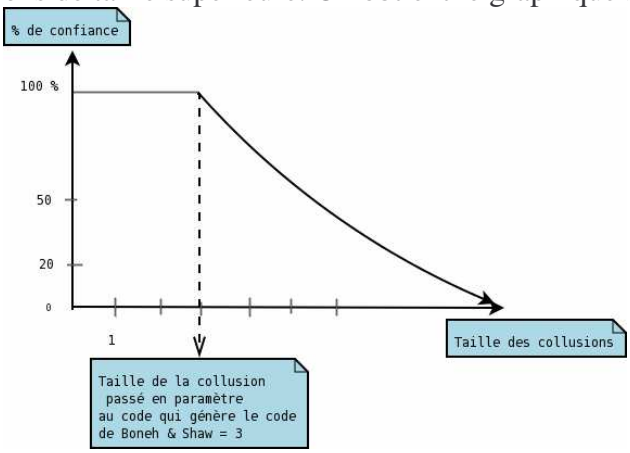
Description d'un test :

Java CréationCode 10 3 0,01 → On génère un code pour 10 utilisateurs avec un taille de collusion de 3 et une erreur de 0,01

Perl Test4.pl → génère toutes les combinaisons de 4 utilisateurs, les passe en paramètre à la fonction Collusion et le résultat est donné en paramètre à la fonction traçage.
Ecrit dans le fichier Res.txt les utilisateurs de la collusion et les utilisateurs trouvés par le traçage

Perl Trier.pl → Lit le fichier Res.txt et affiche les erreurs. C'est-à-dire les faux positifs

Ainsi on a généré un code pour résister à des collusions de taille 3, et on va évaluer sa résistance à des collusions de taille supérieure. On obtient le graphique suivant :



En appliquant cette méthode avec des différents paramètres on remarque que si l'on fabrique un code avec une taille de collusion de c, et que l'on effectue des collusions de taille supérieurs , l'algorithme donne des faux positifs. L'erreur commise devient de plus en plus grande, avec la taille de la collusion. Plus la taille de la collusion augmente et plus l'erreur commise augmente. On voit que si on passe à une collusion de taille c+1 on a une erreur de 20%. Ce qui est trop important pour pouvoir se fier au résultat. On accuserait beaucoup trop d'innocent.

On ne peut donc pas appliquer un code qui tolère une taille de collusion de c, pour pouvoir tracer des pirates appartenant à une collusion de taille c+1. On commettrait une erreur trop importante. On aurait pu croire que l'erreur commise était moins importante, et on aurait pu étendre le code généré pour des collusions de taille c à des collusions de taille c+1.

Deuxième type de test :

On a effectué des tests sur la collusion d'images. Après avoir insérer les mots des utilisateurs dans des images, on applique une fonction de forges aux images. On obtient donc un nouveau mot extrait de l'image forgée, que l'on donne en paramètre à la fonction de traçage. Cependant, ce test n'a pas donné de résultat cohérent. C'est-à-dire que nous avons effectué des tests en utilisant des fonctions de forges dont on pouvait prévoir le résultat mais la fonction de traçage donnait des résultats tout à fait différents. Par manque de temps, nous n'avons pas pu savoir d'où venait l'erreur. Le programme mettant en œuvre ce test, est relativement compliqué et long, de plus il fait intervenir des fonctions de Scilab dont le comportement n'est pas fiable.
Cependant, nous avons utilisé plusieurs fonctions de forges comme la moyenne des images, choisir un bloc de Taille variable, aléatoirement entre plusieurs images. Effectuer la Transformer de Fourrier sur les images ou des bloc et faire la moyenne. Tous ces tests fonctionnent sur des mots de codes de taille relativement petite, mais lorsque l'on passe à des tailles de messages importantes, on voit apparaître des erreurs.

Remarque :

Pour la collusion ou l'on effectue la moyenne des blocs. C'est-à-dire que l'on découpe chaque image en bloc de 8x8 et l'on effectue la moyenne des valeurs des blocs de la collusion.
On s'aperçoit avec l'algorithme de traçage que suivant si l'on a un nombre de pirates pair ou impair on code un 0 ou un 1. Ceci dépendant du seuil de l'algorithme de décodage du watermarking.

7 Améliorations possibles

J'ai expliqué auparavant pourquoi j'ai développé en java, et on peut constater une perte de temps relativement importante sur la gestion des fichiers, c'est-à-dire tout se qui concerne les phases de lecture/écriture de fichier. Il me semble qu'il serait plus avantageux de reprogrammer l'ensemble du code en Scilab. De plus, on utilise la lecture, et l'écriture de matrice relativement souvent, ce qui serait simplifié avec l'utilisation de Scilab. On obtiendrait un gain de temps considérable. De plus, on se passerait de la communication assez fastidieuse entre Java et Scilab.

Nous nous sommes intéressé à quelques fonctions de forges, mais il reste encore plusieurs fonctions de forges à explorer, comme par exemple des fonctions de forges basées sur la décomposition en ondelettes, ou par des transformations sur des groupes de blocs. La liste des fonctions de forges peut être assez longue, et il serait trop long de les étudier une par une.

Nous avons pris comme algorithme de watermarking, celui de Costa Scalaire, on peut facilement développer un autre algorithme afin de le tester avec différentes collusions. On aurait donc une autre étude sur la résistance des attaques par collusion en utilisant toujours le code de Boneh & Shaw mais inséré différemment.

On peut aussi chercher un autre code qui soit collusion-secure afin de l'insérer dans notre chaîne de test afin de comparer les résultats avec cet algorithme et ceux obtenus avec le code de Boneh & Shaw. Comme par exemple l'algorithme de Hirofumi Muratani qui peut présenter un début de recherche sur ce type de code.

Notre étude a utilisée des images comme contenu, on pourrait l'étendre en utilisant des films. Ce passage se ferait assez facilement mais demanderait quand même un travail sur les phases d'insertion/extractions des marques.

8 Annexe

[1] K. Tanaka , Y. Nakamura, et K. Matsui « Embedding secret information into a dithred multi-level image » dans Proc. 1990 IEEE Military Communication Conf. Sept 1990, pp. 216-220.

[2] DigiMarc Co. [Online] disponible WWW: http://www.digimarc.com

[3] N Alon J. Bruck, J. Nao rand R. Roth "Construction of asymptotically good low rate error-correcting codes througth pseudo-random graphs" IEEE Trans. Inform Théory, vol 38 pp 509-516, 1992.

[4] N Alon and J.Spencer, The probabilistic Method New York : Wiley , 1992

[5] Min Wu, Wade Trappe, Z. Jane Wang, and K.J. Ray Liu "Collusion Resistant Multimedia Fingerprinting: AUnified Framework"

[6] H. Joumaa et F. Davoine « Tatouage substitutif d'images intégrant un masque de pondération visuelle »

[7] Robert Bäuml, Roman Tzschoppe, André Kaup and Johannes Huber Information Transmission, Multimedia Communications and Signal Processing, "Optimality of SCS Watermarking"

[8] **Dan Boneh and James Shaw**, IEEE Transaction on information theory, vol. 44, NO. 5, " Collusion-Secure Fingerprinting for Digital Data", September 1998

[9] Hans Georg Schaathun, "The Boneh-Shaw Fingerprinting Scheme is Better than We Thought", 27th November 2003.

[10] Dan Boneh and James Shaw**,** " Collusion-Secure Fingerprinting for Digital Data", October 17, 1996.

[11] Hirofumi Muratani " Optimization and Evaluation of Randomized c-secure CRT Code defined on Polynomial Ring", 2004