IFT 697

Projet Informatique

Cryptographie appliquée à l'image

Ludovic Magerand

22 février 2008

Fig. 1 – Logo de l'Université de Sherbrooke chiffré

Fig. 2 – Logo de l'Université de Sherbrooke déchiffré

Table des matières

Table des figures

Introduction

L'idée de base qui a été à l'origine de ce projet est la constatation suivante : si beaucoup d'algorithmes de cryptographie ont étés développés spécifiquement pour le son ou pour la vidéo, bien peu le sont pour les images fixes. Le but de ce projet était donc de trouver une méthode de cryptographie qui serait spécialisée pour les images, de l'implémenter et ensuite de la comparer aux autres méthodes de cryptographie plus traditionnelles afin de savoir si l'utilisation de telles méthodes peut être bénéfique.

On commencera par définir une méthode de cryptographie basée sur une idée trouvée dans un article au cours de la recherche documentaire. Cette méthode présentera de nombreux avantages par rapport à la méthode originale, mais peut être aussi de légères faiblesses que nous évoquerons. Nous expliquerons donc les problèmes qui était inhérents à la méthode d'origine et la solution adoptée pour y remédier. Pour cela il nous faudra définir une transformée particulière sur laquelle est fondée la méthode implémentée pour réaliser les différents tests. Nous reviendrons d'ailleurs un moment sur quelques autres détails de l'implémentation et notamment sur un certain nombre de choix qui ont été faits et que nous justifierons. Finalement nous montrerons les résultats de quelques tests que nous analyserons.

Nous pourrons ainsi finalement conclure sur le bien fondé et l'intérêt de rechercher de pareilles méthodes.

Remerciements

Je tiens à remercier les différentes personnes qui ont contribuées à ce projet, en commençant tout naturellement par les personnes qui ont acceptés de l'encadrer : Zahir Manadi (étudiant en maîtrise à l'Université de Sherbrooke) et Djemel Ziou (professeur en traitement d'image à l'Université de Sherbrooke). Je remercie aussi Flavien Serge Mani Onana, professeur du cours de sécurité et cryptographie qui en répondant à mes diverses questions sur la cryptographie a, sans même le savoir parfois, contribué à ce projet.

Je remercie aussi les auteurs de l'article de [NTT] sur la transformée théorique de nombres, article qui est la base de travail nécessaire à la méthode développée ici ainsi que les auteurs de [BEK02], pour l'idée fondamentale : utiliser une transformée discrète afin de sécurisé le contenu d'une image.

1 La méthode finale

1.1 Démarche de recherche

1.1.1 Recherche documentaire

Lors de la recherche préliminaire, peu de choses sont vraiment ressorties, principalement des méthodes liées à la vidéo, et notamment à la protection des supports vidéo (comme CSS pour les DVD par exemple). Ces méthodes sont séparables en deux catégories : les premières perturbent les images clefs de la vidéo et sont plutôt rares, les secondes perturbent les vecteurs de mouvements entre les images clefs. Cependant l'une comme l'autre utilisent des spécificités du format vidéo employé (mpeg en général) et ne sont donc pas facilement, voir pas du tout, applicables à des images statiques.

Les mots-clefs utilisés dans la recherche préliminaires n'étaient peut être pas des mieux choisis, en poussant la recherche plus loin, et en essayant de nouveaux mots-clefs, parfois pour le moins étranges, de nouveaux articles sont apparus. Cependant de ces nouveaux articles beaucoup mélangeaient allégrement les techniques de tatouage numérique et de stéganographie avec la cryptographie. Bien que ce soient d'excellent moyen de cacher la présence d'une image et de la camoufler aux yeux des personnes ne possédant pas le logiciel pour l'extraire, l'information reste néanmoins complètement intelligible, il est possible à tout un chacun d'obtenir l'image original. Ce n'est donc pas de la cryptographie au sens pure du terme, où l'information doit être rendu absolument inintelligible sans la clef de déchiffrement.

En utilisant des mots-clefs comme «format d'image», «sécurité», ... D'autres sources d'informations ont encore faites leur apparition, cette fois-ci davantage orientés vers le domaine que l'on visait. Par exemple, bien qu'aucune couche de sécurité n'existe pour le format jpg, il semblerait cependant que les spécifications de la nouvelle norme [JPG2K] prévoient l'ajout d'une couche de sécurité sous la forme d'une extension. Cependant le chapitre de la norme qui concerne la sécurisation de jpeg2000 est encore vierge actuellement. Des autres formats courants d'image, aucun ne semble prévoir une telle extension alors même que certains formats comme par exemple png sont munis d'un mécanisme d'extension puissant.

1.1.2 Inspiration originale

L'idée de base de la méthode implémentée dans ce projet repose sur une idée trouver dans [BEK02] : utiliser une transformée et perturber certaines fréquences bien choisies de celle-ci pour rendre l'image inintelligible. Dans l'article de base, la méthode utilisée est centrée sur le format jpg et donc la transformée utilisée est la transformée en cosinus discrète (la «TCD»). Ce sont dans ce cas bien évidement les basses fréquences qui ont été perturbées car elles sont porteuses des informations les plus grossières, et les perturber rend l'image plus illisible que si l'on perturbait les hautes fréquences contenant les informations liées aux détails et au bruit.

Le but de cette méthode est de rendre des images issues du domaine de l'imagerie médicale, plus précisément des radios, impossibles à interpréter afin de protéger le secret médical des dossiers informatiques des patients. Pour arriver à atteindre ce but, la méthode décrite dans l'article est d'aller, au coeur

des librairies de support de jpg, chiffrer les basses fréquences de l'image après la fin de la compression jpg. Plusieurs niveaux de chiffrement ont été définis, rendant l'image plus ou moins difficile à interpréter, puis ce sont des médecins radiologues qui ont définit le niveau à partir duquel les images n'étant plus utilisable médicalement parlant.

Deux choses cependant me dérangeait avec cette méthode pour une utilisation dans un contexte plus général de cryptographie, avec des images quelconques :

- L'image est certes rendu inutilisable pour une personne extérieur, mais on peut néanmoins encore deviner ce dont il s'agit.
- La méthode est trop facilement attaquable, même si l'image ainsi reconstruite n'est pas utilisable médicalement parlant.

Vous pourrez d'ailleurs constater une illustration de ces deux points en parcourant l'article [BEK02] car les auteurs en était bien conscient.

Il ne s'agit pas ici de dénigrer le travail des auteurs de ce document, l'idée est excellente et leur but atteint : les images ainsi traitées sont absolument inutilisables et impossibles à interpréter même pour d'excellents radiologues, ainsi le secret médical est bien respecté.

1.1.3 Modifications à apporter

La méthode évoquée dans la section précédente n'est pas utilisable en l'état dans un cadre plus général, c'est à dire avec des images moins spécifiques. Néanmoins, comme nous l'avons vu, c'est une très bonne idée, et il devrait donc être possible de la modifier pour en faire une méthode utilisable quelque soit l'image utilisée. C'est donc cette méthode qui allait servir de bases au travail de ce projet, mais il restait à déterminer exactement les problèmes de cette méthode, leur origine et de trouver comment y remédier. C'est ce à quoi nous allons nous attacher dans les pages suivantes.

1.2 La TF (et TCD) versus la TTN

1.2.1 Inconvénients de la TF (et TCD)

Les «faiblesses» de la méthode de base viennent de l'utilisation de la transformée en cosinus discrète, qui est, comme nous le savons, un type particulier de transformée de Fourrier (la «TF»). Cette transformée comporte une propriété qui, bien que très intéressante pour le traitement d'image, devient une faiblesse pour son utilisation en cryptographie : dans le domaine de la transformée de Fourier discrète [1] les valeurs gardent une signification physique bien que l'espace dans lequel elles sont exprimées est subis un changement de repère.

Cela implique deux grands points faibles pour son utilisation dans le domaine du chiffrement :

- La moindre valeur qui ne soit pas chiffré transporte avec elle une certaine quantité d'information directement interprétable non-négligeable qui peut donc être récupéré et utilisé si elle n'est pas chiffré. Pour dissimuler complètement les données d'origine, il faudrait donc chiffrer tout

[1] Aussi appelé domaine fréquentiel car les coordonnées dans ce domaine corresponde à des fréquences physique

le résultat de la transformée, autant chiffrer directement les données d'origine.
– On peut forcer les valeurs non connues (car chiffrées) à des valeurs que l'on sait relativement profitables (typiquement nulles, on ne fait ainsi que retirer une fréquence de l'image). Cela permet une reconstruction partielle de l'image même si l'on aura perdu une certaine quantité d'information. Suivant les fréquences perdues, le résultat sera plus ou moins flou, arrondi, dégradé, ...

Ces deux points à eux seuls suffisent à proscrire l'utilisation de la transformée de Fourier discrète dans une méthode de cryptographie, cependant une autre problématique a aussi poussée à la recherche d'une autre transformée discrète : lors de l'utilisation de la transformée de Fourier discrète ou de la transformée en cosinus discrète, les valeurs renvoyées sont des réels dont les valeurs ne sont pas vraiment facile à encadrer. De plus les nombres réels ne sont pas facile à chiffrer de façon portable. Pour remédier à ce problème, il faut les arrondir en entier puis chiffrer ces entier. Mais même ainsi un autre problème ce pose car les entiers chiffrés peuvent être arbitrairement grand et donc comment stocker le résultat ? Dans le format jpg ce n'est pas tant un problème puisque le format est prévu pour accueillir le résultat d'une transformée dont les valeurs peuvent être quelconque, mais dans un format brute cela peut vite s'avérer problématique.

1.2.2 La transformée théorique de nombres

Pour les raisons expliquées à la section précédente, la recherche d'une nouvelle transformée qui remédiait aux problèmes évoqués était nécessaire. Après une recherche relativement infructueuse dans des moteurs de recherche, c'est finalement l'encyclopédie libre en ligne Wikipedia qui a permis d'apporter une piste de solution grâce à son article [NTT] sur la transformée théoriques des nombres (la «TTN»). Celle-ci est très utilisée en mathématique dans le domaine des calculs avec les grands nombres entiers (voir infini), car elle permet de réaliser des multiplications rapides de ces nombres entre eux.

Cette transformée est basée sur un noyau qui est une racine de l'unité dans les anneaux $\mathbb{Z}/p\mathbb{Z}$ où p est un nombre premier. Les calculs se feront donc toujours modulo p, ce qui est intéressant car le domaine du calcul modulaire est à l'origine de nombreux algorithmes de cryptographie. Elle transforme n nombres entiers en n autres nombres entiers de $\mathbb{Z}/p\mathbb{Z}$, où n est tel que : $p = n\xi + 1,\ \xi \in \mathbb{N}$. En dehors de cela, au niveau mathématique, la formule est la même que celle de la transformée de Fourrier :

$$\forall j \in [0 \dots n-1], f(x_j) = \sum_{k=0}^{n-1} x_k (\omega^{\xi})^{jk} \bmod p$$

L'entité ω^{ξ} est la $\xi^{\text{ième}}$ racine primitive de l'unité dans $\mathbb{Z}/p\mathbb{Z}$. Étant données les contraintes sur n et ξ, on a comme propriété que : $\omega^{p-1} = 1 \bmod p$. On devra rechercher les valeurs de ω à la main car elle n'existe pas d'algorithme pour les calculer automatiquement. A titre de comparaison, dans la transformée de Fourrier les racines primitives de l'unité sont les nombres complexes $e^{-\frac{2\pi i}{n}}$.

Le fait que ω^{ξ} soit une racine primitive de l'unité implique que la plupart des théorèmes valables pour la transformée de Fourrier discrète le sont

aussi pour la transformée théorique de nombres, en particulier : le théorème de la convolution en domaine spatial qui devient une multiplication en domaine fréquentiel (c'est ce théorème qui est utilisé pour réaliser les multiplications rapides de grands nombres entiers) et le théorème qui permet de calculer efficacement les n transformées en une fois en utilisant l'approche «Cooley-Tukey» si n est une puissance de 2.

La transformée inverse est alors donnée par la formule suivante :

$$\forall j \in [0 \dots n-1], f(x_j) = n^{p-2} \sum_{k=0}^{n-1} x_k (\omega^{p-1-\xi})^{kj} \bmod p$$

Cette formule fait intervenir le terme n^{p-2}, l'inverse de n dans $\mathbb{Z}/p\mathbb{Z}$. C'est un terme de normalisation, de même que dans la transformée de Fourrier il faut diviser par la taille de la transformée si l'on souhaite retrouver les valeurs d'origine. Le terme $\omega^{p-1-\xi}$ est quand à lui l'inverse de la $\xi^{\text{ième}}$ racine primitive de l'unité.

1.2.3 La TTN meilleure pour la cryptographie ?

Cette transformée répond à plusieurs attentes en apportant une solution aux inconvénients de la transformée de Fourier discrète et des propriétés qui deviennent des avantages conséquents pour son utilisation en cryptographie :

1. Elle travaille en précision exacte, par conséquent on ne devrait pas, en théorie du moins, avoir de perte de précision (nous y reviendront dans la section suivante).
2. Le domaine d'arrivé est abstrait et n'a pas d'interprétation physique possible. Cela est du au fait que l'on travail dans $\mathbb{Z}/p\mathbb{Z}$ et que donc les calculs se font modulo p.
3. Elle est particulièrement sensible aux changements des valeurs en entrée : il suffit de changer relativement peu une seule valeur et le résultat est complètement différent.

De plus, elle peut se calculer tout aussi rapidement que la transformée de Fourier discrète, ce qui devrait rendre son utilisation aussi efficace que cette dernière, voir meilleure car on ne manipulera que des nombres entiers, modulo p qui plus est donc toujours inférieur à celui-ci.

Il convient en revanche de modérer légèrement le deuxième point, en effet, bien que la plus grande partie des valeurs soient impossibles à interpréter physiquement, il en est une qui, comme dans toutes les transformées, l'est quand même. Il s'agit de la première valeur qui se trouve être la somme (ou moyenne si l'on normalise) des n valeurs d'entrée, en effet l'exposant de la racine primitive est toujours nul lors du calcul de la transformée de la première valeur.

De plus, une fois l'implémentation terminée, quelque chose dénotait dans les résultats obtenus : régulièrement une ligne verticale de pixel (celle correspondant à la valeur du milieu de la transformée) ne semblait pas vraiment «aléatoire». La raison n'est pas exactement déterminée, mais il s'agit très probablement du fait que la quantité $(\omega^{\xi})^{jk}$ passe alors par des valeurs particulière (À démontrer, cela ressemble à une symétrie ou une opposition des valeurs entre elles).

Cependant ces deux points ne sont pas un frein à son utilisation, car il faudra de toute façon perturber certaines de ces valeurs en les chiffrant or étant donné que les autres ne présente pas d'interprétation physique possible, celles-ci seront donc choisies d'office.

1.3 Principe de la méthode de cryptographie

1.3.1 Paramètres de la TTN

Tout d'abord, comme on l'a vu, il faut commencer par déterminer certaines valeurs qui vont être utilisées pour le calcul de la transformée théorique de nombres : p, n et ξ. C'était la un choix délicat car il faut satisfaire plusieurs contraintes en même temps : les conditions pour pouvoir calculer la transformée théorique de nombres mais il faut aussi que n soit une puissance de 2 pour que l'on puisse calculer la transformée théorique de nombres rapidement en utilisant l'approche «Cooley-Tukey». Il faut aussi être capable de déterminer la racine primitive de l'unité pour $\mathbb{Z}/p\mathbb{Z}$.

C'est d'abord le choix de p qui a été fait : l'idéal aurait été de prendre un nombre très proche de 256 puisque les valeurs en entrée vont varié de 0 à 255. On le choisira inférieur car ainsi on pourra réaliser une mise à l'échelle sur les données en entrée, ce qui n'est pas vraiment gênant car on travail sur des images, et une fois la mise à l'échelle annulée lors du déchiffrement, on ne percevra aucune différence à l'oeil entre l'image d'origine et celle déchiffré bien qu'on aura très légèrement perdu en précision au passage (dans les faits : ± 1). Ceci est impensable avec par exemple des données de type texte, car on ne peut se permettre de passer de la lettre «l» à «m» dans ce contexte : l'oeil fera nettement la différence.

Comme il faut que n soit une puissance de 2, inférieur à p de plus, on est finalement arrivé aux valeurs suivantes : $p = 193$, $n = 64$ et donc $\xi = 3$. Il faut maintenant déterminé la racine primitive de l'unité, on peut démontrer facilement que la valeur $\omega = 5$ convient.

1.3.2 Utilisation de la TTN

Maintenant que nous avons les outils et leurs paramètres, nous pouvons décrire l'organisation générale de l'algorithme pour le chiffrement :

- Lire l'image d'entrée
- Pour chaque ligne de l'image d'entrée :
  - Pour chaque bloc de 64 pixels de la ligne :
    - Réaliser la mise à l'échelle du bloc
    - Faire la transformée rapide du bloc
    - Chiffrer les pixels préalablement choisis
  - Écrire la ligne dans l'image de sortie
- Écrire l'image de sortie

Comme on peut le constater, c'est un algorithme simple, qui tiens en quelques lignes de code, en fessant abstraction du code pour faire la transformée (encore que celui-ci ne soit pas bien long non plus). En revanche, exprimé comme ceci, il impose d'avoir une taille horizontale d'image qui soit un multiple de 64 pixels. On trouvera une illustration de ce principe page 7.

On a choisit de faire la transformé sur une seule dimension, ce choix a plusieurs origines. Premièrement, la taille de la transformée étant fixé, si l'on

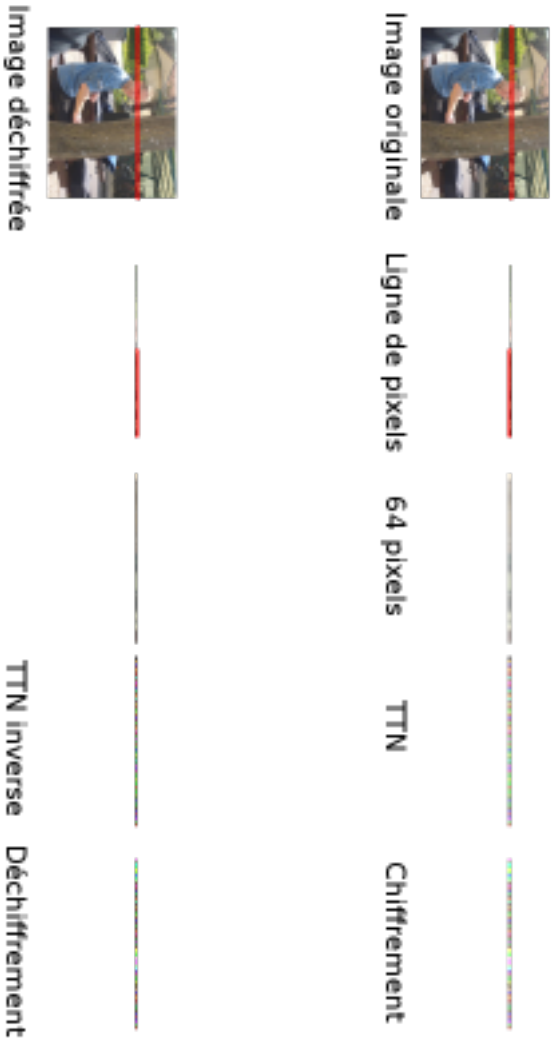

FIG. 3 – Principe de l'algorithme.
De gauche à droite en haut : chiffrement, et de droite à gauche en bas : déchiffrement.

avait voulus faire une transformée en deux dimensions, il aurait fallut imposer que l'image fasse aussi une taille verticale multiple de 64 pixels. Deuxièmement, l'implémentation d'une transformée théorique de nombres en deux dimensions aurait été bien plus complexe, et probablement bien plus lente. Troisièmement, contrairement à la transformée de Fourier discrète ou à la transformée en cosinus discrète, il n'y a en fait aucun intérêt à travailler en deux dimensions. Quatrièmement, le traitement par ligne pourrait avoir de gros avantages pour une implémentation avec certains formats (voir 2.2.2 page 12).

1.3.3 Chiffrement réel

Volontairement, il n'as pas été fait mention de l'algorithme exact utilisé pour le chiffrement des pixels qui doivent l'être, en effet celui-ci peut être n'importe quel algorithme classique de cryptographie. Il y a quand même quelques contraintes qu'il faut satisfaire :

- Il doit pouvoir chiffrer de toute petite quantité de donnée (quelques pixels, donc quelques dizaines de bits)
- Il doit être le plus léger et rapide possible
- Il doit être robuste, sûre, et avec une taille de clef suffisante pour assurer la sécurité

De plus comme il sera capable de chiffrer de petite quantité de donner, on pourra alléger la contrainte de taille sur l'image en la réduisant à un multiple de ce que cet algorithme peut chiffrer.

C'est sur cet algorithme que reposera l'impossibilité d'inverser la transformée car comme on la vue précédemment (voir 3 page 5), la transformée théorique de nombres est très instable, une petite variation sur l'entrée implique une très grosse variation de la sortie par conséquent, en chiffrant réellement qu'une petite partie des valeurs de la transformée, on rend son inversion impossible sans la clef pour déchiffrer ces valeurs. Si l'algorithme de chiffrement est fragile, alors toute la méthode est fragilisée.

1.3.4 Cryptanalyse

La cryptanalyse d'un algorithme de cryptographie peut être une tache longue et ardue (cela fait souvent l'objet de thèses), par conséquent, on n'abordera ici uniquement quelques pistes de possibles failles dans le principe.

Premièrement, lorsque l'on a abordé la stabilité de la transformée théorique de nombres (voir 3 page 5), on a dit qu'il suffisait de changer très peu une seule valeur pour que le résultat soit complètement différent. Si l'on cherche un peu à quantifier ce «très peu», on constatera qu'une variation de seulement un bit est en général suffisante pour perturber complètement la sortie. Néanmoins, lorsque cette variation ce produit sur certains bits de poids faibles, on peut quelques fois retrouver les valeurs d'origine à 20% près.

Deuxièmement, comme le résultat de la transformée théorique de nombres est une suite de valeurs de 0 à 192, celle-ci sont représentées par une suite de 8 bits, hors normalement, on peut représenter 256 valeurs avec 8 bits, par conséquent la probabilité qu'un bit donné soit d'une valeur donnée n'est pas toujours vraiment de 50% comme cela devrait être le cas. Certains bits sont donc plus fragiles que d'autres et pourrait peut être être le point d'entrée pour une faille (notamment les bits de poids forts). Ceci dit, dans le cas des images,

si l'on a une idée de ce que contient l'image, il suffit de se fabriquer quelques histogrammes types et de réaliser une analyse fréquentielle pour arriver au même résultat.

Pour palier ces problème, on a opté pour le chiffrement de plus de bits de la transformée, de sorte que reconstruire les bits soit aussi dur que de brute-forcer la clef de chiffrement car il faut quand même relativisé le fait que ces points faibles concernent la possibilité de reconstruire *une* transformée théorique de nombres, or dans une image moyenne (disons 640x480 pour simplifier le calcul), il y en a déjà 4800 et le temps pour en reconstruire une seule sera déjà bien élevé et ainsi reconstruire l'ensemble de l'image devrait être au moins aussi ardue que de trouver la clef de chiffrement. Donc le temps nécessaire pour reconstruire l'image intégrale doit probablement s'approcher du temps nécessaire pour brute-forcer la clef de chiffrement (128 bits dans la version implémenté).

Un dernier point, connu depuis longtemps, lorsque l'on chiffre un fichier, les zones «uniformes» ont souvent tendance à donner des motifs particulier dans le résultat final. Comme vous pourrez le constatez sur l'exemple figure 4 page 10 qui est la figure 6 de la page 21, ce phénomène est particulièrement flagrant et peut permettre de deviner certaines formes de l'image de façon relativement précise. La solution généralement adopté pour régler ce problème est de ne pas directement chiffré les données en tant que telle avec la clefs, mais d'utiliser le résultat de ce chiffrement comme clef pour une autre opération de chiffrement (en général un simple ou exclusif qui est le système le plus sur pourvu qu'on n'utilise pas plusieurs fois la même clef). Dans notre cas, il existe une autre solution simple et radicalement efficace : lorsque l'on détecte une zone uniforme au chiffrement (assez facile), on peut ajouter un très léger bruit (± 1 tout les deux ou trois pixels) sur cette zone. La transformée fera alors le reste du travail en diffusant ce léger bruit à travers les 64 valeurs, cassant ainsi l'effet montré. On trouvera ainsi page 21 sur la figure 6 la même image que celle que l'exemple, mais chiffré avec notre méthode, en ajoutant un léger bruit sur les zones uniformes.

2 Implémentation

2.1 Système symétrique ou asymétrique ?

2.1.1 Présentation du problème

Dans le domaine de la cryptographie, on distingue deux grandes familles d'algorithmes. La première est la famille des algorithmes dits symétriques, dans cette famille c'est la même clef qui sert au chiffrement et au déchiffrement du message, il faut donc que celle-ci soit connue des deux interlocuteurs. Ce sont des algorithmes de façon générale très rapides, basés sur la difficulté mathématique à inverser certaines fonctions sans un paramètre donné (qui sera la clef). Il existe de nombreux domaines mathématiques pouvant générer de telles fonctions : arithmétique modulaire, courbes elliptiques, combinaisons de calculs ensemblistes, ...

La deuxième familles est celle des algorithmes dits asymétriques, dans cette deuxième catégorie, il y a deux clefs qui interviennent : dans le chiffrement du message pour l'une, et pour le déchiffrement des messages pour l'autre. Les al-

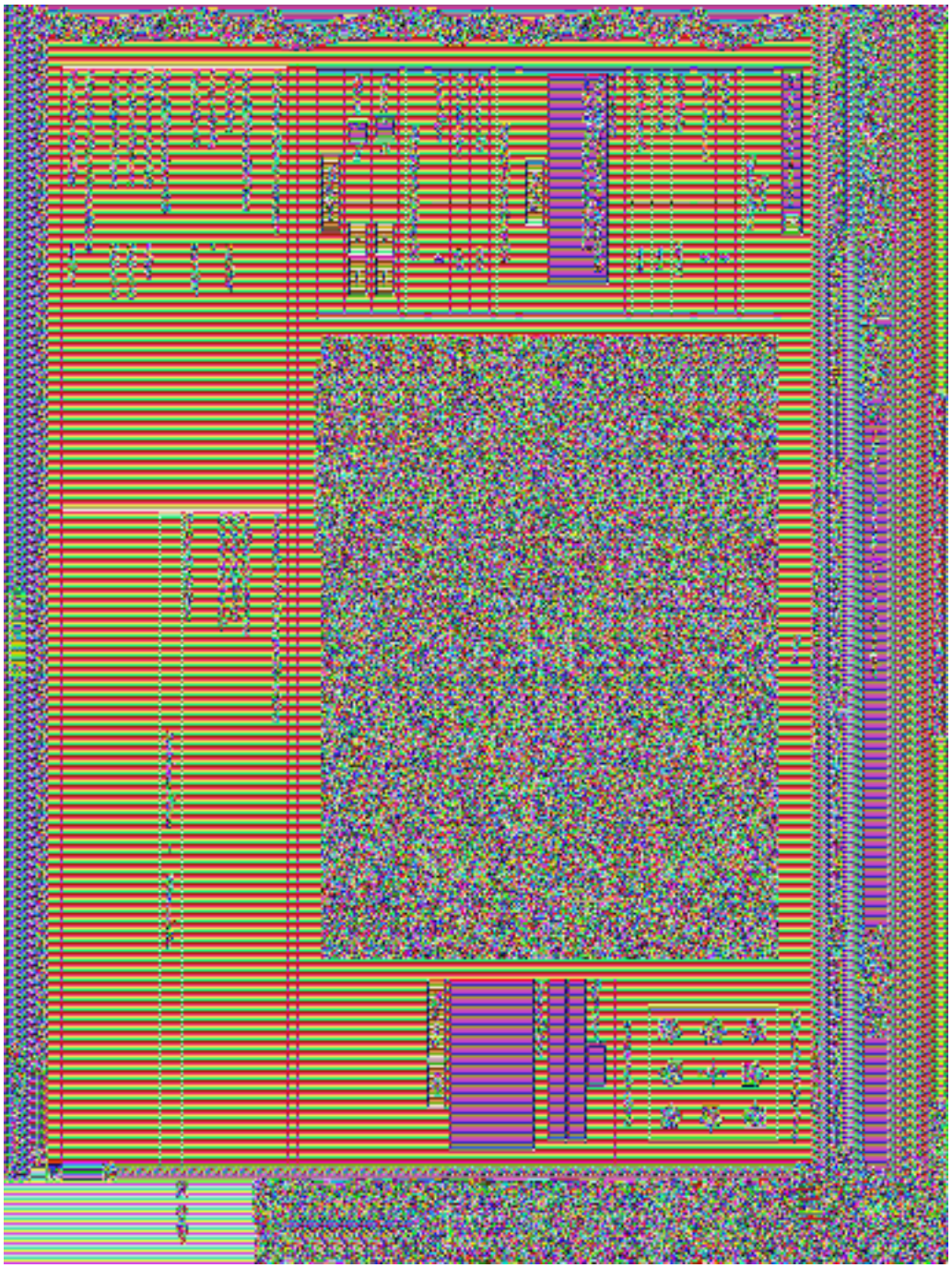

Fig. 4 – Résultat d'un chiffrement sans utilisé de mode

gorithmes de cette famille sont presque tous basés sur la difficulté de factoriser les très grands nombres entiers. Ces algorithmes sont d'autant plus lent que la longueur des clefs augmentent or de nos jours il est de plus en plus facile de factoriser des nombres entiers, les ordinateurs étant de plus en plus puissants, par conséquent la taille des clefs réputées sécuritaires augmentent avec le temps. Ce sont donc des algorithmes relativement très lent et demandant une bonne puissance de calcul.

2.1.2 Choix peu important mais nécessaire

Dans les échanges où le temps de calculs est critique, comme par exemple la vidéo temps réel, il est nécessaire que les opérations de chiffrement et de déchiffrement soient extrêmement rapides. Cependant ce n'est pas ce type d'application qui nous intéresse, par conséquent ce n'est pas le critère principal dans le choix du système. Malgré tout, il ne serait pas agréable que, quand on cherche à afficher une image, le déchiffrement de celle-ci prenne trop de temps. Il faut donc néanmoins rechercher à minimiser le plus possible le temps nécessaire aux opérations de chiffrement et déchiffrement.

Le choix c'est donc porté sur un algorithme à clef symétrique, car de plus il est toujours possible, et c'est ce qui est fait en pratique dans toute les communications bilatéral ou temps réel, d'utiliser un algorithme à clef publique pour chiffrer et échanger la clef de chiffrement symétrique qui sera utilisée par la suite pour l'algorithme qui réalisera la communication effective et qui lui est symétrique.

L'algorithme choisit dans l'implémentation, puisqu'il fallait quand même en choisir un, est l'algorithme «XTEA» (eXtended Tiny Encryption Algorithm, voir [NW97]) qui est un algorithme de chiffrement reconnu comme ne comportant à l'heure actuelle aucune faille et résistant à la cryptanalyse, ce qui a d'ailleurs fait l'objet d'une thèse, voir [VRA03]. Il est de plus complètement libre de droit et permet de chiffré 64 bits de données avec une clef de 128 bits, ce qui en terme de taille de clefs symétriques est tout à fait correct. Pour finir, il est très facile d'implémentation, et tiens en quelques lignes de codes et est par conséquent extrêmement rapide. Tout ces points en fessait le candidat idéal pour l'implémentation que nous souhaitions.

2.2 Format des images

2.2.1 Choix possibles et choix final

Après avoir trouvé la transformée théorique de nombres, l'algorithme commençait à prendre forme, et ces caractéristiques aussi : il fallait notamment pouvoir conserver sans perte la sortie de celui-ci pour pouvoir restaurer l'image originale. Comme il n'était pas question de créer complètement un nouveau format d'image, il fallait donc un format qui soit sans perte pour sauvegarder le résultat.

Plusieurs formats disponibles remplissent cette condition, mais certaines considérations d'ordre pratique ont vite déterminés le choix pour le projet. Il fallait un format qui soit vraiment pratique à utilisé et avec une librairie disponible librement et relativement facile d'utilisation. Deux formats pouvaient répondre à ce nouveau critère :

- Le format «PNG» avec sa librairie libpng, c'est un format qui dispose d'une compression de très bonne qualité et sans perte. Il a de plus un mécanisme d'extension qui est vraiment très intéressant dans le cadre du projet.
- Le format brute probablement le plus connu et utilisé dans le traitement d'image : le «PPM». Il n'est pas forcément nécessaire d'avoir une librairie tant ce format est facile à lire et écrire.

De ces deux formats le choix à été assez dure, et la différence c'est fait principalement sur deux points :

- Premièrement, bien que la libpng soit d'une utilisation facile, l'api étant très bien faite, pour une utilisation optimale du format, il fallait regarder la spécification de près, et celle-ci n'est pas très légère (voir [PNG]).
- Deuxièmement, la possibilité de pouvoir facilement obtenir un résultat en format brute ascii, donc lisible par un humain, en utilisant le format ppm était très attrayant pour résoudre les éventuels problèmes qui aurait pu apparaître (et qui sont apparu parfois) au cours du développement.

Le choix c'est donc finalement porté sur ppm pour son aspect pratique pour les corrections d'erreurs et sa rapidité de mises en oeuvre grâce à la librairie «NetPBM».

2.2.2 Possibilités d'adaptation

Malgré le fait que l'on est choisit le format ppm, il y a un certain nombre de caractéristiques de png qui serait très intéressante pour une intégration future.

Premièrement, il est possible dans png de définir des entêtes optionnels qui permettrait de spécifier que l'image à été chiffré, et pourquoi pas aussi de spécifier par quel algorithme de chiffrement réel. Dans le cas de l'utilisation d'une clef symétrique chiffrée avec un algorithme asymétrique, on pourrait aussi mettre la clef dans cet entête.

Deuxièmement, png prévoit un mécanisme de filtrage des lignes intégré dans le moteur de lecture de l'image. Il serait donc possible de se servir de ce mécanisme pour appeler directement la fonction qui traite les lignes dans l'algorithme, permettant ainsi une énorme économie de temps, ce qui aurait un impact vraiment non-négligeable sur les performances.

Ces deux points permettrait une intégration au coeur de la librairie, et par conséquent permettrait de gagner un temps considérable et donnerais une simplicité d'utilisation remarquable pour l'utilisateur final, tout comme les auteurs de [BEK02] l'ont fait avec leur solution intégré au coeur du moteur jpg.

2.3 Implémenter la TTN

2.3.1 Par la définition de la TTN

Dans un premier temps, on a commencé par une implémentation de la transformée théorique de nombres en suivant la stricte définition pour ne pas induire d'erreur imputable à la version plus complexe qui permet de calculer la transformée plus rapidement. Cette implémentation rapide mais absolument pas efficace nous à permis de tester le comportement de l'algorithme et de valider son principe.

Cette première implémentation à aussi pointée du doigt divers problèmes, parmi lesquels on peut citer :

- La valeur médiane du résultat de la transformée théorique de nombres qui n'était pas vraiment arbitraire, comme on l'a mentionné dans la section 1.2.3 page 5. Néanmoins ce problème a rapidement trouvé une solution via le chiffrement de cette valeur.
- Le fait, prévisible et prévu, que les régions uniformes était décelable, problèmes qui à été facilement résolution en ajoutant un léger bruit car ce phénomène ne se produisait absolument pas sur une des images tests légèrement bruité.
- Une mauvaise gestion de la mise à l'échelle qui avait tendance à foncer l'image et une mauvaise gestion des arrondis qui conduisait à des erreurs sur les pixels blancs.

Une fois tout ces problèmes résolus, l'algorithme marchait remarquablement bien, comme on l'avait espéré. Celui-ci à donc été retenu pour passer à l'étape suivante et ensuite réaliser les tests finals.

Le temps nécessaire pour chiffrer une image était assez conséquent, mais restait néanmoins raisonnable au sens qu'il ne fallait pas non plus attendre plusieurs minutes pour chiffrer ou déchiffrer une image. Il était cependant bien évident que l'utilisation d'une version rapide de la transformée théorique de nombres était absolument nécessaire.

2.3.2 Version rapide

Comme nous l'avons vu à la section 1.2.2 page 4, une des propriétés de la transformée théorique de nombres est que, tout comme la transformée de Fourrier, il est possible d'en faire une version rapide en utilisant l'approche dite de Cooley-Tukey par exemple. C'est l'approche que nous avons adoptée et implémentée, mais dans les faits, d'autres approches de calculs rapides devraient aussi fonctionner.

Cette implémentation à quand même nécessité une somme de travail conséquente, car plusieurs problèmes n'existant pas avec une implémentation de la transformée de Fourrier rapide apparaisse avec la transformée théorique de nombres. Il fallait notamment remplacer le noyau utilisé par le noyau de la transformée théorique de nombres, hors celui-ci n'est pas aussi simple à calculer aux différents étages de la transformée, et tout particulièrement pour la transformée inverse. En effet, dans l'approche Cooley-Tukey, le calcul des valeurs se fait grâce à des étages de «papillons» qui calculent des valeurs temporaires puis réalise sur ces valeurs deux calculs. Dans ces calculs, on utilise l'opposé du noyau, hors si pour l'étage final il est facile de le calculer (il s'agit de $\omega^{p-1-\xi}$), cela n'est pas aussi simple dans les premiers étages. Il a fallut généraliser un peu la formule du calcul du noyau et de ces puissance en fonction de la taille de la transformée n et de la valeur de ξ, ce qui n'est pas nécessairement évident puisque il s'agit de calculs d'arithmétique modulaire.

Il a aussi fallut réaliser une permutation binaire pour conserver l'ordre des valeurs des pixels. En fait cette opération est censé pouvoir être évité puisque si l'on fait la transformée sans, puis qu'on l'inverse sans faire la permutation non plus, normalement au final les valeurs sont censé revenir à leur place d'origine. Malheureusement, ce n'est pas ce qui semblait ce produire, il a donc fallut se rabattre sur une permutation binaire systématique avant

chaque transformée. Néanmoins, une analyse plus poussé du problème devrait parvenir à le résoudre. En attendant, un solution simple mais efficace à été implémenté : l'utilisation de «LookUp Table» pour réaliser rapidement la permutation en même temps que la mise à l'échelle. Cela a été rendu possible car on connaissait n, le calcul de la table était donc facile.

2.3.3 Légères optimisations

Une fois ces problèmes réglés, il fallait optimiser un peu cette transformée car assurément, le résultat n'était pas aussi rapide que escompté et loin d'être efficace.

Pour cela différentes techniques ont été utilisé :

- Remaniement complet des formules de calculs du noyau, plusieurs fois. Cela à pris beaucoup de temps, mais finalement on est arrivé à une solution de permet de pré calculer une partie des valeurs du noyau et de ces puissances
- Utilisation de «LookUp Table» pour conserver les valeurs du noyau et de ces puissances
- Remaniement des formules de calculs des valeurs intermédiaires dans le papillon ainsi que des valeurs finales

Finalement le temps moyen de calcul de la transformée théorique des nombres a été divisé par 15, tout en conservant toutes ces propriétés. Cela nous a permis d'arriver à un temps moyen de calcul pour le chiffrement qui était satisfaisant au vu du critère visé au départ pour le temps de chiffrement : pouvoir chiffrer une image 640x480 assez vite pour pouvoir la transmettre en temps réels sur une vidéo à 12 images secondes, c'est à dire quelques centaines de milli-secondes. C'est un critère tout à fait subjectif mais qui permettait ainsi d'avoir un objectif, et qui surtout permet de rendre compte du fait que l'on ne souhaite pas que l'utilisateur perçoivent l'impact du temps de chiffrement ou déchiffrement, car si c'est correct pour une vidéo, ça le sera bien aussi pour une image statique !

3 Comparaisons

3.1 Méthode de comparaison

Les comparaisons ont été effectuées sous GNU/Linux en utilisant la commande time pour évaluer le temps utilisateur total nécessaire pour chiffrer un ensemble de fichiers tests.

Voici les différents tests qui ont été effectués et chronométrés pour notre méthode :

1. Chiffrement sans ajout de bruit : for i in ‘ls *.ppm‘ ; do ./Release/siea "$i" "AQWzsxEDC 1234567890RfVtGbYhN-.>" e n ; done ;
2. Chiffrement avec ajout de bruit : for i in ‘ls *.ppm‘ ; do ./Release/siea "$i" "AQWzsxEDC 1234567890RfVtGbYhN-.>" e y ; done ;
3. Déchiffrement des resultats : for i in ‘ls out*.ppm‘ ; do ./Release/siea "$i" "AQWzsxEDC 1234567890RfVtGbYhN-.>" d ; done ;

Et voici les tests effectués en utilisant d'autres méthodes de cryptographie traditionnelle :

1. Chiffrement avec l'algorithme rijndael-256 avec le mode CBC : time for i in ‘ls *.ppm‘ ; do mcrypt -a rijndael-256 -k "AQWzsxEDC 1234567890RfVtGbYhN-.>" -m cbc -o asis "$i" ; done ;
2. Chiffrement avec l'algorithme TripleDES avec le mode CBC : time for i in ‘ls *.ppm‘ ; do mcrypt -a tripledes -k "AQWzsxEDC 1234567890RfVtGbYhN-.>" -m cbc -o asis "$i" ; done ;
3. Déchiffrement avec l'algorithme rijndael-256 : time for i in ‘ls *.nc‘ ; do mcrypt -a rijndael-256 -k "AQWzsxEDC 1234567890RfVtGbYhN-.>" -m cbc -o asis -d "$i" ; done ;
4. Déchiffrement avec l'algorithme TripleDES : time for i in ‘ls *.nc‘ ; do mcrypt -a tripledes -k "AQWzsxEDC 1234567890RfVtGbYhN-.>" -m cbc -o asis -d "$i" ; done ;

La commande mcrypt est une commande générique de chiffrement sous unix et qui contient une bonne dizaine d'algorithme de cryptographie. Il était plus simple d'utiliser cette commande plutôt que de chercher à implémenter soit même ces algorithmes. Cependant, comme on le fait remarquer à la section 3.4.1, il est nécessairement bien optimisé.

3.2 Les fichiers de tests

On a utilisé sept fichiers de tests différents que nous allons présenter :

1. Une photographie en extérieur avec un appareil numérique,
   figure 5 page 21 (taille : 1024x768)
2. Une capture d'écran d'un jeu sur le web,
   figure 6 page 21 (taille : 1024x768)
3. Une photographie en intérieur avec une webcam,
   figure 7 page 22 (taille : 720x540)
4. Une image qui est une texture,
   figure 8 page 22 (taille : 512x512)
5. Le drapeau français trouvé sur Wikipedia,
   figure 9 page 22 (taille : 800x533)
6. Le drapeau québécois trouvé sur Wikipedia,
   figure 10 page 23 (taille : 800x533)
7. Le logo de l'Université de Sherbrooke,
   figure 2 sur la couverture (taille : 1024x162)

Ces différents fichier permettent de tester toutes sortes de cas de figure qui peuvent se présenter :

1. Cette photographie comporte un peu toute sorte d'éléments : des textures, des zones uniformes, des détails fins, ...
2. Cette capture d'écran permet de mettre en évidence la nécessité des modes pour les méthodes de cryptographie standard.
3. Cette photographie permet de mettre en évidence le fait le bruit permet de ne pas utiliser de mode.
4. Cette image texturée permet de tester le comportement spécifique face à des textures.

5. Ce drapeau permet de tester le comportement vis à vis de grandes zones uniformes.
6. Celui-la, c'est juste pour ne pas faire de jaloux avec le précédent.
7. Et ça c'est juste pour pouvoir le mettre en première page du rapport :)

Vous trouverez ces fichiers dans l'annexe A page 21, en dehors du logo que l'on trouve sur la première page.

3.3 Résultats

3.3.1 Performances

Voici les temps utilisateurs relevés par la commande time pour les différents tests de notre méthode :

Test	T. 1	T. 2	T. 3	T. 4	Moy.
Chiffrement sans bruit	3.603s	3.597s	3.606s	3.605s	3,603s
Chiffrement avec bruit	3.882s	3.896s	3.885s	3.883s	3,886s
Déchiffrement	3.857s	3.853s	3.867s	3.852s	3,857s

On constate que étrangement le temps nécessaire au déchiffrement est plus élevé que le temps nécessaire au Chiffrement sans bruit, alors qu'on se serait attendu à des temps comparables. Bien évidement le temps nécessaire pour le chiffrement en ajoutant du bruit (et donc en fessant appelle à une générateur aléatoire) est plus élevé. Cependant on a quand même réussis à bien limité cet impact en ne le réalisant que sur les zones uniformes [2].

Voici maintenant les temps utilisateurs relevés par la commande time pour les différents tests des autres algorithmes :

Test	T. 1	T. 2	T. 3	T. 4	Moy.
Chiffrement Rijndael	1.192s	1.176s	1.177s	1.179s	1.181s
Déchiffrement Rijndael	1.172s	1.173s	1.179s	1.183s	1.177s
Chiffrement 3DES	3.672s	3.665s	3.633s	3.661s	3.658s
Déchiffrement 3DES	3.630s	3.644s	3.633s	3.631s	3,635s

Évidement le triple DES, qui est un algorithme relativement vieux et lent, est bien plus lent que des algorithmes plus modernes comme par exemple Rijndael. Cependant c'est un algorithme qui est encore utilisé assez souvent de nos jours, aussi je pense que la comparaisons avec celui-ci n'est pas si insignifiante qu'il pourrait y paraître.

3.3.2 Efficacité

Vous pourrez directement constater par vous même l'efficacité de l'algorithme en consultant l'annexe et notamment les figures 11 à 16 page 23 à 25.

Comme vous pouvez le voir, le fichier semblerait presque remplis de données complètement aléatoire [3]. C'est le signe partiel d'un bon résultat en cryptographie car un bon algorithme de cryptographie doit satisfaire la condition

[2] Et pourtant la détection est des plus simples qui soit puisque on se contente de regarder si la valeur du pixel précédent est la même que celle du pixel courant, et cela uniquement tout les 3 pixels. Cet ajout de bruit est d'ailleurs insuffisant si on l'utilise sur un chiffrement classique, preuve de la haute diffusivité de la TTN.

[3] Si on regarde de plus près, on peut distinguer de grandes lignes verticales plus claires à intervalles réguliers. Ce n'est pas mauvais en soit, puisque il s'agit la uniquement de l'effet du chiffrement réel des pixels aux fréquences choisies, chiffrement qui donne un résultat entre

suivante : «on ne doit pouvoir distinguer les données générées par l'algorithme de données générées par un générateur aléatoire» (il y a bien évidemment d'autres condition).

Il faut noter que ces images-ci ont été obtenue en utilisant la perturbation des zones uniformes, sans quoi le résultat de certaines aurait probablement plutôt ressemblé à l'image 4 page 10 vu précédemment.

3.4 Analyse

3.4.1 Retour sur les performances

On peut voir que notre algorithme semble un peu moins performant que certains des plus récents algorithmes de cryptographie. Cependant il n'a pas non plus à rougir tant que cela puisque il égale presque les moins performants d'entre eux.

Cela est en plus tout à fait normal puisque l'implémentation que nous avons faite est très loin d'être optimale, pour faciliter les corrections d'erreur et les retoucher pendant le développement le code est très compartimenté et de nombreuses portions de code pourrait être fusionner pour réduire grandement la complexité global et amélioré les performances. Il reste aussi des améliorations à apporter et des optimisations à faire sur la transformée théorique de nombres.

Qui plus est, l'intégration au coeur d'une librairie d'image améliorera très probablement encore la situation, et il faut aussi préciser que le logiciel utilisé pour chiffrer avec les algorithmes habituels est particulièrement vieux et très bien optimisé depuis fort longtemps.

3.4.2 Retour sur l'efficacité

Plusieurs tentatives de reconstructions ont été tenté sur différentes images, en essayant différentes techniques pour substituer les valeurs chiffrées :

- Laissé telles quelles (testé sur toutes)
- Mise à zéro (testé sur toutes)
- Forçage à 128 (testé sur toutes)
- Forçage à 255 (testé sur toutes)
- Mise à la moyenne des valeurs non-chiffrées (testé une seule fois)
- Mise à la médiane des valeurs non-chiffrées (testé une seule fois)
- Forçage à des valeurs aléatoires (testé une seule fois)

Les manipulations réalisées une seule fois l'ont été sur deux petites images (taille : 128x20) extraites des fichiers tests puis en effectuant la manipulation à la main sur une sortie effectuée en utilisant le format ascii de ppm.

De toute ces tentatives, aucune n'a permis de reconstruire même partiellement l'image, alors qu'on aurait pu s'attendre à ce que certaines images, comme les images composées de grandes régions uniformes, donnent des résultats. Systématiquement le résultat obtenu était du même type que les images sortis de l'algorithme, les bandes claires verticales en moins. Les fichiers résultats n'ont pas été joint au rapport pour ne pas le rendre plus lourd qu'il ne

0 et 255 alors que le résultat de la transformée théorique de nombre est entre 0 et 192, ce qui parait plus foncé.

l'ai déjà [4].

Cela ne constitue pas techniquement une preuve que l'algorithme est sécuritaire, mais néanmoins, il semble relativement résistant à des techniques simples qui auraient donné des résultats sur la méthode qui nous a servit de base. Il faudrait faire une cryptanalyse plus poussée de l'algorithme et notamment voir son comportement face à des méthodes d'analyse fréquentielles pour pouvoir conclure de façon plus net sur le niveau d'efficacité de l'algorithme.

[4] Et en fait surtout parce que je n'ai plus ces images car j'ai du faire de la place sur mon disque dur.

Conclusion

Comme nous l'avons vu tant dans la recherche documentaire que par la méthode que nous avons implémentée, il est donc possible de trouver des algorithmes spécifiques au domaine de l'image, qui, de plus, pourraient dépasser en performances les méthodes de cryptographies traditionnelles. Ces algorithmes, en prenant en compte les différentes spécificités techniques des images, peuvent simplifier largement les algorithmes de chiffrements et de déchiffrements. De plus l'intégration de ces algorithmes au coeur des librairies de traitement des formats de fichiers d'image peut s'avérer très bénéfique car les images sont déjà chargées en mémoire, prêtes à être chiffrées ou déchiffrées. Le traitement à la volée est alors très rapide en associant le processus de chiffrement (ou déchiffrement) avec celui de compression (ou décompression).

On peut d'ailleurs se demander quelles seraient les performances de la méthode implémentée ici si elle l'était de façon optimale au sein d'une telle librairie. Il faudrait aussi réaliser une étude plus poussé sur l'algorithme de façon à mettre à jour toutes ces éventuelles failles ainsi que toutes les possibilités d'amélioration qui pourrait encore y être apportées tant au niveau de la performance que de la sécurité.

On peut aussi s'interroger sur la possibilité d'utiliser la même méthode avec des images en haute résolution de couleurs (sur plus de 24 bits donc). En effet il serait alors possible d'augmenter significativement la taille de traitement de la transformée et chiffrer des lignes entières en une seule transformée. De plus cela diminuerait les pertes d'information dues à la mise à l'échelle. C'est donc une adaptation à étudier, le problème étant alors de trouver les bonnes valeurs pour les différents paramètres de la transformée théorique de nombres.

Références

[BEK02] **Nouvelle approche basée sur la TCD pour la crypto-compression des images médicales.**
Mohamed Salim BOUHLEL - Mourad ELLOUMI - Lotfi KAMOUN.
2002, École Nationale d'Ingénieurs de Sfax.

[PR] **Sécurisation d'image par crypto-tatouage.**
William PUECH - José Marconi RODRIGUES.
LIRMM, Université de Montpellier II.

[R06] **Transfer sécurisé d'image par combinaison de techniques de compression, cryptage et marquage.**
José Marconi RODRIGUES.
2006, Université de Montpellier II.

[B04] **Élaboration d'une technique d'accès conditionnel par tatouage et embrouillage vidéo basée sur la perturbation des vecteurs de mouvement.**
Yann BODO.
2004, ENST.

[PP04] **Le tatouage d'image ou «watermarking».**
Julien PUGLIESI - Cedric PIOVANO.
2004, Université de Nice - Sophia Antipolis.

[LKC97] **Image compression and encryption using tree structures.**
F. Li - J. Knipe - H. Cheng.
1997, Pattern Recognition Letters.

[CHC01] **A new encryption algorithm for image cryptosystem.**
C.C. Chang - M.S. Hwang - T-S Chen.
2001, The Journal of Systems and Software.

[SS03] **A technique for image encryption using digital signature.**
A. Sinha - K. Singh.
2003, Optics Communications.

[PCD01] **Transfert sécurisé d'images par chiffrement de Vigenère.**
W. Puech, J.J. Charre, et M. Dumas.
2001, NîmesTic, La relation Homme - Système : Complexe.

[NW94] **TEA, A Tiny Encryption Algorithm.**
Roger NEEDHAM - David WHEELER.
1994, Cambridge University.

[NW97] **Tea extensions.**
Roger NEEDHAM - David WHEELER.
1997, Cambridge University.

[VRA03] **A cryptanalysis of the Tiny Encryption Algorithm.**
Vikram Reddy ANDEM.
2003, The University of Alabama.

[NTT] **Number-theoretic transform (lien)**.

[PNG] **PNG (lien)**.

[JPG2K] **JPEG200 (lien)**.

A Fichiers de tests

FIG. 5 – Photographie prise en extérieur avec un appareil photo numérique

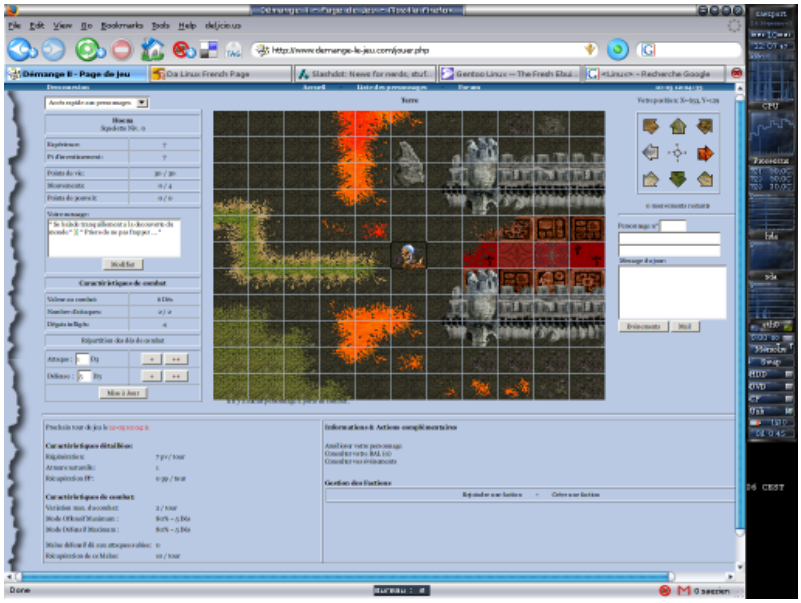

FIG. 6 – Capture d'écran d'un jeu web

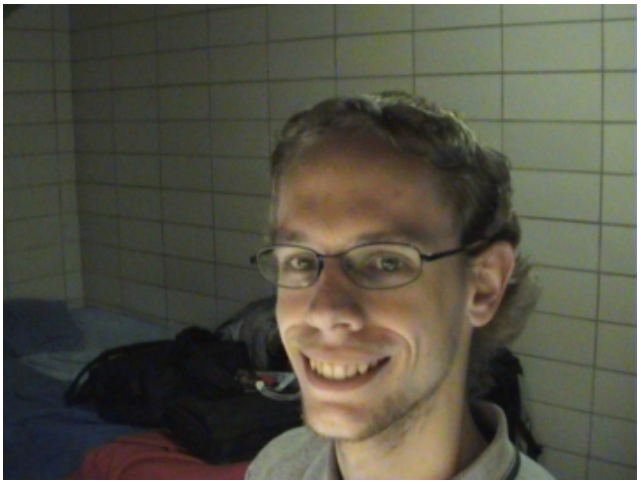

FIG. 7 – Photographie prise en intérieur avec une webcam (bruitée)

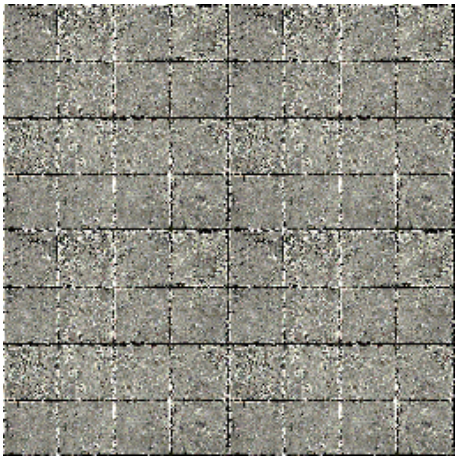

FIG. 8 – Image d'une texture de sol

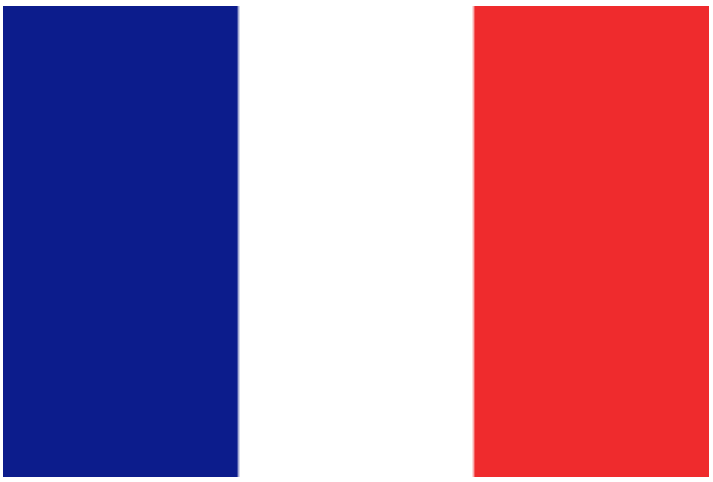

FIG. 9 – Drapeau français

Fig. 10 – Drapeau québécois

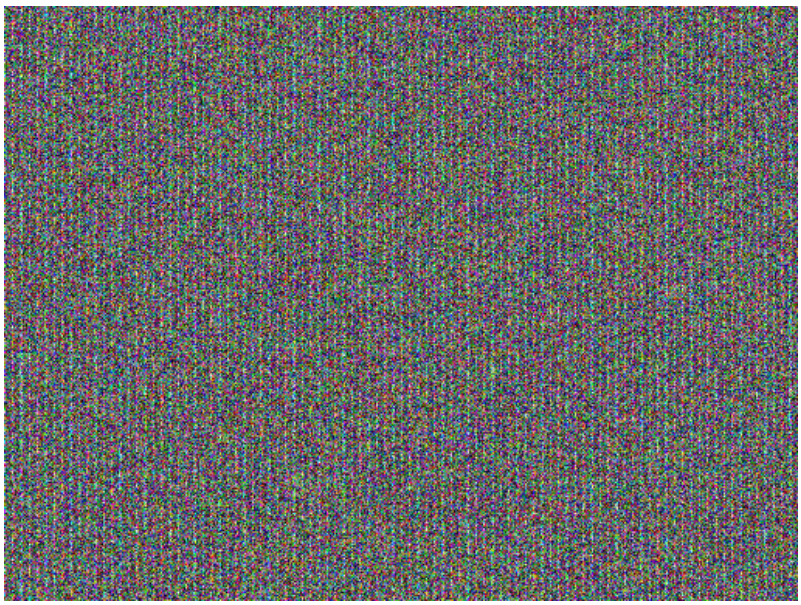

Fig. 11 – Figure 5 chiffrée

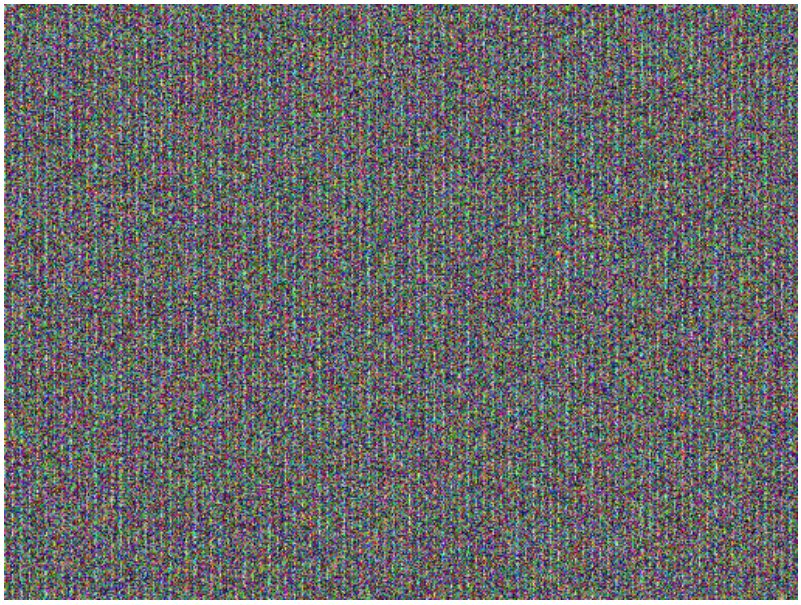

FIG. 12 – Figure 6 chiffrée

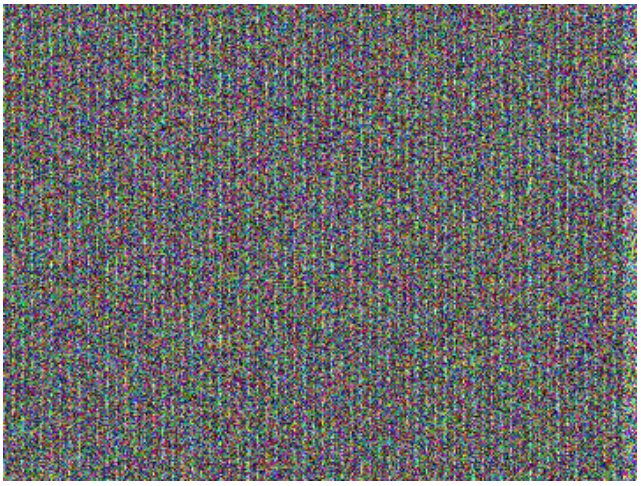

FIG. 13 – Figure 7 chiffrée

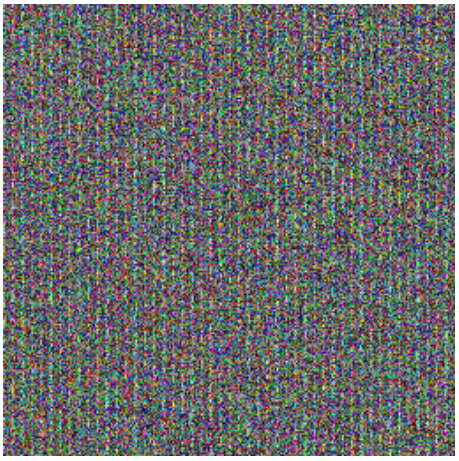

FIG. 14 – Figure 8 chiffrée

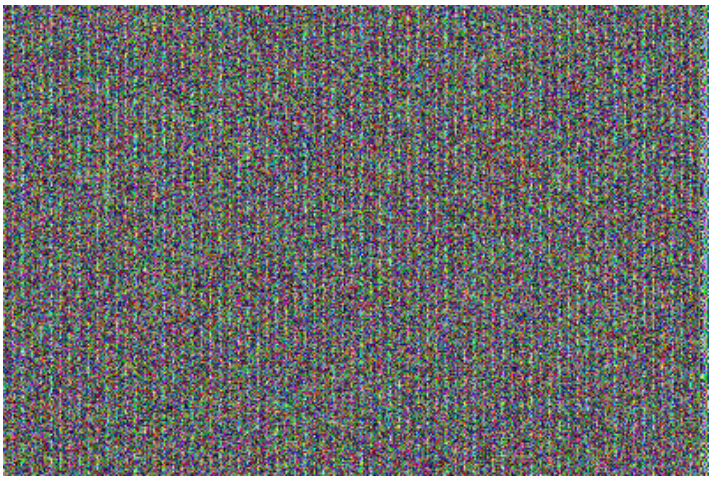

FIG. 15 – Figure 9 chiffrée

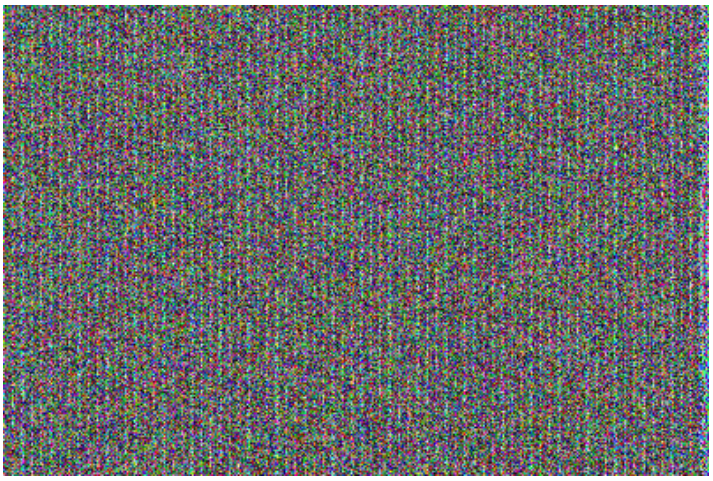

FIG. 16 – Figure 10 chiffrée

B Compilation et utilisation du programme

B.1 Compilation

L'archive fournie contient le dossier du projet réalisé sous l'environnement libre «Eclipse» en utilisant le «CDT» (C/C++ Development Tooling, aussi libre) et sous le système d'exploitation «GNU/Linux». La procédure d'importation est alors simple : créez un nouveau projet (du type «Managed Make C Project»), choisissez lui un nom et validez les choix par défaut. Ensuite sélectionnez ce projet, puis faites «Import», sélectionnez «Archive File» dans la catégorie «General» et enfin donnez lui le chemin de l'archive en cochant en bas la case «Overwrite existing ressource whitout warning». Fermez le projet (pas Eclipse), et ouvrez le de nouveau. Ainsi vous devriez retrouver exactement la configuration du projet qui permet sa compilation.

Pour les autres environnements, il faudra créer un nouveau projet, importer le code source et configurer l'environnement. Pour information, les dépendances, entre fichiers et avec l'extérieur, sont les suivantes :

- main.c : main.h, siea.h et siea.c, ainsi que la librairie NetPBM
- siea.c : ntt.h et ntt.c ainsi que xtea.h et xtea.c
- ntt.c : mod.h et mod.c

Tout les fichiers dépendent aussi de stdinc.h. De plus, le code source respecte le standard C99 (et doit donc être compilé en tant que tel) et nécessite la librairie standard de cette norme pour pouvoir être compiler (notamment la librairie mathématique et les librairies de gestion des nombres entiers portables). Enfin, il faudra donc aussi installer la librairie NetPBM avant de pouvoir compiler le source. Elle est sous licence libre et ce trouve sur le site de NetPBM. Il faudra aussi configurer l'environnement pour qu'il est accès à la librairie.

B.2 Utilisation

L'utilisation est très simple : ouvrer un terminal, rendez dans le répertoire ou ce trouve les fichiers images au format ppm que vous voulez chiffrer ou déchiffrer. Appeler le programme avec en paramètres :

- Le nom du fichier à traiter
- La clef à utiliser (entourez la de préférence de ")
- Une lettre correspondant à la transformation que l'on veut effectuer : «e» pour chiffrer et «d» pour déchiffrer
- Dans le cas du chiffrement, précisez également si l'on doit ajouter du bruit ou non («y» ou «n»)

Vous pouvez trouvez des exemples de lignes de commandes complètes à la section 3.1 page 14, ainsi même que la ligne de commande sous unix pour chiffrer plusieurs fichiers en une fois avec la même clef.