Etudes en sécurité informatique

Patrick Lacharme

Normandie Université

Habilitation à Diriger des Recherches

Pour obtenir le diplôme de d'habilitation à diriger des recherches

Spécialité INFORMATIQUE

Préparée au sein de l'ENSICAEN et de l'UNICAEN

Études en sécurité informatique

Présentée et soutenue par
Patrick LACHARME

HDR soutenue publiquement le 6 février 2020 devant le jury composé de		
Marc-Olivier KILLIJIAN	DR CNRS, LAAS, Toulouse	Rapporteur
Maria NAYA-PLASENCIA	DR INRIA, Paris	Rapporteur
William PUECH	Professeur LIRMM Université de Montpellier	Rapporteur
Philippe GABORIT	Professeur XLIM Université de Limoges	Examinateur
Damien VERGNAUD	Professeur LIP 6 Sorbonne université	Examinateur
Christophe ROSENBERGER	Professeur GREYC, Ensicaen	Examinateur (garant)

UNICAEN UNIVERSITÉ CAEN NORMANDIE

ENSICAEN ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

GREYC

Table des matières

Introduction **1**

1 Codes et génération aléatoire **3**
1.1 Présentation des correcteurs 4
1.2 Correcteurs non linéaires 8
1.3 Codes de Kerdock et engagements flous 13

2 Données personnelles et anonymat **19**
2.1 Protection des données de e-santé 20
2.2 Pass de transport anonyme et intraçable 23

3 Protection des données biométriques **29**
3.1 Transformations de données biométriques 30
3.2 Transformation biohashing 32
3.3 Transformations GRP-IoM et URP-IoM 36

4 Analyse forensique de dumps **43**
4.1 Détection des données aléatoires 44
4.2 Analyse textuelle des dumps 46
4.3 Recherche de dates 49

5 Authentification des personnes **53**
5.1 Attaque sur un CAPTCHA 54
5.2 Robustesse des mots de passe 57

Conclusion **63**

Bibliographie **65**

Introduction

MIT, début des années 60. Robert Morris raconte : "*A system administrator on the CTSS system was editing the password file and another system administrator was editing the daily message that is printed on everyone's terminal on login. Due to a software design error, the temporary editor files of the two users were interchanged*" [102]. Oups les mots de passe ! À l'époque, le fichier contenant les mots de passe des utilisateurs n'était protégé que par un contrôle d'accès. Les mots de passe sous UNIX ont ensuite été stockés sous forme hachés avec la fonction `crypt(3)`, qui correspond au chiffrement de la chaîne nulle par une série de DES avec une clé dérivée du mot de passe. L'utilisation d'une fonction de hachage ne protégeait toutefois pas les mots de passe faibles. C'est cette vulnérabilité qui a été exploitée en 1988 par Robert Morris (fils) pour le développement du premier ver informatique qui a infecté une partie du réseau Internet de l'époque [132].

Comme l'illustre cette anecdote historique, la sécurité informatique est une discipline qui touche à tous les domaines de l'informatique et il est difficile d'en maîtriser la globalité. La liste des failles de sécurité dans l'histoire de l'informatique et de l'Internet est très longue et tout aussi variée. Les causes proviennent aussi bien d'une méconnaissance des protocoles cryptographiques, de problèmes d'implémentation ou d'une trop grande confiance de certains acteurs industriels dans la sécurité par l'obscurité, sans parler du facteur humain.

Par exemple, le générateur de données aléatoires du navigateur Netscape utilisait à ses débuts une graine dérivée d'une donnée temporelle et du PID du processus [58]. D'un autre coté, les fuites de bases de données importantes de mots de passe [73], comme celles de données biométriques [67], sont aussi largement répandues. Ce dernier type de fuite dépasse d'ailleurs largement le domaine de la sécurité pour englober aussi celui de la protection de la vie privée, la biométrie étant directement visée par plusieurs articles de la loi Informatique et Liberté de la CNIL [33].

Ainsi, analyser la sécurité d'un système informatique demande une vision globale du système et une maîtrise d'outils très différents. D'autant plus qu'un système est le plus souvent sécurisé jusqu'à ce que quelqu'un trouve une attaque sur celui-ci, ce qui entraine souvent son lot de contre-mesures, puis de nouvelles attaques. De telles attaques ont parfois un effet très positif car elles permettent l'émergence de nouveaux objets adaptés à une problématique donnée.

Mes travaux de recherche couvrent aussi bien des aspects théoriques que des applications de la sécurité informatique. La cryptographie y est largement présente, mais une bonne partie de ces travaux ne fait que s'approcher de cette thématique. Ce manuscrit présente différentes études indépendantes que j'ai mené en tant qu'enseignant chercheur, essentiellement à l'Ensicaen (au sein du laboratoire GREYC) depuis que j'y ai été recruté maître de conférences en informatique en septembre 2010. La plupart de ce qui est présenté a fait l'objet d'une étude théorique et a été mis en oeuvre. On y trouve plus précisément les six domaines suivants :

— la génération de données aléatoires qui est un point critique en sécurité, en particulier pour la génération de clés cryptographiques dans le monde de l'embarqué. Un tel générateur est nécessaire dans tout système utilisant de la cryptographie et une mauvaise conception peut engendrer une faille de sécurité importante. Cette partie est notamment liée aux notions de codes correcteurs d'erreurs et de fonctions booléennes et est présentée au chapitre 1.

— la protection de la vie privée, notamment à traves les notions de minimisation des données personnelles, d'anonymat et de non traçabilité. Des protocoles cryptographiques dédiés à ces questions de protection de la vie privée, ainsi qu'un système de partage de clés pour du contrôle d'accès sont utilisés. Cette partie a été implémentée sous des contraintes industrielles pour permettre son déploiement et est présentée au chapitre 2.

— la protection des données biométriques, à travers les engagements flous et la présentation de plusieurs attaques sur des transformations biométriques. L'élaboration d'un système d'engagement flou basé sur un code non linéaire termine le chapitre 1, tandis que des attaques utilisant des techniques allant de l'utilisation d'algorithmes génétiques à de la programmation linéaire sont présentées au chapitre 3 sur des données provenant d'empreintes digitales.

— la forensique sur des dumps mémoire de carte à puce. L'analyse automatique des mémoires a permis notamment de séparer les données non aléatoires des données aléatoires, de retrouver les informations textuelles et les dates sur plusieurs centaines de dumps. Cette partie utilise en particulier des techniques d'apprentissage automatique telles que le boosting ou l'utilisation d'un classifieur bayésien. Elle est présentée au chapitre 4.

— l'attaque d'un CAPTCHA récent. Ces tests de Turing doivent proposer un test facile à résoudre pour un être humain et difficile pour une machine. L'attaque présentée au début du chapitre 5 donne de meilleurs résultats pour une machine que pour un être humain. Ce travail englobe une analyse de code, de l'algorithmique et une implémentation qui réalise l'attaque sur le serveur en injectant directement du code javascript dans le navigateur du client.

— la sécurité des mots de passe et leur résistance aux énumérateurs académiques et professionnels. Les métriques généralement utilisées pour évaluer la performance de ces énumérateurs ne correspondent pas à la réalité car le temps d'énumération n'est pas pris en compte. Cette partie propose une méthodologie pour répondre à ce problème et est testée sur plusieurs bases de mots de passe réels. Elle est présentée en fin de chapitre 5.

Chapitre 1

Codes et génération aléatoire

Sommaire

1.1 Présentation des correcteurs 4
1.2 Correcteurs non linéaires 8
1.3 Codes de Kerdock et engagements flous 13

La cryptographie fait un usage important de données aléatoires, par exemple pour la génération de clés, de challenges dans des protocoles d'authentification ou pour la protection des implémentations contres des attaques par canaux cachés. Un générateur aléatoire comporte une partie non déterministe appelée ici source de bruit. Cette source produit des données binaires qui peuvent être biaisées ou corrélées. On analyse ici la réduction du biais d'une source de bruit à l'aide de fonctions booléennes appelées correcteurs. Ce travail a été réalisé en partie pendant ma thèse, soutenue en 2007 à l'université de Toulon, avec mon directeur de thèse Philippe Langevin [89, 90, 91], puis poursuivi dans les deux années qui ont suivi [92], d'abord à l'université de Toulon (Institut mathématiques de Toulon) puis à l'université de Limoges (laboratoire XLIM). D'autres travaux dans le domaine de la génération de données aléatoires non décrits dans ce document ont aussi été réalisés au même moment : une analyse détaillée des générateurs aléatoires de Linux `/dev/random` et `/dev/urandom/` a notamment été présentée dans [95].

Ce chapitre se termine par une section qui étudie la construction d'engagements flous, qui a été réalisée vers 2015 et n'a jamais été publiée. Ces engagements ont été proposés en 1999 par A. Juels et M. Wattenberg pour la protection des données biométriques [81]. Ils utilisent des codes correcteurs et toutes les constructions sont basées sur des codes linéaires. Deux attaques ont néanmoins été publiées en 2009 par K. Simoens, P. Tuyls et B. Preneel dans [130], basées notamment sur la linéarité des codes. Je me suis donc intéressé aux codes de Kerdock car cette classe de codes est non linéaire et peut être implémentée efficacement.

1.1 Présentation des correcteurs

Les données aléatoires doivent non seulement posséder de bonnes propriétés statistiques mais doivent aussi être imprévisibles, ce qui distingue la cryptographie à d'autres applications comme les algorithmes de Monte-Carlo. Des failles dans la génération de données aléatoires, comme celle présentée en 2012 dans [70] pour les clés RSA ont des conséquences immédiates pour la sécurité.

Un schéma général d'un générateur de données aléatoires tel que décrit dans l'AIS 31 [87] est présenté figure 1.1 où les correcteurs définis dans cette partie se trouvent entre la source d'entropie et le traitement cryptographique déterministe. Notons que ce modèle n'est pas celui du générateur de Linux `/dev/random` qui possède un réservoir d'entropie comme ceux de la classe PTG4 de l'AIS 31. Historiquement, le premier correcteur est le débiaiseur de J. Von Neumann [106]. Pour une référence plus globale sur l'utilisation des générateurs de nombres aléatoires en cryptographie, on se reportera à la thèse de 2015 de S. Ruhault [1, 124].

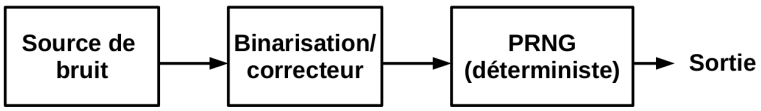


FIGURE 1.1 – Modèle d'un générateur aléatoire (AIS 31, classe PTG3 [87])

Dans cette section, on suppose que les données binaires sont indépendantes, mais biaisées. Plus précisément, la probabilité qu'un bit soit 0 est notée p et celle d'être à 1 est $1-p$ (en considérant $0 < p < 0.5$, sans perte de généralité). L'objectif est de déterminer quelle fonction peut être appliquée à cette source pour réduire ce biais en sortie. En prenant par exemple le ou exclusif (xor) entre chaque paire de bit, la probabilité d'avoir 0 en sortie est $p^2 + (1-p)^2$ et celle d'avoir 1 est $2p(1-p)$. En notant $p = 0.5 + e$, où e est appelé le biais, on obtient un biais en sortie de $2e^2$. Si on généralise cette construction au xor de n bits consécutifs, on obtient un biais en sortie de $2^{n-1}e^n$, mais le rendement (rapport entre le nombre de bits en sortie et celui en entrée) est plus faible.

M. Dichtl a identifié en 2007 certaines fonctions qui permettaient d'avoir de meilleurs résultats [42] en traitant les bits par blocs et en réalisant le ou exclusif $\oplus$ sur certains bits des blocs. Plus précisément, il proposait de traiter les données par blocs de deux octets, notés x_1 et x_2, par des fonctions H_1, H_2 et H_3 définies par :

$$\begin{aligned} H_1(x_1, x_2) &= x_1 \oplus RL(x_1, 1) \oplus x_2, \\ H_2(x_1, x_2) &= x_1 \oplus RL(x_1, 1) \oplus RL(x_1, 2) \oplus x_2, \\ H_3(x_1, x_2) &= x_1 \oplus RL(x_1, 1) \oplus RL(x_1, 2) \oplus RL(x_1, 4) \oplus x_2, \end{aligned}$$

où $RL(x, i)$ correspond à la rotation circulaire de i bits par la droite de x. Ces fonctions avaient été trouvées de manière expérimentale et possédaient un meilleur compromis rendement/réduction du biais. Cette partie détermine les meilleures fonctions booléennes pour la réduction du biais, en démontrant et généralisant les résultats de Dichtl par l'introduction des (n, m, t)-correcteurs.

Fonctions booléennes et biais. On note $\mathbf{F}_2$ le corps fini à deux éléments, c'est-à-dire l'ensemble $\{0,1\}$ muni de l'adddition et de la multiplication modulo 2. Une fonction booléenne à n variables est une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2$. Par définition, une fonction booléenne est dite équilibrée si on a l'égalité suivante : $P(f(x) = 0) = P(f(x) = 1) = 0.5$, quand l'entrée x est aléatoire. Une fonction booléenne de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ (parfois appelée fonction vectorielle ou simplement S-box) correspond à m fonctions booléennes à n variables, que l'on note $f = (f_1, \ldots, f_m)$. Une telle fonction est dite équilibrée si et seulement si pour tout $y \in \mathbf{F}_2^m$, la probabilité $P(f(x) = y)$ est 2^{-m} si x est aléatoire, généralisant ainsi la définition précédente. C'est équivalent à ce que toute combinaison linéaire de la sortie, qui est une fonction booléenne à n variables que l'on note $uf = \sum_{i=1}^{m} u_i f_i$, soit équilibrée. Le lecteur est renvoyé au livre de C. Carlet [28, 29] pour tout ce qui touche l'utilisation des fonctions booléennes et vectorielles en cryptographie et théorie des codes.

Principe d'un correcteur. Dans cette section, l'entrée $x = (x_1, \ldots, x_n)$ de la fonction booléenne n'est pas parfaitement aléatoire : elle est biaisée, c'est-à-dire que $P(x_i = 0) = 0.5 + e$ pour tout i. Pour une fonction booléenne f de $\mathbf{F}_2^n$ dans $\mathbf{F}_2$, le biais en sortie $\Delta_f(e)$ est défini par $2\Delta_f(e) = P(f(x) = 0) - P(f(x) = 1)$. Ce biais peut être aussi vu comme un polynôme en la variable e, en réécrivant l'égalité ci-dessus :

$$2\Delta_f(e) = \sum_{x \in \mathbf{F}_2^n} (\frac{1}{2} + e)^{n - w_H(x)} (\frac{1}{2} - e)^{w_H(x)} (-1)^{f(x)},$$

où $w_H(x)$ est le poids de Hamming de x. Dans ce cas, pour que ce biais soit le plus petit possible, il faut que le plus petit coefficient non nul, appelé valuation, de ce polynôme soit le plus grand possible, qui est sans coefficient constant si et seulement si f est équilibrée, puisque le biais e est plus petit que 1.

Dans le cas d'une fonction booléenne vectorielle f de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$, pour tout m-uplet en sortie $y \in \mathbf{F}_2^m$, le biais $\Delta_{y,f}(e)$ est défini par $|P(f(x) = y) - 2^{-m}|$. Plus précisément, on le définit par le polynôme en la variable e suivant :

$$\Delta_{y,f}(e) = \sum_{x \in f^{-1}(y)} (\frac{1}{2} + e)^{n - w_H(x)} (\frac{1}{2} - e)^{w_H(x)}.$$

Une telle représentation amène à la définition suivante :

Définition 1. *une fonction booléenne équilibrée de $\boldsymbol{F}_2^n$ dans $\boldsymbol{F}_2^m$, telle que pour tout $y \in \boldsymbol{F}_2^m$, le biais $\Delta_{y,f}(e)$ soit un polynôme de valuation supérieure ou égale à t est appelée (n, m, t)-correcteur.*

Par ailleurs, on peut aussi montrer que f est un (n, m, t)-correcteur si et seulement si pour tout $u \in \mathbf{F}_2^m$ non nul, la combinaison linéaire uf est un $(n, 1, t)$-correcteur. Ceci est une conséquence de résultats en cryptanalyse linéaire tirés de T. Baignères, P. Junod et S. Vaudenay [11] et adaptés au cadre de nos correcteurs. Remarquons qu'en fait, cela avait déjà été montré 10 ans avant par N. Alon, O. Goldreich, J. Hastad et R. Peralta dans un cadre plus général [3]. Plus précisément,

si on note $\Delta_{uf}(e)$ le biais en sortie de la fonction booléenne $uf = \sum_{i=1}^{m} u_i f_i$ alors pour tout $y \in \mathbf{F}_2^m$ non nul, on a :

$$\Delta_{y,f} = 2^{-m} \sum_{u \neq 0} (-1)^{uy} \Delta_{uf}.$$

On en déduit que

$$|\Delta_{y,f}| \leq 2 \max_{u \neq 0} |\Delta_{uf}|, \tag{1.1}$$

ce qui permet d'obtenir une borne maximale sur le biais, en fonction du biais maximal entre toutes les combinaisons linéaires non nulles de la sortie.

Codes correcteurs linéaires. Avant de présenter des constructions de correcteurs, on commence par donner quelques définitions sur les codes correcteurs d'erreur.

Un code correcteur sera considéré binaire, sauf s'il est précisé le contraire. Un code binaire de paramètres (n, M, d) est un ensemble de M mots de n bits, tels que la distance de Hamming entre chaque mots est au moins d, appelée distance minimale du code. Un code linéaire binaire de paramètres $[n, m, d]$ est un sous espace vectoriel de $(\mathbf{F}_2)^n$ de dimension m et de distance minimale d. En d'autres termes, un $[n, m, d]$ code linéaire est un $(n, 2^m, d)$ code. En tant que sous espace vectoriel, tout code linéaire binaire contient le mot de code $(0, \ldots, 0)$ et la distance minimale du code correspond au plus petit poids de Hamming des mots de code non nuls.

Tout $[n, m, d]$ code linéaire peut être généré par une matrice de m lignes et n colonnes, appelée matrice génératrice du code. Plus précisément, si G est une matrice binaire composée de m lignes et n colonnes, alors l'ensemble $\{xG \ : \ x \in \mathbf{F}_2^m\}$ est un code linéaire de longueur n et dimension m. Le code dual $C^\perp$ d'un code linéaire C est le $[n, n-k, d']$ code linéaire, defini par l'ensemble $\{x \in \mathbf{F}_2^n : \forall c \in C, \ cx = 0\}$, où d' est appelée distance duale. La matrice génératrice H de $C^\perp$ est appelée matrice de contrôle de C. Enfin $\forall x \in \mathbf{F}_2^n$, on définit le syndrome de x comme le vecteur de longueur $n-k$, defini par Hx et on a $\forall c \in C, \ Hc = (0, \ldots, 0)$.

Bien entendu, il n'existe pas de codes linéaires binaires pour tout ensemble de paramètres $[n, m, d]$. De manière générale, quand deux des paramètres sont fixés, il y a une borne sur le troisième paramètre. De nombreuses bornes existent, comme la borne de Hamming pour un code binaire : $K \sum_{i=0}^{t} \binom{n}{i} \leq 2^n$ pour un code de longueur n, cardinal K pouvant corriger t erreurs. Le lecteur est renvoyé vers le livre de F.J. Mac Williams and N.J.A. Sloane pour l'ensemble de la théorie des codes [147].

Le lien entre code linéaire et correcteur linéaire se fait avec la représentation suivante : N'importe quelle fonction booléenne vectorielle linéaire de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ peut être représentée par une matrice binaire $G = (g_{i,j})$:

$$\begin{pmatrix} g_{1,1} & \cdots & g_{1,n} \\ \vdots & & \vdots \\ g_{m,1} & \cdots & g_{m,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}.$$

En utilisant la représentation précédente, on obtient le théorème suivant :

Théorème 1. *Soit f une fonction booléenne de $\boldsymbol{F}_2^n$ dans $\boldsymbol{F}_2^m$, définie par $f(x) = Gx$, produit de l'entrée x vue comme un vecteur colonne par une matrice G et soit e le biais de l'entrée. Alors le biais de n'importe quelle combinaison linéaire non nulle des bits de sortie est inférieur ou égal à $2^{d-1}e^d$, où d est la distance minimale du code linéaire de matrice génératrice G.*

En effet, si on note $y = f(x)$, en utilisant la représentation précédente, on obtient $y = Gx$. De plus, n'importe quelle combinaison linéaire uy des bits de sortie, avec $u \in \mathbf{F}_2^m$, correspond au produit scalaire du mot de code uG, appartenant au code généré par G, par x. Par définition de la distance minimale d du code, ce produit scalaire réalise donc la somme (xor) d'au moins d bits de l'entrée, ce qui implique un biais en sortie de uf d'au plus $2^{d-1}e^d$. Remarquons que la représentation précédente correspond en fait au calcul d'un syndrome.

Ce premier résultat permet de démontrer rigoureusement presque toutes les constructions de Dichtl, qui étaient toutes linéaires sauf une. Plus précisément, en réécrivant les trois constructions présentées en début de section sous forme matricielle, on en déduisait que cela revenait à utiliser des $[16, 8, 3]$, $[16, 8, 4]$ et $[16, 8, 5]$ codes linéaires à l'aide de la représentation ci-dessus. Ainsi, un (n, m, t)-correcteur linéaire est obtenu à partir de n'importe quel $[n, m, t+1]$ code linéaire par la construction décrite ci-dessus. Par exemple, la matrice suivante correspond au correcteur H_3 :

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

La figure 1.2 compare le biais maximal de chaque combinaison linéaire en sortie des trois correcteurs linéaires précédents.

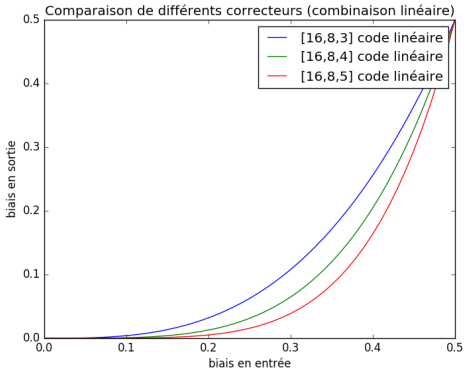


FIGURE 1.2 – Biais maximal entre toutes les combinaisons linéaires

Applications. L'utilisation de codes linéaires cycliques permet d'implémenter ces correcteurs linéaires à l'aide d'un registre à décalage à rétroaction linéaire (LFSR), dont les implémentations matérielles sont extrêmement efficaces. Par exemple, le code précédent H_3 est un code cyclique, dont le code dual est généré par le polynôme $X^8 + X^4 + X^2 + X + 1$.

De manière alternative, un code cyclique BCH de longueur 255, dimension 21 et distance minimale 111 permet d'implémenter un $(255, 21, 110)$-correcteur linéaire à l'aide d'un registre à décalage à rétroaction linéaire de longueur 21 représentant le polynôme $X^{21} + X^{19} + X^{14} + X^{10} + X^7 + X^2 + 1$ qui génère son code dual. Par exemple, pour un biais de $1/4$ en entrée, on obtient :

$$\forall y \in \mathbf{F}_2^{21} \quad \left|P(f(X) = y) - 2^{-21}\right| \leq 2^{-111}.$$

Pour tout savoir sur les codes cycliques et leur implémentation, ainsi que sur les codes BCH en particulier, on se reportera aux chapitres 7, 8 et 9 de [147].

1.2 Correcteurs non linéaires

Une des fonctions proposées par Dichtl (non décrite dans ce manuscrit) était non linéaire et était censée avoir une meilleure réduction du biais que n'importe quelle fonction linéaire. Pour étudier le cas d'une fonction booléenne f à n variables, on utilise sa transformée de Walsh :

$$\forall v \in \mathbf{F}_2^n \quad \widehat{f}(v) = \sum_{x \in \mathbf{F}_2^n} (-1)^{f(x) \oplus v.x}.$$

Dans ce cas, j'ai montré que le biais $\Delta_f(e)$ d'une fonction booléenne à n variables vérifie l'équation :

$$\Delta_f(e) = \frac{1}{2^{n+1}} \sum_{v \in \mathbf{F}_2^n} (2e)^{w_H(v)} \widehat{f}(v). \tag{1.2}$$

Cette formule permet de calculer précisément le biais en sortie à l'aide de la transformée de Walsh (dont les coefficients du polynôme en e), mais le lien étroit entre la transformée de Walsh et les fonctions booléennes résilientes va permettre en outre de généraliser le résultat de la section précédente sur les codes linéaires. Notons que les coefficients de Walsh peuvent être de taille non négligeable (voir exemple tité du code de Nordstrom-Robinson ci-dessous).

Lien entre résilience et biais. Les fonctions booléennes t-résilientes ont été introduites en 1984 par T. Sigenthaler dans [129] et en 1985 par B. Chor *et al.* pour les fonctions vectorielles [31] : une fonction booléenne équilibrée à n variables est t-resiliente si elle reste équilibrée quand on fixe t variables en entrée, quelles que soient les t entrées choisies. B. Chor *et al.* montrent de plus qu'une fonction vectorielle f de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$, est t-résiliente si et seulement si pour tout $u \in \mathbf{F}_2^m$ non nul, la fonction booléenne uf est t-resiliente. Pour finir, G. Xiao et J. L. Massey ont caractérisé les fonctions résilientes avec leur transformée de Walsh : une fonction booléenne f est

t-résiliente si et seulement si pour tout $v \in \mathbf{F}_2^m$ de poids de Hamming inférieur ou égal à t on a : $\hat{f}(v) = 0$ [148]. Les fonctions résilientes et leur utilisation en cryptographie font l'objet du chapitre 7 de [28] dans le cas des fonctions booléenes et du chapitre 4 de [29] dans le cas des fonctions vectorielles.

Le théorème suivant peut s'obtenir en combinant les résultats précédents sur les fonctions résilientes avec l'équation 1.2 ci-dessus :

Théorème 2. *une fonction booléenne vectorielle t-resiliente de $\boldsymbol{F}_2^n$ dans $\boldsymbol{F}_2^m$ est un (n, m, t) correcteur.*

Cette proposition généralise en fait la construction des correcteurs linéaires car tout code linéaire peut être utilisé pour avoir une fonction booléenne résiliente linéaire [31]. Néanmoins, il existe aussi des fonctions booléennes non linéaires qui sont résilientes. Par exemple, il n'existe pas de $(16, 8, 5)$ fonctions résilientes linéaires mais il existe une $(16, 8, 5)$ fonction résiliente non linéaire construite à partir du code de Nordstrom-Robinson [107]. Ce $(15, 256, 5)$ code a été trouvé par A. W. Nordstrom, étudiant de l'université d'Iowa, sur un problème posé par son professseur J. P. Robinson. Il est décrit sous forme systématique dans l'article original, en notant $x_0, \ldots, x_7$ les bits d'information, où les 7 bits de contrôle $y_0, \ldots, y_6$ sont calculés par $y_0 = x_7 \oplus x_6 \oplus x_0 \oplus x_1 \oplus x_3 \oplus (x_0 \oplus x_4)(x_1 \oplus x_2 \oplus x_3 \oplus x_5) \oplus (x_1 \oplus x_2)(x_3 \oplus x_5)$ et en substituant $x_{i+j \bmod 7}$ par x_i pour les autres bits y_i. On obtient un $(16, 256, 6)$ code en ajoutant un bit de contrôle de parité. La construction de la fonction résiliente, à partir de ce code est basée sur le lien étroit entre tableau orthogonaux et fonctions résilientes, telle que décrite par J. L. Massey et D. Stinson [135].
Cette fonction résiliente non linéaire peut servir directement de $(16, 8, 5)$-correcteur non linéaire. Si on calcule la transformée de Walsh des 255 fonctions booléennes correspondant aux combinaisons linéaires des bits de sortie de cette $(16, 8, 5)$ fonction résiliente, on obtient dans le pire cas quatre coefficients non nuls, trois à 2^{15} et un à -2^{15}, tous pour des entrées de poids six. Dans ce cas, en utilisant l'équation 1.2, on obtient un biais en sortie de $2^5 e^6$. On retrouve ainsi la même formule que dans le cas linéaire, mais avec des paramètres qui ne peuvent pas être atteints avec des codes correcteurs linéaires.

Entropie minimale. Soit X une variable aléatoire discrète sur $\{0, 1\}^n$. L'entropie minimale de X est le plus grand entier k tel que $\forall x \in X,\ P(X = x) \leq 2^{-k}$. L'entropie minimale est une mesure de l'entropie plus stricte que l'entropie au sens de Shannon. L'équation 1.1 combinée avec le théorème 2 permet d'obtenir une borne sur l'entropie minimale de la sortie d'un (n, m, t)-correcteur.

Suite de ces travaux. Plusieurs articles ont été publiés pour donner suite à ces travaux, notamment sur les correcteurs linéaires. Ainsi, une étude du comportement asymptotique a été proposée en 2011 par H. Zhou et J. Bruck dans [150]. A. Meneghetti, M. Sala et A. Tomasi ont repris la partie sur l'entropie minimale et étudié en particulier l'utilisation des codes de Reed-Muller en 2014 dans [99]. Le cas des codes BCH est étudié en 2019 par H. Park, Y. Yeom et J.-S. Kang dans [111] pour des entrées qui ne sont plus indépendantes.

Distribution de poids dual et biais. Tout comme pour les codes correcteurs d'erreurs, il existe aussi des bornes sur les paramètres des fonctions résilientes. L'objectif est donc de chercher une construction plus générale des correcteurs qui ne soit pas basée sur ces fonctions. Cette partie a été réalisée pendant mes deux années d'ATER à l'université de Toulon. En particulier, on décrit dans cette section le lien entre distribution de poids duale d'un code et les correcteurs.

Soit C un (n, M, d) code binaire. Sa distribution de poids est $(A_0, \dots A_n)$, où A_i est le nombre de mots de code de poids i. La distribution des distances $(B_0, \dots B_n)$ est définie par

$$B_i = \frac{1}{M}\#\{(x, y) \in C \times C \mid d_H(x, y) = i\}.$$

En particulier, on a $\sum_{i=0}^{n} A_i = \sum_{i=0}^{n} B_i = M$ et on montre que si le code est linéaire les deux distributions sont identiques. Le code étant non linéaire, il n'y a pas de code dual, la distribution de poids duale du code C est $(A'_0, \dots A'_n)$ est donc définie formellement par la transformée de Mac Williams de la distribution de poids :

$$A'_k = \frac{1}{M}\sum_{i=0}^{n} A_i P_k(i),$$

où $P_k(i)$ est le polynôme de Krawtchouk défini par

$$P_k(i) = \sum_{j=0}^{k} (-1)^j \binom{i}{j}\binom{n-i}{k-j}.$$

Distance duale. De même, la distribution de distance duale du code C est définie formellement par $(B'_0, \dots B'_n)$, où

$$B'_k = \frac{1}{M}\sum_{i=0}^{n} B_i P_k(i).$$

Le plus petit entier d tel que $B'_d \neq 0$ est appelé distance duale du code, introduite par P. Delsarte en 1973 dans [38]. Si le code est linéaire, cette définition correspond à la distance minimale du code dual. Les chapitres 5 et 6 de [147] sont consacrés aux problématiques liées aux définitions ci-dessus.

Soit f une fonction booléenne équilibrée de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Alors, pour tout $y \in \mathbf{F}_2^m$, on peut voir l'ensemble $f^{-1}(y)$ comme un code de longueur n et de cardinal 2^{n-m}. Il a notamment été démontré par J. Bierbrauer, K. Gopalakrishnan et D. Stinson dans [17] que si f est une fonction t-résiliente, alors $f^{-1}(y)$ est un code correcteur dont la distance duale est $t + 1$.

On obtient un résultat similaire à l'aide cette fois de la distribution de poids dual, afin de construire des correcteurs qui ne sont pas construits à partir de fonctions résilientes, comme présenté ci-dessous.

On définit la fonction indicatrice $\phi_x(y)$ du code $f^{-1}(y)$ par

$$\phi_x(y) = \begin{cases} 1 \text{ si } f(x) = y \\ 0 \text{ si } f(x) \neq y \end{cases}$$

Dans ce cas, la distribution de poids de $f^{-1}(y)$ est définie par $A_i = \sum_{w_H(x)=i} \phi_x(y)$, et comme f est équilibrée, alors on a : $\sum_{i=0}^{n} A_i = 2^{n-m}$. Par définition de $\phi_x(y)$:

$$\begin{aligned} P(f(x) = y) &= \sum_x \phi_x(y)(\frac{1}{2} + e)^{n-w_H(x)}(\frac{1}{2} - e)^{w_H(x)} \\ &= 2^{-n} \sum_{k=0}^{n} (2e)^k \sum_{i=0}^{n} A_i P_k(i) = 2^{-m} \sum_{k=0}^{n} A'_k (2e)^k, \end{aligned}$$

où les dernières lignes viennent de [147], chapitre 5, et de la définition de la distribution de poids duale du code qui est proposée. On en déduit le résultat suivant :

Théorème 3. *un (n, m, t)-correcteur est équivalent à un ensemble de 2^m codes disjoints de longueur n et de cardinal 2^{n-m}, dont la distribution de poids duale de chaque code vérifie $A'_i = 0$ pour tout $0 \leq i \leq t$.*

On remarque que si le code est linéaire, la notion de poids dual correspond à celle de distance duale, le résultat précédent correspond donc à la construction de correcteurs linéaires de la partie précédente. Néanmoins, on peut construire des correcteurs qui ne sont pas obtenus par des codes linéaires ou des fonctions résilientes. Par exemple, les deux ensembles

$$\begin{pmatrix} 000 \\ 100 \\ 101 \\ 111 \end{pmatrix} \begin{pmatrix} 010 \\ 001 \\ 011 \\ 110 \end{pmatrix}$$

correspondent à la fonction booléenne équilibrée non linéaire $f(x) = f(x_1, x_2, x_3) = x_2 \oplus x_3 \oplus x_1 x_2 \oplus x_2 x_3$, qui est $(3, 1, 1)$-correcteur mais qui n'est pas 1-résiliente.

Constructions secondaires. Il est utile de pouvoir construire des correcteurs à partir de correcteurs existants. On parle ainsi de constructions secondaires comme c'est déjà le cas pour les codes correcteurs d'erreurs. En appliquant les constructions secondaires des codes correcteurs d'erreurs linéaires ou des fonctions booléennes résilientes, on obtient de nouveaux correcteurs linéaires, ou basés sur des fonctions résilientes. Néanmoins, on peut aussi construire des correcteurs non linéaires qui ne sont pas basés sur ces premières constructions.

Par exemple, soient f_1 un (n_1, m, t_1)-correcteur et f_2 un (n_2, m, t_2)-correcteur, alors la fonction booléenne $f_1(x_1) \oplus f_2(x_2)$ est un $(n_1 + n_2, m, t_1 + t_2)$-correcteur, tandis que $(f_1(x_1), f_2(x_2))$ est un (n_1+n_2, m_1+m_2, t)-correcteur avec $t = \min(t_1, t_2)$. De même, si f est un (n, m, t)-correcteur, alors $(f(x_1) \oplus f(x_2), f(x_2) \oplus f(x_3)$ est un $(3n, 2m, 2t + 1,)$-correcteur. En généralisant cette procédure, on obtient pour tout $h \geq 2$ un $((h + 1)n, hn, 2t + 1)$-correcteur par $(f(x_1) \oplus f(x_2), \ldots, f(x_h) \oplus f(x_{h+1})$, puis en appliquant cette procédure k fois au résultat, on obtient pour tout $k \geq 1$ un $((h + 1)^k n, h^k n, 2^k(t + 1) - 1)$-correcteur.

Entrées non identiquement distribuées. Le cas où les entrées sont biaisées avec un biais différent a aussi été étudié. Il est montré que si f est une fonction booléenne à n variables et si on note e_i le biais de l'entrée x_i alors le biais en sortie vérifie :

$$\Delta_f = \frac{1}{2^{n+1}} \sum_{v \in \mathbf{F}_2^n} \prod_{\substack{i=1 \\ v_i=1}}^{n} (2e_i) \widehat{f}(v) \ ,$$

qui généralise l'hypothèse d'un biais constant, mais n'est pas vraiment utile en soit.

Suite à ce travail. Une suite à ce travail sur des entrées non identiquement distribuées a été proposée en 2017 par A. Tomasi, A. Meneghetti et M. Sala [139].

Distance minimale et codes poinçonnés. Une partie de mon travail de thèse a pris en compte l'hypothèse ou certain bits d'entrée du correcteur puissent être attaqués, ce qui pouvait se modéliser à l'aide des codes poinçonnés (définis dans le chapitre 1 de [147]). Ce travail réalisé en collaboration avec mon directeur Philippe Langevin n'a jamais été publié (voir [90] pour les détails et les démonstrations). Il consistait à donner une borne minimale sur la distance minimale du code ainsi poinçonné. Nous y avons en particulier démontré le théorème suivant dans le cadre des codes de Reed-Muller (voir section suivante pour la définition de ces codes) :

Théorème 4. *La distance minimale d_1^* d'un code de Reed-Muller $RM(1,m)$ qui est poinçonné N fois vérifie $d_1^* \geq (N - 2^{m/2}\ln(2^m - 1))/2$. La distance minimale d_2^* d'un code de Reed-Muller $RM(2,m)$ qui est poinçonné N fois vérifie (avec $q = 2^m$) :*

$$d_2^* \geq \min_{k \geq 0} \{ \frac{N}{2}(1 - \frac{\sqrt{2^k q}}{q-1}) - \frac{1}{2}(\sqrt{2(q-2^k)\sqrt{q} + q + 2^k - 1})\ln(q-1) \}.$$

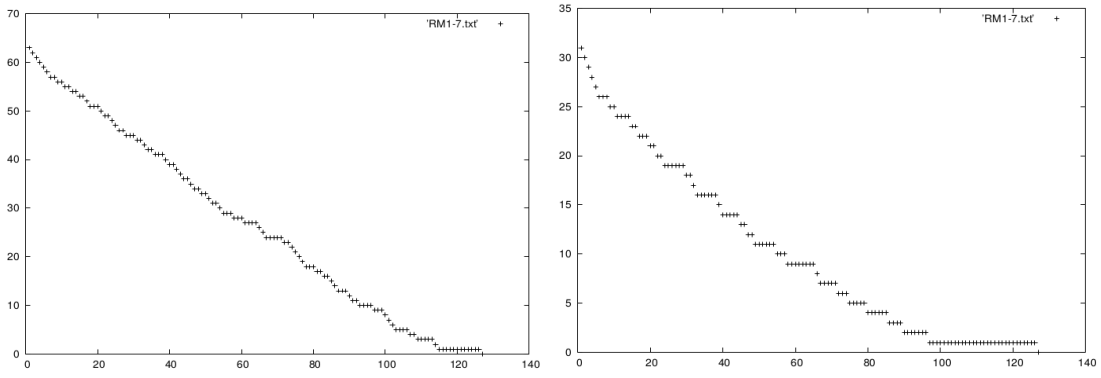


FIGURE 1.3 – Distance minimale des codes poinçonnés $RM(1,7)$ et $RM(2,7)$

La figure 1.3 illuste la baisse de la distance des codes de Reed-Muller $RM(1,7)$ et $RM(2,7)$ quand ils sont poinçonnés. Les paramètres du $RM(1,7)$ sont $[128, 8, 64]$ et ceux du $RM(2,7)$ sont $[128, 21, 32]$. L'ordonnée correspond à la distance du code obtenu en le poinçonnant N fois où N se lit en abscisse.

1.3 Codes de Kerdock et engagements flous

Engagements flous. Un schéma d'engagement flou (*fuzzy commitment*) est un système de protection des données biométriques, proposé par A. Juels et M. Wattenberg en 1999 [81]. La sécurité d'une donnée biométrique binaire u est obtenue en réalisant le xor bit à bit de cette donnée avec un mot de code aléatoire c. Les données stockées sont $u \oplus c$ et $H(c)$ où H est une fonction de hachage. Lors de la phase de vérification on calcule $u' \oplus u \oplus c$ où u' est la nouvelle donnée biométrique. Si celle-ci provient du même utilisateur, en notant $e = u \oplus u'$, on s'attend à ce que le poids de Hamming $w_H(e)$ soit inférieur à la capacité de correction du code si u et u' viennent du même utilisateur, ce qui permet de décoder et de retrouver c (en testant avec la fonction de hachage) pour authentifier la personne.

Sécurité des engagements flous. Le cardinal du code doit bien sûr être suffisament important pour éviter une recherche exhaustive. La première implémentation d'un tel schéma, basée sur la combinaison d'un [32, 20] code de Reed-Solomon code et d'un [64, 7] code de Hadamard, était proposée en 2005 par F. Hao, R. Anderson et J. Daugman sur des iriscodes dans [69]. Une attaque par indistinguabilité sur ce shéma est proposée en 2009 par K. Simoens, P. Tuyls et B. Preneel dans [130]. L'attaque est basée sur le fait que si un attaquant possède $u \oplus c$ et $u' \oplus c'$ où u, u' sont deux données biométriques et c, c' deux mots d'un même code, alors il calcule $u \oplus u' \oplus c \oplus c'$. Comme $c \oplus c'$ est un mot du code, si le poids de $e = u \oplus u'$ est au dela de la capacité de correction du code $t = \lfloor (d-1)/2 \rfloor$, où d est la distance minimale du code, alors l'attaquant sait que u et u' ne viennent pas du même individu (dans le cas contraire il ne peut pas conclure). Par exemple, le code de Hadamard utilisé dans [69] est vulnérable à cette attaque.

Utilisation d'un code non linéaire dans un engagement flou. L'attaque ci-dessus est possible car la somme de deux mots d'un code linéaire reste dans le code. L'idée est donc de s'intéresser à l'utilisation un code non linéaire pour éviter une telle attaque. Clairement, si on utilise un code non linéaire, la somme de deux mots de codes n'est pas forcément un mot de code, mais ça peut être le cas pour deux mots donnés. Pire, si $c \oplus c'$ n'est pas un mot de code mais que $c \oplus c' \oplus e'$ l'est avec $w_H(e')$ petit, alors l'attaque est encore potentiellement faisable. Plus précisément, si $e \oplus e' \leq t = \lfloor (d-1)/2 \rfloor$ l'attaque est possible et si $e' \oplus e > t = \lfloor (d-1)/2 \rfloor$, alors l'attaque n'est pas possible. Il serait néanmoins intéressant d'avoir une borne indépendante des données (ci-dessus e dépend de u et u' et e' de c et c').

Cela amène à introduire la définition de la distribution de non- linéarité d'un code :

Définition 2. *Pour un (n, K, d) code C, de longueur n, cardinal K et distance minimale d, on définit la distribution de non-linéarité $D = (D_0 \ldots, D_n)$ par*

$$D_i = \frac{1}{K} \sharp\{(c_1, c_2) \in C \ \mid d_H(c_1 \oplus c_2, C) = i\},$$

où $d_H(c_1 \oplus c_2, C) = \min_{c \in C} d_H(c_1 \oplus c_2, c)$.

Cette distribution décrit donc la distance entre la somme de chaque paire de mots de code et le code lui même. Bien sûr, si le code est linéaire, on a $D_0 = K$, tandis que les autres coefficients sont nuls.

Cas d'un code aléatoire. Pour un code aléatoire de longueur n, cardinal $K = 2^k$ et distance minimale d (pouvant corriger $t = \lfloor (d-1)/2 \rfloor$ erreurs), la probabilité de résister à l'attaque est supérieure à $P(d_H(x, C) > t)$, avec $x \in \mathbf{F}_2^n$, qui est exactement la probabilité de succès de l'attaque sur un code linéaire. La figure 1.4 présente une distribution de non linéarité d'un code aléatoire de distance minimale d supérieure à $t = \lfloor (d-1)/2 \rfloor$, ce qui est plutôt encourageant.

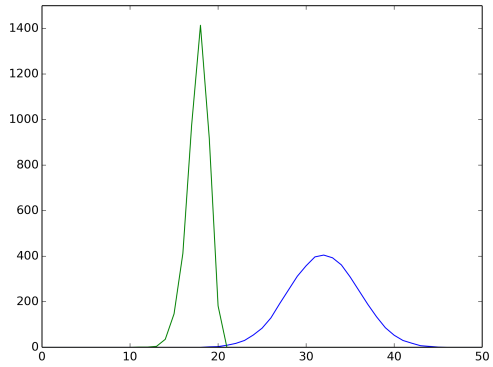


FIGURE 1.4 – Distribution des distances (bleu) et de non linéarité (vert) pour un code non linéaire aléatoire de longueur 64 et cardinal 4096.

Codes de Reed-Muller. Le code binaire de Reed-Muller $RM(r, m)$ est un code linéaire de longueur $n = 2^m$, composé de tout les mots de codes représentant une fonction booléenne à m variables de degré au plus r. Ces codes ont été proposés en 1954 par D. Muller, puis étudiés par I. Reed (algorithme de décodage [122]). La dimension du code $RM(r, m)$ est $1 + \binom{m}{1} + \ldots + \binom{m}{r}$ et sa distance minimale est 2^{m-r}. La distribution de poids de $RM(2, m)$ est donnée dans [147], chapitre 14.

Non linéarité et fonctions courbes. La non linéarité d'une fonction booléenne f à m variables est la distance minimale entre f et $RM(1, m)$, ce qui correspond à la distance minimale du code $RM(1, m) \cup (f \oplus RM(1, m))$, où le coset $f \oplus RM(1, m)$ est défini par l'ensemble $\{f \oplus c : c \in RM(1, m)\}$. Une fonction booléenne f à m variables est courbe si et seulement si m est pair et si la distance de Hamming entre f et $RM(1, m)$ est $2^{m-1} - 2^{\frac{m}{2}-1}$ ou $2^{m-1} + 2^{\frac{m}{2}-1}$. Ces fonctions ont été introduites par J. Dillon en 1974 [43]. La non linearité d'une fonction booléenne à m variables est inférieure à $2^{m-1} - 2^{m/2-1}$ avec égalité si et seulement si f est courbe. Les cosets de $RM(1, m)$ ont été étudiés en 2001 par A. Canteaut, C. Carlet, P. Charpin et C. Fontaine dans [27] : ainsi, pour $f \in RM(2, m)$, la distribution de poids du coset $f + RM(1, m)$ prend trois valeurs non nulles $\{2^{m-1}, 2^{m-1} \pm 2^{m-h-1}\}$, avec $1 \leq h \leq m/2$ (où $2h$ étant le rang de la forme de symplectique associée à f), sauf quand $h = m/2$ où il n'y a plus que deux valeurs $\{\pm 2^{m/2}\}$, cas où f est courbe.

Codes de Kerdock. Soit $N = 2^{m-1} - 1$ et $f_1, \ldots, f_N$ des fonctions quadratiques courbes telles que la somme de chaque paire $f_i \oplus f_j$ soit courbe. Alors l'ensemble $\{f_1, \ldots, f_N\}$ est appelé ensemble de Kerdock [16]. Remarque : $f_i \oplus f_j$ n'est pas forcément dans l'ensemble de Kerdock. Le code de Kerdock $K(m)$, avec m pair, est le code construit à partir d'un ensemble de Kerdock, composé de $RM(1,m)$ et de $2^{m-1}-1$ cosets $f_i \oplus RM(1,m)$ de $RM(2,m)$, qui est donc un sous code de $RM(2,m)$. En d'autres termes $K(m) = RM(1,m) \cup (f_1 \oplus RM(1,m)) \cup \ldots \cup (f_N \oplus RM(1,m))$.

Le code de Kerdock $K(m)$ est un $(2^m, 2^{2m}, 2^{m-1} - 2^{\frac{m}{2}-1})$ code non linéaire car le nombre de ces cosets est $2^{m-1} - 1$, chacun de cardinal 2^{m+1}, ce qui implique $(2^{m-1} - 1 + 1)2^{m+1} = 2^{2m}$ mots de codes. La distance minimale vient du fait que la non linéarité d'une fonction courbe est $d = 2^{m-1} - 2^{\frac{m}{2}-1}$ et que la somme de deux fonctions courbes du code est une fonction courbe. Par exemple, les paramètres de $K(4)$ sont $(16, 256, 6)$ et ceux de $K(6)$ sont $(64, 4096, 28)$.

Les ensembles de Kerdock et les codes associés sont proposés par A. M. Kerdock en 1972 dans [86] et décrits dans [147], chapitre 15. On y trouve notamment que la distribution de poids de $K(m)$ est la même que la distribution des distances et qu'elle est décrite par $A_0 = A_{2^m} = 1$, $A_d = A_{2^m-d} = 2^{m-1}(2^m - 2)$ et $A_{2^{m-1}} = 2^{m+1} - 2$. Le nombre asymptotique de ces codes est connu pour être exponentiel, comme étudié par W.M. Kantor en 1982 [83]. Remarquons que le code de Nordstrom-Robinson vu dans la section précédente correspond en fait au code de Kerdock $K(4)$.

Distribution de non linéarité d'un code de Kerdock. Pour qu'un code de Kerdock soit intéressant pour la construction d'un engagement flou, il faut que les petits coefficients de sa distribution de non linéarité soient le plus petit possible. La distribution de non linéarité du code de Kerdock $K(m)$, de cardinal est 2^{2m}, se décrit avec un ensemble de 2^{2m} fonctions booléennes $f_1, \ldots f_{2^{2m}}$ par

$$D_i = \frac{1}{2^{2m}} \sharp\{(f_1, f_2) \in K(m)^2 \mid \min_{f \in K(m)} \{d_H(f_1 \oplus f_2, f) = i\}.$$

Cette distribution est décrite par le théorème suivant :

Théorème 5. *Soit m pair et K l'ensemble de Kerdock $\{f_1, \ldots, f_{2^{m-1}-1}\}$, composé par $2^{m-1}-1$ fonctions courbes définissant le code de Kerdock $K(m)$, où l'on suppose que si f_i et f_j sont dans K alors $f_i \oplus f_j \notin K$. Alors la distribution de non linéarité de $K(m)$ est donnée par $D_0, \ldots, D_{2^m}$ où tous les coefficients compris entre D_1 et $D_{2^{m-2}-1}$ sont nuls, $D_0 = 2^{m+1} + 2^{m+2} - 8$ et $\sum_{i \geq 2^{m-2}} D_i = (2^m - 2)(2^m - 4)$.*

Preuve : soient c_1 et c_2 deux mots de code aléatoires de $K(m)$. Si c_1 (resp. c_2) est dans $RM(1,m)$ et c_2 (resp. c_1) est dans $f_i \oplus RM(1,m)$, alors $c_1 \oplus c_2$ est dans $f_i \oplus RM(1,m)$. Deuxièmement, si c_1 et c_2 viennent du même coset $f_i \oplus RM(1,m)$ alors $c_1 \oplus c_2$ est dans $RM(1,m)$. Finalement, si c_1 est dans $f_i \oplus RM(1,m)$ et c_2 est dans $f_j \oplus RM(1,m)$, avec $i \neq j$, alors $c_1 \oplus c_2$ est dans le coset $f_i \oplus f_j \oplus RM(1,m)$ qui n'est pas dans $K(m)$, par hypothèse. Dans ce cas, la distance de Hamming $d_H(c_1 \oplus c_2, K(m)) \geq 2^{m-2}$, directement par la distance minimale de $RM(2,m)$.

Le nombre de paires (c_1, c_2) avec $c_1 \in RM(1,m)$ est $2^{m+1}2^{2m}$. Le nombre de paires (c_1, c_2) avec $c_1 \in f_i \oplus RM(1,m)$ et $c_2 \in RM(1,m)$ ou $c_2 \in f_i \oplus RM(1,m)$ est $(2^{m-1}-1)2^{m+1}2^{m+1}2 = 2^{3m+2} - 2^{2m+3} = 2^{2m}(2^{m+2}-8)$. Ainsi, $D_0 = 2^{m+1} + 2^{m+2} - 8$ (après division par 2^{2m} pour normaliser). Inversement, $\sum_{i \geq 2^{m-2}} D_i = (2^m - 4)(2^m - 2)$ $= 2^{2m} - 2^{m+2} - 2^{m+1} + 8$, ce qui termine la preuve.

On rappelle que l'attaque précédente n'est pas possible si la distance entre la somme de deux mots de code et le code est supérieure ou égale à $t = \lfloor \frac{d-1}{2} \rfloor < 2^{m-2}$, dans le cas des codes de Kerdock. Ainsi, $D_0/2^{2m} = 2^{1-m} + 2^{2-m}$ - 2^{3-2m} est exactement la probabilité de réaliser l'attaque, tandis que $\sum_{i \geq 2^{m-2}} D_i/2^{2m} = 1$ - 2^{2-m} - 2^{1-m} + 2^{3-2m} est la probabilité de résister à l'attaque. Clairement, $D_0/2^{2m}$ devient asymptotiquement négligeable devant $\sum_{i \geq 2^{m-2}} D_i/2^{2m}$.

Recherche par clique des codes de Kerdock $K(4)$**.** Il existe 28 cosets $f_i \oplus RM(1,4)$ dans $RM(2,4)$, où les f_i sont des fonctions quadratiques courbes à 4 variables (sans les parties linéaires). Soit $G(4)$ le graphe composé de 28 sommets f_i, où une arête entre deux sommets f_i et f_j signifie que $f_i \oplus f_j$ est courbe. Une recherche exhaustive de cliques dans ce graphe donne beaucoup de cliques d'ordre 3 et huit cliques d'ordre 7. Les cliques d'ordre 3 sont composées de triplets $(f_i, f_j, f_i \oplus f_j)$, tandis que les cliques d'ordre 7 sont des ensembles de fonctions booléennes dont la somme de chaque paire n'est pas dans l'ensemble. Chacune de ces huit cliques donne donc un ensemble de Kerdock de cardinal $7 = 2^{4-1} - 1$ pour un code de Kerdock $K(4)$, qui vérifie dans tous les cas la distribution $D_0 = 88$ and $D_4 = 168$.

Mise en oeuvre. Des $[16, 7, 32]$ codes linéaires sont utilisés pour la construction d'engagements flous (Hadamard $H(6)$ et Reed Muller $RM(1,6)$), en combinaison avec un second code dans les implémentations proposées [69, 22]. Si on compare les paramètres $(2^m, 2^{2m}, 2^{m-1} - 2^{\frac{m}{2}-1})$ d'un code de Kerdock $K(m)$ et ceux des codes de Hadamard $H(m)$ et Reed Muller $RM(1,m)$, de paramètres $[2^m, m+1, 2^{m-1}]$, alors la distance minimale de $K(m)$ est inférieure, mais le cardinal est bien plus grand.

La construction d'un code de Kerdock comme l'image d'un code cyclique sur $\mathbb{Z}_4$ par l'application de Gray est proposée par A. R. Hammons Jr., P. V. Kumar, A. R. Calderbank, N. J. A. Sloane et P. Solé en 1994 [80]. Cette construction fournit un code équivalent aux codes de Kerdock Elle a été testée expérimentalement sur $K(4)$ et $K(6)$ et les distributions suivantes de non linéarité ont été obtenues : $D_0 = 88$ et $D_4 = 168$ pour $K(4)$, et $D_0 = 374$ et $D_{16} = 3720$ pour $K(6)$, ce qui vérifie le théorème 5.

Soit $g(X) = \sum_{i=0}^{25} g_i X^i$ le polynôme de degré 25 de $\mathbb{Z}_4[X]$ défini par les coefficients $g_0, \ldots, g_{25} = 1, 1, 1, 2, 0, 1, 2, 2, 0, 1, 0, 3, 0, 3, 1, 3, 3, 0, 1, 3, 2, 1, 2, 2, 1, 3$. On considère le code cyclique sur $\mathbb{Z}_4$ de longueur $n = 2^{m-1} - 1 = 31$ avec $m = 6$, généré par $g(X)$, c'est-à dire que le code contient l'ensemble des polynômes $m(x)g(x) \bmod X^n - 1$ où $m(X)$ est un polynôme de $\mathbb{Z}_4[X]$ de degré inférieur à $n - 25 = 6$. Il y a donc $4^6 = 2^{12} = 4096$ mots de code, que l'on étend à l'aide d'un symbole de parité pour obtenir un code cyclique sur $\mathbb{Z}_4$ de longueur 32 et de dimension 6. Ce code est ensuite transformé en un code binaire par l'application de Gray pour obtenir le

code de Kerdock binaire $K(6)$ de longueur 64 et cardinal 4096 comme décrit par Hammons *et al.* dans [80]. L'application de Gray $\phi : \mathbb{Z}_4 \to \mathbf{F}_2^2$ se calcule simplement par $\phi(0) = 00$, $\phi(1) = 10$, $\phi(2) = 11$ et $\phi(3) = 01$, la construction précédente est donc efficace car elle demande essentiellement le calcul d'un produit de polynômes qui peut se faire à l'aide d'un registre à décalage à rétroaction linéaire sur $\mathbb{Z}_4$, tandis que le décodage correspond encore à celui d'un code cyclique après avoir appliqué la transformation de Gray inverse.

Conclusion et perspectives. La construction précédente d'un engagement flou date de 2015 et reste inachevée car l'utilisation de $K(4)$ ou $K(6)$ doit se faire en combinaison avec un autre code correcteur d'erreurs comme ceux proposés dans [69, 22]. Il faut plutôt la voir comme une brique de base, en remplacement des codes linéaires de Hadamard $H(6)$ ou de Reed-Muller $RM(1, 6)$, qui soit résistante à l'attaque décrite en début de section. Il reste donc à étudier la sécurité du schéma global.

Les engagements flous n'ont pratiquement plus été étudiés ces dix dernières années (notons toutefois le travail de T. Ignatenko et F.M.J. Willems en 2010 [74]). Une contre mesure à l'attaque, utilisant une matrice de permutation publique, a été proposée par E.J.C. Kelkboom *et al.* en 2011 [84], mais celle-ci est de nouveau attaquée par B. Tams en 2014 [137]. Il n'est toutefois pas clair que cette seconde attaque fonctionne sur un engagement basé sur un code de Kerdock, l'utilisation d'une telle matrice de permutation pourrait aussi être ajouté au schéma. Notons pour finir que ces constructions peuvent aussi servir à dériver une clé de chiffrement à partir d'une donnée biométrique.

Chapitre 2

Données personnelles et anonymat

Sommaire

2.1 Protection des données de e-santé 20

2.2 Pass de transport anonyme et intraçable 23

Aude Plateaux a fait sa thèse cifre entre septembre 2010 et septembre 2013 sur la protection des données personnelles [114]. J'ai co-encadré cette thèse qui a été dirigée conjointement par Christophe Rosenberger et Kumar Murty. Les deux cas d'usages principaux de la thèse concernent les données de e-santé et celles échangées lors d'un paiement en ligne. Le système de protection des données de e-santé a été publié dans [115, 118], où seule la partie chiffrement des données est présentée dans ce manuscrit. Elle est basée sur un système de partage de secrets à seuil. Dans cette première partie, je compare ce système avec un système de partage de secret différent. La partie paiement a été publiée dans [116, 117, 119].

Durant la même période, j'ai été responsable scientifique pour le GREYC de l'ANR LYRICS (portée par Jacques Traoré à Orange entre 2011 et 2014) qui avait pour but la mise en oeuvre de protocoles préservant les données personnelles dans les transactions sur mobiles. A la fin de ce chapitre, je présente un pass de transport anonyme et non traçable avec un procédé pour lever l'anonymat en cas de fraude. Le protocole utilise notamment une signature de groupe (les utilisateurs du système), combinée avec l'utilisation d'un élément sécurisé (délégation de calculs). La solution obtenue respecte la contrainte pratique des standards de transport actuels qui est que la validation du titre doit être réalisée en moins de 300 ms. Ce travail collaboratif, que j'ai présenté à APVP 2014, a été publié dans [7, 6] où Jacques Traoré, Sébastien Canard et Roch Lescuyer ont aussi participé aux publications.

2.1 Protection des données de e-santé

Un système d'information de e-santé contient de multiples données personnelles, pouvant être très sensibles. Ces données font d'ailleurs l'objet d'une section entière (articles 64 à 77) dans la loi Informatique et Liberté [33]. Celles-ci peuvent être accessibles par plusieurs types de personnel avec une certaine granularité dans le contrôle d'accès. Une architecture centralisée avec un simple système de contrôle d'accès pose des problèmes de sécurité et de protection de la vie privée. Des solutions cryptographiques, basées sur les accréditations anonymes [37] ou les systèmes de rechiffrement [8] ont été proposées dans ce cas d'usage (voir aussi les thèses de J. Devigne et R. Lescuyer sur ces protocoles [96, 41]). Néanmoins, l'utilisation d'un système à clé publique peut s'avérer inefficace sur un très grand nombre de données.

Dans cette partie, on s'intéresse à l'utilisation d'un système de partage de secrets pour répondre à cette problématique. L'accès aux données médicales doit être contrôlé par un mécanisme plutôt basé sur les attributs (ABAC) que sur les rôles (RBAC) [72]. En particulier, les données doivent être chiffrées par une clé qui ne peut être connue que par le personnel autorisé. Nous avons fait le choix d'un système de chiffrement symétrique pour son efficacité car les données peuvent être importantes.

Sans perte de généralité, nous considérons ici trois catégories de personnel : médecin, infirmières, patients et deux types de données : les données de prescription et celles de diagnostic, auxquelles l'infirmière n'a pas accès. On rajoute les contraintes suivantes : deux patients peuvent être suivis par les même personnes ou des personnels différents, et chaque entité ne doit posséder qu'un seul secret. Pour gérer le système on considère aussi une autre entité appelée serveur qui peut posséder plusieurs secrets, collabore avec tout le personnel mais qui ne doit pas pouvoir accéder aux données.

Système de partage de secret. Un système de partage de secrets, tel que décrit par B. Chor et E. Kushilevitz en 1993 [32], peut se définir comme ci-dessous. Soient $\mathcal{P} = \{p_1, \ldots, p_n\}$ un ensemble de parties et $\mathcal{F}_n$ une famille de sous ensembles de $\mathcal{P}$. On dit que $\mathcal{F}_n$ est monotone si $F \in \mathcal{F}_n$ et $F \subset F'$ implique $F' \in \mathcal{F}_n$. On dit que $\mathcal{F}_n$ est un système de partage de secrets sur un corps fini $\mathbf{F}_q$ s'il existe une application $\Pi : \mathbf{F}_q \to B_1 \times \ldots \times B_n$, de l'ensemble des secrets vers l'ensemble des tokens (n-uplets) tel que :

1. $\forall F \in \mathcal{F}_n$, il existe une fonction $h_F : \times_{i \in F} B_i \to \mathbf{F}_q$ tels que pour tout ensemble de tokens $\{s_1, \ldots s_n\} = \Pi(a)$ on retrouve le secret a avec $h_F(\{s_i\}_{i \in F}) = a$.
2. $\forall F \notin \mathcal{F}_n$, $\forall a_1, a_2 \in \mathbf{F}_q$, , pour tout ensemble de tokens $\{s_i\}_{i \in F}$, on a l'égalité $P(\{s_i\}_{i \in F} | a_1) = P(\{s_i\}_{i \in F} | a_2)$.

Remarquons que la définition décrite ci-dessus est, en fait, un cas particulier de la définition donnée dans [32], corrrespondant au cas parfait. Par ailleurs, dans le cas où F_n contient tous les sous ensembles de taille k, on obtient un schéma à seuil (*threshold schemes*), comme l'étaient les premiers systèmes de partage de secrets, tels que ceux proposés en 1979 par G.R. Blakley [20] ou A. Shamir [127].

Partage de clés de Shamir. Soient q un grand entier et $a \in \mathbf{F}_q$ le secret. On génère aléatoirement $t-1$ éléments $a_1, \ldots, a_{k-1}$ de $\mathbf{F}_q$ et on considère le polynôme $P(X) = a + a_1 X + \ldots + a_{t-1} X^{t-1}$. On génère ensuite n couples de tokens $s_i = (i, P(i))$, où $i \in \mathbf{F}_q$, et chaque partie du système récupère un de ces couples. Avec t couples de points, on retrouve le polynôme P par interpolation, et donc la clé a en particulier, tandis qu'avec moins de t points on ne peut pas retrouver P (on a donc un système à seuil). Le système de protection des données de e-santé proposé par Aude Plateaux dans sa thèse, et décrit ci-dessous, est basé sur ce schéma de Shamir.

Protection des données avec un partage de secret à seuil. On cache la clé de chiffrement d'une donnée de prescription dans un polynôme de degré un qui peut être retrouvé par le médecin, le patient ou l'infirmière en s'associant avec le serveur : chaque entité possède un point sur la droite. Pour chiffrer une donnée de diagnostic, le docteur et le patient ont un point sur un autre polynôme ne passant pas par le point de l'infirmière. On utilise un polynôme de degré deux où le serveur possède deux points supplémentaires (figure 2.1, gauche).

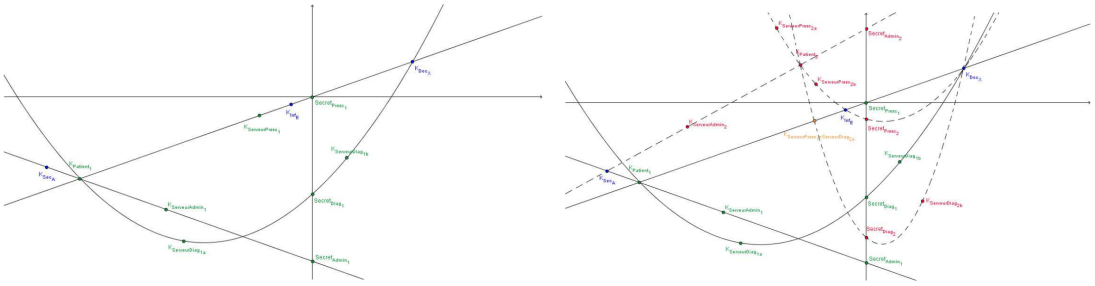


FIGURE 2.1 – Partage de clés pour un patient ou deux patients [114]

Dans le cas de deux patients suivis par les même personnels, la clé de prescription du second patient est placée dans un polynôme de degré deux afin de relier les clés du médecin, de l'infirmière et celle du second patient. On utilise encore deux nouvelles clés pour le serveur (figure 2.1, droite).

Analyse du système. On peut se questionner sur la pertinence d'utiliser un schéma à seuil dans ce type de cas d'usage. Par exemple, il n'y a aucun intéret pour deux parties de collaborer indépendament du serveur, puisque le rôle de celui-ci est de rendre le système disponible à tout moment. Par ailleurs, une solution alternative à celle proposée ici aurait été l'utilisation de la cryptographie à seuil (Y. Desmedt et Y. Frankel [39], T. Pederson [112]).

Comparaison avec un autre système de partage de secret. Les partages de secrets généralisant les schémas à seuils ont été proposés à la fin des années 80 par J. Benaloh et J. Leichter [15], et par M. Ito, A. Saito et T. Nishizeki [77]. Pour des références plus récentes sur les sytèmes de partage de secrets et leurs généralisations on se reportera à [13, 88]. L'idée est de construire un schéma de partage de secret à partir d'un circuit booléen sur F_n. Le schéma qui suit est inspiré de la description des schémas précédents donnée par D. Stinson dans [134].

Soit $\mathcal{F}_5 = \{p_1, p_2, p_3, p_4, p_5\}$ l'ensemble des parties correspondant respectivement à deux patients, un médecin, une infirmière et un serveur. Considérons le secret $a_1 \in \mathbf{F}_q$ correspondant à la clé de chiffrement des données de prescriptions du premier patient. On associe alors a_1 à l'ensemble $\mathcal{F}_5^1 = \{\{p_1, p_5\}, \{p_3, p_5\}, \{p_4, p_5\}\}$, qui correspond au circuit booléen suivant : $(p_1 \wedge p_5) \vee (p_3 \wedge p_5) \vee (p_4 \wedge p_5)$. On obtient le système de partage de secret sur a_1 suivant :

$$\Pi(a_1) = \{s_1, s_2, s_3, s_4, s_5\} = \{\{y_1^1\}, \emptyset, \{y_2^1\}, \{y_3^1\}, \{a_1 - y_1^1, a_1 - y_2^1, a_1 - y_3^1\}\},$$

où chaque token s_i est distribué à p_i, et où $y_1^1, y_2^1, y_3^1 \in \mathbf{F}_q$ sont aléatoires. Considérons maintenant trois autres secrets a_2, a_3, $a_4 \in \mathbf{F}_q$, associés aux ensembles $\mathcal{F}_5^2 = \{\{p_1, p_5\}, \{p_3, p_5\}\}$, $\mathcal{F}_5^3 = \{\{p_2, p_5\}, \{p_3, p_5\}, \{p_4, p_5\}\}$, et $\mathcal{F}_5^4 = \{\{p_2, p_5\}, \{p_3, p_5\}\}$, qui correspondent aux circuits booléens $(p_1 \wedge p_5) \vee (p_3 \wedge p_5)$, $(p_2 \wedge p_5) \vee (p_3 \wedge p_5) \vee (p_4 \wedge p_5)$ et $(p_1 \wedge p_5) \vee (p_3 \wedge p_5)$, associés respectivement à la protection des données de diagnostic du premier patient et à la protection des données de prescription et de diagnostic du second patient. On obtient le système de partage de secret suivant, en adaptant la définition précédente à de multiples secrets, avec $y_1^3 \in \mathbf{F}_q$ aléatoire :

$$\Pi(a_1, a_2, a_3, a_4) = \{\{y_1^1\}, \{y_1^3\}, \{y_2^1\}, \{y_3^1\}, \{a_1 - y_1^1, a_1 - y_2^1, a_1 - y_3^1, a_2 - y_1^1, a_2 - y_2^1, a_3 - y_1^3, a_3 - y_2^1, a_3 - y_3^1, a_4 - y_1^3, a_4 - y_2^1\}.$$

Comparaison. Dans ce deuxième schéma, deux parties ne peuvent plus collaborer entre elles pour obtenir le secret sans l'aide du serveur. Les calculs y sont plus simples que dans le précédent schéma, mais le serveur possède plus de clés.

Prototype. Un prototype a été développé durant la thèse comportant des données supplémentaires telles que des données administratives et un autre type de personne (administratif). Ce prototype gère plusieurs établissements de santé, l'ajout ou la supression de droits d'accès sur certaines données, etc... (figure 2.2).

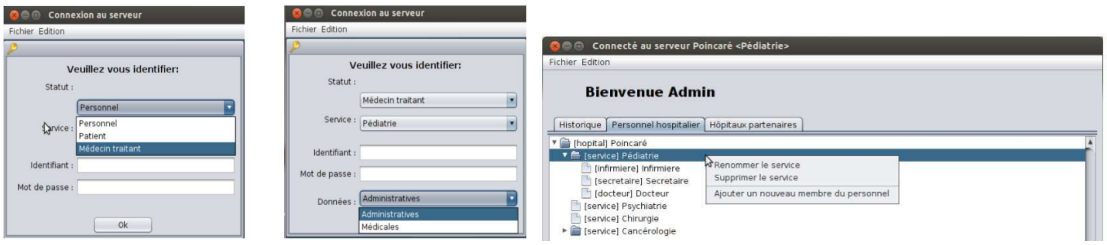


FIGURE 2.2 – Prototype pour la gestion des données de e-santé [114]

Conclusion. Le système proposé permet aux différents acteurs de n'avoir qu'une seule clé, en dehors d'un serveur dédié qui gère efficacement le cas de changement d'acteur dans le système. Cette solution possède bien sûr des limites dans sa mise en oeuvre, mais a le mérite d'apporter une alternative différente. Le passage d'un prototype à un déploiement réel demanderait néanmoins une analyse plus fine des acteurs et de leurs accès aux données, ainsi que du cycle de vie des clés. Curieusement, il y a eu très peu de travaux académique ces dernières années sur la mise en place de nouvelles architectures de e-santé respectueuse de la vie privée.

2.2 Pass de transport anonyme et intraçable

Le déploiement de la technologie NFC s'est largement développé sur smartphone, permettant ainsi le déploiement de nouvelles applications telles que le paiement ou le ticket électronique. Ces services doivent garantir la protection des données personnelles des utilisateurs, ce qui inclue l'anonymat et la non-traçabilité avec une possibilité de lever cet anonymat en cas de fraude. Les smartphones possèdent une grande capacité de calcul, permettant ainsi l'utilisation de primitives cryptographiques dédiées à la protection de la vie privée, comme les signatures de groupe, combiné avec un élément sécurisé (typiquement la carte SIM/UICC) pouvant contenir des applications (cardlet). Pour pouvoir être déployé dans des applications de transport, un tel système doit fonctionner en moins de 300 ms.

Signatures de groupes. Le schéma de signature de groupe proposé dans cette section utilise un couplage e qui est une application bilinéaire de $\mathbb{G}_1 \times \mathbb{G}_2$ vers $\mathbb{G}_T$, où $\mathbb{G}_1$ et $\mathbb{G}_2$ sont deux groupes cycliques additifs d'ordre p, formés de points d'une courbe elliptique, tandis que $\mathbb{G}_T$ est un groupe multiplicatif d'ordre p. Une telle application de couplage doit être efficacement calculable et doit vérifier les deux propriétés suivantes :

$$\forall x_1 \neq 1_{\mathbb{G}_1},\ \forall x_2 \neq 1_{\mathbb{G}_2}\ e(x_1, x_2) \neq 1_{\mathbb{G}_T}$$

$$\forall G_1 \in \mathbb{G}_1,\ \forall G_2 \in \mathbb{G}_2,\ \forall a, b \in \mathbb{Z}_p,\ e([a]G_1, [b]G_2) = e(G_1, G_2)^{ab},$$

où $[a]G_1$ et $[b]G_2$ représentent des multiplications d'un point d'une courbe elliptique par un entier et $1_{\mathbb{G}_1}$, $1_{\mathbb{G}_2}$, $1_{\mathbb{G}_T}$ sont les éléments neutres des groupes. De tels applications existent mais seront considérées comme des boites noires dans cette section. L'utilisation des couplages en cryptographie est décrite dans [54].

Les signatures de groupe ont été introduites par D. Chaum et E. Van Heyst en 1991 [30]. Ces schémas, comme leurs propriétés, ont fortement évolué depuis, mais le principe reste le même : ceux-ci permettent aux membres d'un groupe de signer anonymement (sans pouvoir différentier les membres du groupes), avec une possibilité de révocation d'anonymat. Une variante appelée signature de liste permet en outre de lier deux signatures si celles-ci sont réalisées par la même personnes dans un intervalle de temps donné (S. Canard, B. Schoenmakers, M. Stam et J. Traore [26]). On ne détaille pas plus l'ensemble de ces protocoles d'un point de vue général car ils sont complexes et ont pas mal évolué dans le temps. Notons que certains sont décrits dans le standard ISO 20008-2 sur les signatures anonymes.

Les signatures DAA (*Direct Anonymous Authentication*) sont une autre variante proposée en 2004 par E. Brickell, J. Camenisch et L. Chen [21]. Elles gèrent une certaine délégation de calcul pour l'entité qui signe ce qui est très utile sur un smartphone. Le protocole DAA s'appuie en particulier sur les CL-signatures proposées par J. Camenish et A. Lysyanskaya en 2002 [25]. Remarquons que le protocole DAA d'origine de 2004 (tout comme les CL-signatures de 2002) n'utilisait pas les couplages et les courbes elliptiques et était développé dans le cadre d'un module TPM (*trusted platform module*). Globalement, ce type de signature permet à un utilisateur d'obtenir une signature sur une donnée mise en gage (i.e. sans révéler sa valeur).

Phase d'enregistrement. L'autorité de transport choisit deux fonctions de hachage $\mathcal{H}$ et $\mathcal{H}_1$, la première retournant une empreinte dans $\{0,1\}^k$ et la seconde dans $\mathbb{G}_1$. L'autorité de transport génère aussi aléatoirement $G_1 \in \mathbb{G}_1$ et $G_2 \in \mathbb{G}_2$, ainsi qu'un couple clé publique/privée de signature (pk_t, sk_t) où $sk_t = (x, y) \in \mathbb{Z}_p^2$ et $pk_t = (X, Y) = ([x]G_2, [y]G_2)$. L'autorité d'ouverture génère un couple clé publique/privée de signature (pk_o, sk_o) de manière identique et l'utilisateur génère sa clé privée sk_u en la tirant aléatoirement dans $\mathbb{Z}_p$. Enfin, on note pk_g l'ensemble des paramètres $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, \mathcal{H}, \mathcal{H}_1, pk_t, pk_o)$ qui forment la clé publique du groupe.

Lors de la phase d'enregistrement de l'utilisateur, celui-ci commence par calculer deux engagements C_1 et C_2 sur sa clé privée sk_u et les envoie à l'autorité d'ouverture. Celle-ci vérifie qu'ils sont corrects en contrôlant que $e(C_1, G_2) = e(G_1, C_2)$, gràce à la propriété des couplages. Si c'est le cas, cela signifie que les deux engagements C_1 et C_2 concernent bien la même clé sk_u. Dans ce cas, l'autorité signe C_1 avec sa clé privée sk_o et retourne la signature μ à l'utilisateur. Les données C_1, C_2 et μ sont stockées dans une base de données DB_o, comme présenté dans la table 2.1.

Enregistrement dans la base de donnée DB_o de l'autorité d'ouverture		
Utilisateur (cardlet)		**Autorité d'ouverture**
Entrées publiques : pk_g Entrées privées : sk_u		Entrées publiques : pk_g Entrées privées : sk_o, DB_o
$(C_1, C_2) \leftarrow ([sk_u]G_1, [sk_u]G_2)$	C_1, C_2 $\longrightarrow$	Si $e(C_1, G_2) = e(G_1, C_2)$ Alors :
	μ $\longleftarrow$	$\mu \leftarrow Signature(sk_o, C_1)$ **Stocke** (C_1, C_2, μ) dans DB_o

TABLE 2.1 – Protocole d'enregistrement auprès de l'autorité d'ouverture

Après avoir reçu cette signature, l'utilisateur s'enregistre auprès de l'autorité de transport en lui envoyant μ et C_1. Celui-ci vérifie la signature μ avec pk_o. Si la signature est correcte, l'autorité de transport calcule et retourne à l'utilisateur une une CL-signature de la forme (A, B, C, D). Cette signature est calculée à partir d'un aléa a par le calculs de quatre points : $A = [a]G_1$, $B = [y]A$, $C = [x + a]D$ et $D = [ay]C_1$. A la fin, l'autorité de transport stocke C_1 ainsi que l'identifiant de l'utilisateur dans sa base de données DB_t, comme présenté dans la table 2.2. Notons que les deux bases de données sont utilisées uniquement pour la de-anonymisation des utilisateurs en cas de fraude par exemple.

Phase de validation. Le processus de validation du pass de transport consiste pour l'utilisateur à utiliser son pass auprès de la borne pour la validation de manière anonyme. Des pré-calculs sont effectués, entre le smartphone et la cardlet, permetant de générer une nouvelle CL-signature (R, S, T, W) à partir de la CL-signature (A, B, C, D) obtenue lors de la phase d'enregistrement et d'une donnée aléatoire l par le calcul de quatre points : R= $[l]A$, $S = [l]B$, $T = [l]C$ et $W = [l]D$. En plus de cette CL-signature, la cardlet calcule le point $R_2 = [k]S$ pour un entier aléatoire k. La CL-signature (A, B, C, D) n'est jamais directement utilisée et ne sort pas de l'élément sécurisé (voir table 2.3). Si on a de la place dans la mémoire de l'élément sécurisé, il est possible de précalculer plusieurs CL-signatures.

Enregistrement dans la base de donnée DB_t de l'autorité de transport		
Utilisateur (cardlet)		**Autorité de transport**
Entrées publiques : pk_g, μ Entrées privées : sk_u		Entrées publiques : pk_g Entrées privées : $sk_t = (x, y)$, DB_t
	C_1, μ $\longrightarrow$	Si verify(pk_o, μ) est correct, alors calculer une CL-signature sur C_1 : $a \in \mathbb{Z}_p$ aléatoire $A \leftarrow [a]G_1$ $B \leftarrow [y]A$ $C \leftarrow [x](A + D)$
	A, B, C, D	$D \leftarrow [a.y]C_1$
Stocke A, B, C, D	$\longleftarrow$	**Stocke** (utilisateur, C_1) dans DB_t

TABLE 2.2 – Protocole d'enregistrement auprès de l'autorité de transport

Precalculs : randomization des CL-Certificats		
Utilisateur (cardlet)		**Smartphone**
Entrées publiques : pk_g Entrées privées : sk_u		Entrées publiques : pk_g, (A, B, C, D)
$k \in \mathbb{Z}_p$ aléatoire		$l \in \mathbb{Z}_p$ aléatoire $(R, S) \leftarrow ([l]A, [l]B)$
	R, S, T, W $\longleftarrow$	$(T, W) \leftarrow ([l]C, [l]D)$
$R_2 \leftarrow [k]S$		

TABLE 2.3 – Précalculs entre le smartphone et la cardlet

La validation consiste ensuite en un protocole challenge/réponse où la borne envoie un challenge aléatoire rc à l'utilisateur qui retourne une signature de groupe σ sur ce challenge. Celle-ci est formée de plusieurs composantes (R, S, T, W, J, K, c, s), où $K = [sk_u]J$, $c = \mathcal{H}(J, K, R, S, T, W, R_1, R_2, rc)$, avec $R_1 = [k]J$ et $s = k + csk_u (\text{mod} p)$ et où J est un dispositif anti pass-back expliqué dans le paragraphe suivant. La borne vérifie cette signature en vérifiant plusieurs points :

1. La borne contrôle si $e(R, Y) = e(S, G_2)$. L'égalite vient de la bilinéarité du couplage entre la partie gauche $(e([la]G_1, [y]G_2))$ et celle de droite $(e([ly]A, G_2))$.
2. La borne contrôle si $e(T, G_2) = e(R + W, X)$. En effet, la partie de gauche est égale à $e([alx(1 + ysk_u)]G_1, G_2)$ tandis que la partie de droite est $e([la(1 + ysk_u)]G_1, [x]G_2)$ qui sont égales par la propriété des couplages.
3. La borne retrouve la valeur c, à partir des autres composantes de σ et de $R_1 = [s]J$ - $[c]K$ et $R_2 = [s]S$ - $[c]W$. En effet, $[s]J$ - $[c]K = [k + csk_u]J$ - $[csk_u]J = [k]J$ qui est bien la valeur de R_1 que la cardlet a calculé lors du calcul de la signature. Pour la seconde valeur, on a $[s]S$ - $[c]W = [(k + csk_u)lya]G_1$ - $[claysk_u]G_1 = [klya]G_1$ qui correspond à $R_2 = [k]S$, précalculé par la cardlet.

Validation du pass de transport		
Utilisateur (cardlet)		**Borne**
Entrées publiques : pk_g, R, S, T, W, R_2, k Entrées Privées : sk_u		Entrées Publiques : pk_g, bsn
Calculer la signature de groupe σ : $J \leftarrow \mathcal{H}_1(bsn)$ $R_1 \leftarrow [k]J$ $K \leftarrow [sk_u]J$ $c \leftarrow$ $\mathcal{H}(J, K, R, S, T, W, R_1, R_2, rc)$ $s \leftarrow k + c.sk_u \ modp$	rc, bsn $\longleftarrow$ $\sigma =$ (R, S, T, W, J, K, c, s) $\longrightarrow$	$rc \in \{0,1\}^\gamma$ aléatoire Verifier la signature σ : $J' \leftarrow \mathcal{H}_1(bsn)$ $R_1 \leftarrow [s]J - [c]K$ $R_2 \leftarrow [s]S - [c]W$ $e(R, Y) \stackrel{?}{=} e(S, G_2)$ $e(T, G_2) \stackrel{?}{=} e(R + W, X)$ $c \stackrel{?}{=}$ $\mathcal{H}(J', K, R, S, T, W, R_1, R_2, rc)$ Si l'utilisateur est sur liste noire, alors l'accès est refusé Sinon l'accès est autorisé

TABLE 2.4 – Le protocole de la phase de validation

Dispositif anti pass-back. En plus du challenge rc, la borne envoie aussi une donnée temporelle bsn pour que l'utilisateur ne puisse pas réutiliser son pass avant un certain temps. Pour cela, la cardlet calcule $J = \mathcal{H}_1(bsn)$ et la borne stocke la signature σ, incluant J et K, durant ce laps de temps. Si elle reçoit la donnée K dans une nouvelle signature, la borne pourra en conclure qu'il sagit du même utilisateur. Au dela de ce laps de temps, l'utilisateur ne pourra plus être identifié à l'aide de K, qui varie avec bsn.

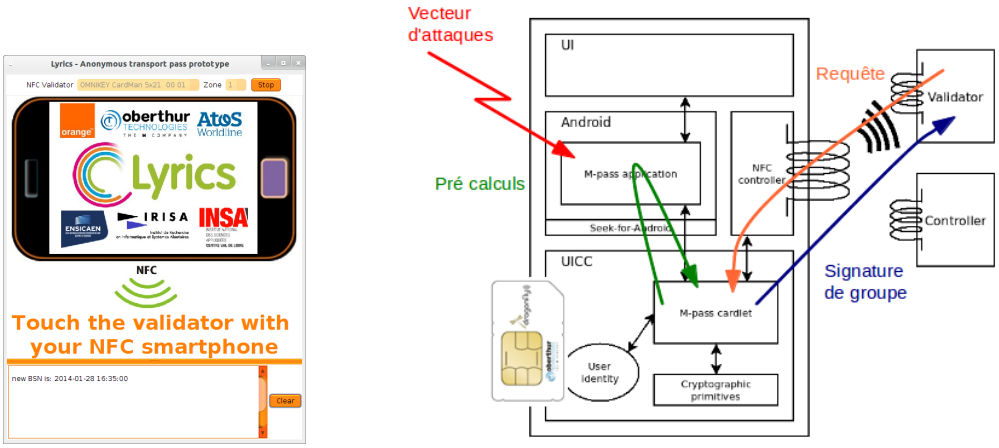


FIGURE 2.3 – Implémentation sur smartphone du pass de transport

De-anonymisation. Le processus de de-anonymisation du pass est réalisé par l'autorité de transport si elle a le consentement de l'autorité d'ouverture et coopère avec elle. Pour cela, l'autorité de transport envoie la signature σ à l'autorité d'ouverture qui cherche dans sa base de donnée l'utilisateur pour lequel on a $e(J, C_2) = e(K, G_2)$ (il y a égalité car les deux engagements sur sk_u sont $K = [sk_u]J$ et $C_2 = [sk_u]G_2$). Dans ce cas, l'autorité d'ouverture retourne C_1 à l'autorité de transport qui retrouve l'identifiant de l'utilisateur. L'autorité de transport n'ayant pas accès à la donnée C_2 ne peut pas de-anonymiser l'utilisateur sans l'aide de l'autorité d'ouverture.

Autorité de transport		**Autorité d'ouverture**
Entrées publiques : pk_g, DB_t, σ Entrées privées : sk_t		Entrées publiques : pk_g Entrées privées : sk_o, DB_o
	σ $\longrightarrow$	Trouver (C_1, C_2, μ) dans DB_o tels que : $e(J, C_2) = e(K, G_2)$
Connaissant C_1, trouver (ID_{user}, C_1) dans DB_t	C_1 $\longleftarrow$	

TABLE 2.5 – Le protocole de-anonymisation

Liste noire. Une procédure de liste noire peut aussi être mise en place par exemple si un utilisateur a perdu son portable. Elle se déroule de la manière suivante : l'autorité de transport retrouve l'engagement C_1 qui correspond à l'utilisateur à mettre en liste noire et envoie cette valeur à l'autorité d'ouverture. Celle-ci retrouve l'engagement C_2 correspondant et calcule pour toutes les valeurs de bsn possibles $e(\mathcal{H}_1(bsn), C_2)$. Ces valeurs sont envoyées à l'autorité de transport. Au moment de la validation d'un pass, la borne devra calculer $e(K, G_2)$ et vérifier que cette valeur n'est pas dans la liste noire retournée par l'autorité d'ouverture. En effet, s'il y a égalité, cela signifie que C_2 et K sont deux engagement sur la clé sk_u, par la propriété des couplage, et donc qu'un utilisateur sur liste noire essaie d'utiliser le pass de transport.

Conclusion. Le pass de transport a été implémenté par Orange sur smartphone Galaxy S3 contenant une carte UICC munie d'un co-processeur arithmétique permettant d'obtenir de bonnes performances en arithmétique modulaire mais aussi au niveau des opérations sur les courbes elliptiques. Le temps de calcul d'une signature par la cardlet était de 140 ms contre seulement 40ms pour la validation. Bien sûr, la cardlet n'a aucun calcul de couplage à faire dans tout le protocole, sinon les temps de calcul n'auraient pas été les mêmes. L'implémentation sur mobile fonctionne en moins de 300ms afin de pouvoir être déployée en pratique.

Suite à ces travaux. Un complément à ce travail a été réalisé dans la thèse de G. Arfaoui, sur un service de billeterie électronique pour transport [5]. Par ailleurs, la procédure de liste noire, dans un protocole de transport basé sur les signatures de groupe, a été développée par Umar *et al.* dans [141].

Chapitre 3

Protection des données biométriques

Sommaire

3.1 Transformations de données biométriques 30
3.2 Transformation biohashing 32
3.3 Transformations GRP-IoM et URP-IoM 36

Dans un système d'authentification biométrique, un chiffrement standard ne permet pas de sécuriser la phase d'authentification car les données, étant floues, doivent être comparées en clair. Des solutions existent, incluant les engagements flous ou le chiffrement homomorphe, mais ces systèmes ont leurs limites et doivent en outre être utilisés sur des données binaires. Une alternative consiste à appliquer une transformation non inversible à ces données (généralement avec une clé), puis à effectuer la phase d'authentification directement sur les données transformées. Certaines de ces transformations binarisent les données sur lesquelles peut s'effectuer en supplément un chiffrement au moment de la transmission et du stockage. Ces transformations ne doivent naturellement pas trop réduire la performance du système, en terme de taux de fausse acceptation et de faux rejet.

Dans ce chapitre, on s'intéresse la sécurité de ces transformations, qui pose notamment des problèmes si la clé est connue (scénario souvent considéré en authentification car celle-ci doit être stockée quelque part). Dans un premier temps, on évalue la sécurité d'une transformation appelée hachage biométrique ou biohashing. Cette étude a donné lieu à deux publications en 2013 [93, 94] où la seconde est une collaboration avec Estelle Cherrier et Christophe Rosenberger. Dans une dernière section, on s'intéresse à la sécurité de deux autres transformations récentes, dans le cadre d'une collaboration avec Kevin Atighehchi, Loubna Ghammam et Koray Karabina qui est en cours de publication [9].

3.1 Transformations de données biométriques

Toutes les définitions de cette partie sont adaptées de [9]. Elles sont utilisées dans la suite du chapitre pour définir les attaques sur différentes transformations. Des approches plus anciennes de formalisation d'attaques ont été proposées en 2010 puis 2013 dans [104, 76], mais celles-ci n'étaient pas suffisantes pour décrire les attaques de ce chapitre. Nous avons aussi formalisé les algorithmes sur les transformations.

Définition 3. *Soit $(\mathcal{M}_A, D_A)$ l'espace métrique des vecteurs de caractéristiques de données biométriques, associé à la distance D_A. On définit un système de vérification biométrique par la paire d'algorithmes $\Pi := (E, V)$ tels que :*

1. *E est un algorithme d'extraction qui prend en entrée une donnée biométrique b et retourne un vecteur de caractéristiques $x \in \mathcal{M}_A$.*
2. *V est un algorithme de vérification qui prend en entrées deux vecteurs de caractéristiques $x = E(b)$, $x' = E(b')$ et un seuil τ_A, et qui retourne $V(x, x', \tau_A) =$ Vrai si $D_A(x, x') \leq \tau_A$ et Faux si $D_A(x, x') > \tau_A$.*

Définition 4. *Soit $(\mathcal{M}_B, D_B)$ un espace métrique et $\mathcal{K}$ l'espace des clés. On définit un système de vérification biométrique avec transformation par le triplet d'agorithmes $\Xi := (\mathcal{G}, \mathcal{T}, \mathcal{V})$ tels que :*

1. *$\mathcal{G}$ est un algorithme de génération qui prend en entrée une clé $s \in \mathcal{K}$ et retourne un ensemble de paramètres sp secrets.*
2. *$\mathcal{T}$ est un algorithme de transformation qui prend en entrée un vecteur de caractéristiques $x \in \mathcal{M}_A$ ainsi que les paramètres sp, et retourne le template biométrique $\mathcal{T}(sp, x) \in \mathcal{M}_B$.*
3. *$\mathcal{V}$ est un algorithme de vérification qui prend comme entrées deux templates biométriques $u = \mathcal{T}(sp, x)$, $u' = \mathcal{T}(sp', x')$ ainsi qu'un seuil τ_B, et qui retourne $\mathcal{V}(u, u', \tau_B) \in \{Vrai, Faux\}$, selon si la distance $D_B(u, u') \leq \tau_B$ ou non.*

Typiquement, on a $\mathcal{M}_A \subset \mathbb{R}^n$ et D_A la distance euclidienne. Dans cette partie, on a $\mathcal{M}_B = \{0,1\}^m$, dans le cas du hachage biométrique, ou $(\mathbb{Z}_k)^m$, dans le cas des deux dernières transformations et D_B la distance de Hamming. Dans ce manuscrit, on appelle vecteur (de caractéristiques) une donnée avant transformation et template (biométrique) une donnée après transformation, afin de faciliter la lecture.

Performance d'un système biométrique. On note FAR_A, FRR_A, EER_A, FAR_B, FRR_B, EER_B les taux de fausse acceptation, de faux rejets et d'erreur égale du système biométrique, avant et après transformation. Selon le contexte, ceux-ci seront vus comme des pourcentages ou des probabilités et dépendent des seuils τ_A et τ_B.

On utilise la notation $x \leftarrow\!\$\, \mathcal{M}$ pour indiquer que x est choisi uniformément à partir d'un ensemble $\mathcal{M}$. Soit $\mathcal{U}$ l'ensemble des utilisateurs d'un système et $\mathcal{BC}(.)$ un mécanisme prenant en entrée $usr \in \mathcal{U}$ et retournant une donnée biométrique b. On note $\mathcal{BC}[\mathcal{U}]$ l'ensemble des données biométriques pouvant être obtenues de $\mathcal{U}$. Pour $usr \in \mathcal{U}$, $\mathcal{BC}^0[usr]$ est l'ensemble des données biométriques venant de usr, et $\mathcal{BC}^1[usr] = \mathcal{BC}[\mathcal{U} \setminus usr]$. On utilisera la notation $b \leftarrow\!\$\, \mathcal{BC}^c[usr]$, avec $c \in \{0,1\}$.

La formalisation précédente est utile pour décrire la notion d'utilisateur légitime et illégitime. En effet, on remarque que $\mathcal{BC}^0[usr]$ et $\mathcal{BC}^1[usr]$ ne sont pas forcément des ensembles disjoints à cause des taux d'erreurs ci-dessus. Ainsi, deux vecteurs x et x' peuvent venir de la même personne et vérifier $D_A(x, x') \geq \tau_A$ ou venir de personnes différentes et vérifier $D_A(x, x') \leq \tau_A$, ce qui complique la formalisation de critères de sécurité comme la notion d'indistinguabilité.

Attaques sur une tranformation biométrique. On formalise maintenant les attaques possibles, dans le cas où un attaquant a accès à la clé s, et donc aux paramètres secrets sp (mais pas au vecteur de caractéristique x) :

Définition 5. *Une attaque par inversion sur la paire $(x, u) \in \mathcal{M}_A \times \mathcal{M}_B$ est un algorithme $\mathcal{A}_1$ qui prend en entrée sp et $u = \Xi.\mathcal{T}(sp, x)$ et qui retourne $\mathcal{A}_1(sp, u) = x^* \in \mathcal{M}_A$ tel que $\Pi.V(x, x^*, \tau_A) = Vrai$. Le taux de succès est :*

$$\mathrm{Rate}_{\mathcal{A}_1}^{Rev_1} = \Pr\left[\Pi.V(x, x^*, \tau_A) = Vrai \;\middle|\; \begin{array}{l} b \leftarrow\!\$\, \mathcal{BC}[\mathcal{U}];\ x \leftarrow \Pi.E(b); \\ s \leftarrow\!\$\, \mathcal{K};\ sp \leftarrow \Xi.\mathcal{G}(s)\ ; \\ u \leftarrow \Xi.\mathcal{T}(sp, x);\ x^* \leftarrow \mathcal{A}_1(sp, u); \end{array}\right].$$

Cette attaque est considérée comme réussie si $\mathrm{Rate}_{\mathcal{A}_1}^{Rev_1} > \mathrm{EER}_A$.

Définition 6. *Une attaque par authentification sur $(x, u) \in \mathcal{M}_A \times \mathcal{M}_B$ est un algorithme $\mathcal{A}_2$ qui prend en entrée sp et $u = \Xi.\mathcal{T}(sp, x)$ et qui retourne $\mathcal{A}_2(sp, u) = x^* \in \mathcal{M}_A$ tel que $u^* = \Xi.\mathcal{T}(sp, x^*)$ et $\Xi.\mathcal{V}(u, u^*, \tau_B) = Vrai$. Le taux de succès est :*

$$\mathrm{Rate}_{\mathcal{A}_2}^{Rev_2} = \Pr\left[\Xi.\mathcal{V}(u, u^*, \tau_B) = Vrai \;\middle|\; \begin{array}{l} b \leftarrow\!\$\, \mathcal{BC}[\mathcal{U}];\ x \leftarrow \Pi.E(b); \\ s \leftarrow\!\$\, \mathcal{K};\ sp \leftarrow \Xi.\mathcal{G}(s)\ ; \\ u \leftarrow \Xi.\mathcal{T}(sp, x);\ x^* \leftarrow \mathcal{A}_2(sp, u); \\ u^* \leftarrow \Xi.\mathcal{T}(sp, x^*); \end{array}\right].$$

Cette attaque est considérée comme réussie si $\mathrm{Rate}_{\mathcal{A}_2}^{Rev_2} > \mathrm{EER}_B$.

En d'autres termes, une attaque par inversion sur (x, u) doit retourner un vecteur de caractéristique proche de x, tandis qu'une attaque par authentification sur (x, u) doit retourner un vecteur de caractéristique dont l'image soit proche du template u. Remarquons que les taux de succès ne sont pas calculés pour (x, u) fixé.

Définition 7. *Soit $x^* = \mathcal{A}_2(sp, \Xi.\mathcal{T}(sp, x))$ une préimage sur $x \in \mathcal{M}_A$. Une attaque par préimage réutilisable sur la paire $(x', u') \in \mathcal{M}_A \times \mathcal{M}_B$ est un algorithme $\mathcal{A}_3$ qui prend en entrée x^*, sp', $u' = \Xi.\mathcal{T}(sp', x')$, où x et x' viennent du même utilisateur, qui calcule $u^* = \Xi.\mathcal{T}(sp', x^*)$ et qui retourne $\mathcal{A}_3 \in \{Vrai, Faux\}$ selon si $\Xi.\mathcal{V}(u', u^*, \tau_B) = Vrai$ ou $Faux$. Le taux de succès de $\mathcal{A}_3$ est :*

$$\mathrm{Rate}_{\mathcal{A}_3}^{Rev_3} = \Pr\left[\Xi.\mathcal{V}(u', u^*, \tau_B) = Vrai \;\middle|\; \begin{array}{l} b \leftarrow\!\$\, \mathcal{BC}^0[usr];\ x' \leftarrow \Pi.E(b); \\ s' \leftarrow\!\$\, \mathcal{K};\ sp' \leftarrow \Xi.\mathcal{G}(s')\ ; \\ u^* \leftarrow \Xi.\mathcal{T}(sp', x^*);\ u' \leftarrow \Xi.\mathcal{T}(sp', x')\ ; \end{array}\right].$$

Cette attaque est considérée comme réussie si $\mathrm{Rate}_{\mathcal{A}_3}^{Rev_3} > \mathrm{EER}_B$.

En d'autres termes, une attaque par préimage réutilisable est une attaque par authentification quand une préimage trouvée peut être rejouée sur une autre acquisition du même utilisateur, dont la transformation utilise une autre clé secrète.

On remarque que les taux de réussite de ces attaques dépendent des seuils τ_A et τ_B utilisés par le système et des taux EER_A et EER_B. On a donc clairement un compromis entre taux de succès des attaques et performance du système.

Définition 8. *Une attaque par associativité sur $x, x' \in \mathcal{M}_A$ est un algorithme $\mathcal{A}_4$ qui prend en entrée sp, sp', $u = \mathcal{T}(sp, x)$ et $u' = \mathcal{T}(sp', x')$, et qui retourne $\mathcal{A}_4 \in \{0, 1\}$, selon si x et x' proviennent du même individu ou non. Le taux de succès de $\mathcal{A}_4$ est :*

$$\mathrm{Adv}_{\mathcal{A}_4}^{Link} = \Pr\left[c' = c \;\middle|\; \begin{array}{l} usr \leftarrow\$\, \mathcal{U};\ b \leftarrow\$\, \mathcal{BC}^0[usr];\ x \leftarrow \Pi.E(b); \\ s \leftarrow\$\, \mathcal{K}; sp \leftarrow \Xi.\mathcal{G}(s);\ s' \leftarrow\$\, \mathcal{K}; sp' \leftarrow \Xi.\mathcal{G}(s')\ ; \\ c \leftarrow\$\, \{0,1\};\ b' \leftarrow\$\, \mathcal{BC}^c[usr];\ x' \leftarrow \Pi.E(b'); \\ u \leftarrow \Xi.\mathcal{T}(sp, x); u' \leftarrow \Xi.\mathcal{T}(sp', x'); \\ c' \leftarrow \mathcal{A}_4(sp, u, sp', u'); \end{array} \right].$$

Cette attaque est considérée comme réussie si $\mathrm{Adv}_{\mathcal{A}_4}^{Link} > 0.5$.

Une attaque par associativité consiste donc à décider si deux templates biométriques proviennent d'une même personne ou non. Notons l'ambiguïté de cette définition dans le cas de la biométrie. En effet, les deux templates peuvent provenir de deux données biométriques provenant du même individu, mais trop différentes pour que le système de vérification les accepte. Le problème est le même avec deux données biométriques proches mais provenant d'individus différents. L'attaque par associativité qui sera développée dans la suite du chapitre essaira de distinguer deux individus, indépendamment de la qualité des données d'acquisition (pire cas).

3.2 Transformation biohashing

Le hachage biométrique (*biohashing*) a été introduit en 2004 par A. J. B. Teoh, M. Goh et D. Ngo [34], sur plusieurs modalités biométriques. Cette transformation est intéressante par sa simplicité et par le fait que la seule contrainte sur la modalité biométrique est que celle-ci puisse être représentée par des caractéristiques réelles et de taille fixe. Cette transformation prend un vecteur de n caractéristiques réelles x et retourne un template binaire u de m bits avec $m \leq n$.

Génération des paramètres. Algorithme $\mathcal{G}$ pour la génération de paramètres qui prend en entrée la clé secrète s, la taille n du vecteur de caractéristiques et la taille m du template, pour sortir une famille sp de vecteurs orthonormés :

1. Pour $i = 1, \ldots, m$, génèrer m vecteurs pseudo-aléatoires v_i de longueur n à partir de la clé secrète s.
2. Utiliser l'algorithme de Gram-Schmidt sur les m vecteurs v_i pour obtenir une famille orthonormée sp de m vecteurs $w_1, \ldots, w_m$.

Transformation de la donnée. Algorithme de $\mathcal{T}$ pour la transformation qui prend en entrée sp, un vecteur x de caractéristiques dans $\mathbb{R}^n$, ainsi qu'un seuil τ (typiquement $\tau = 0$) et qui retourne en sortie un template biométrique dans $\{0,1\}^m$:

1. Pour $i = 1, \ldots, n$, calculer les m produits scalaires $p_i = < x, w_i >$.
2. Retourner $u = (u_1, \ldots, u_m) \in \mathcal{M}_B \subset \{0,1\}^m$ à l'aide du processus de quantification :

$$u_i = \begin{cases} 0 & \text{if} \quad p_i < \tau \\ 1 & \text{if} \quad p_i \geq \tau \end{cases}$$

A l'origine, les auteurs ont présenté ce système de hachage biométrique comme ayant un $\text{EER}_B = 0$. Néanmoins, ce taux était calculé sans la connaissance de la donnée secrète servant à générer la famille de vecteurs orthonormée. Deux ans plus tard, l'étude de la sécurité de ce système a été affinée par A. B. J. Teoh, T. Connie, D. Ngo et C. Ling dans [138]. En 2010, une première étude sur l'inversibilité de cette transformation était proposée par A. Nagar, K. Nandakumar et A. K. Jain dans [104], à l'aide de programmation linéaire, qui sera améliorée dans la suite de ce chapitre sur deux autres transformations.

Performance sur une base d'empreintes digitales. La figure 3.1 illustre la perte des performances avec la transformation ($n = 512$, $m = 128$) sur la base d'empreintes digitales FVC 2002, qui contient 800 empreintes (8 empreintes pour chacun des 100 individus), décrite dans [98]. On considère que pour chaque individu, la première empreinte est l'empreinte d'enrollement et les sept autres sont des empreintes de vérification. L'algorithme d'extraction qui retourne un vecteur de caractéristiques à partir d'une donnée biométrique, utilise des filtres de Gabor et n'est pas décrit. On peut se reporter à la thèse de R. Belguechi pour l'extraction de telles caractéristiques [14]. On obtient respectivement $\text{EER}_A \simeq 0.1$ et $\text{EER}_B \simeq 0.16$ avec la distance euclidienne et celle de Hamming. Dans les deux cas, les personnes illégitimes utilisées pour calculer l'EER viennent de la base FVC 2002. Pour chaque personnes, on compare la première empreinte avec les 8 empreintes des 99 autres.

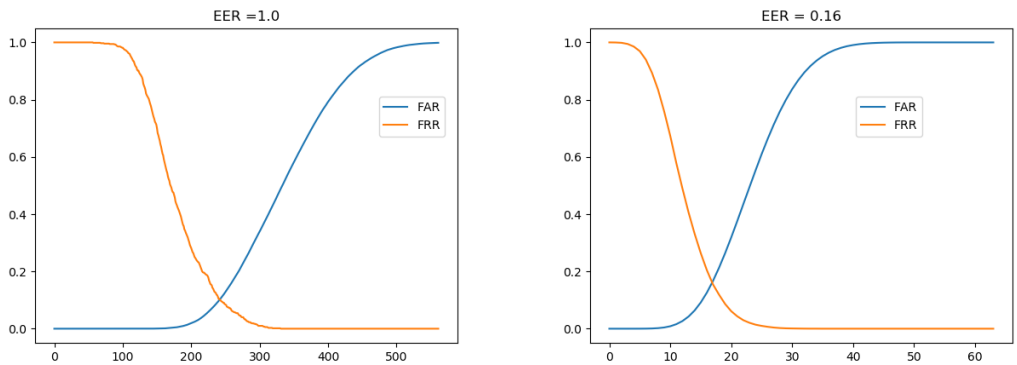


FIGURE 3.1 – Calcul d'EER sans et avec hachage sur la base FVC 2002

Performance avec un secret partiellement connu. Dans cette partie, on va regarder en détails les performances d'une transformation dans le cas où la clé secrète s est plus ou moins connue d'un attaquant. On note FAR_{Bk} le taux de fausse acceptation du système quand l'attaquant connait la clé et FAR_{Bu} le taux de fausse acceptation quand l'attaquant ne connait pas la clé. On note EER_{Bk} et EER_{Bu} les taux correspondants calculés avec le taux de faux rejet FRR_B. On commence par étudier ces taux sur un exemple trivial $u = \mathcal{T}(s, x) = s \in \{0,1\}^m$. On a bien sûr $\mathrm{FAR}_{Bk} = 100\%$ et $\mathrm{FRR_B} = 0\%$. Dans le cas où la donnée s est inconnue de l'attaquant, on obtient

$$\mathrm{FAR}_{Bu} = \frac{1}{2^m} \sum_{i=0}^{\tau_B} \binom{m}{i}. \tag{3.1}$$

Cela donne un ERR_{Bu} de 50% et un ERR_{Bu} de 0%.

L'objectif est de déterminer comment les performances d'une transformation se comportent quand certains bits de la clé sont connus. Plus précisément, on suppose que t bits de la clé sont inconnus de l'attaquant. On note O_t l'évènement signifiant que l'attaquant a généré les t bits restants correctement, N_t celui signifiant que l'attaquant n'a pas généré les t bits correctement et A celui signifiant que l'attaquant se soit authentifié avec succès. On obtient alors les probabilité conditionelles :

$$P(A|N_t) = P(A|\text{ s est inconnue}) \text{ et } P(A|O_t) = P(A|\text{ s est connue}).$$

Et par definition,

$$\begin{aligned} P(A) &= P(A \mid O_t)P(O_t) + P(A \mid N_t)P(N_t) \\ &= \frac{1}{2^t}P(A \mid O_t) + \frac{2^t - 1}{2^t}P(A \mid N_t). \end{aligned}$$

Ainsi, pour tout $t \geq 1$, le $\mathrm{FAR_B}$ est lié à FAR_{Bk} et FAR_{Bu}, par le théorème suivant :

Théorème 6. *Soit f une transformation dont le taux de fausse acceptation est FAR_{Bk} si le secret est connu et FAR_{Bu} si le secret est inconnu. Alors le taux de fausse acceptation quand t bits du secret sont inconnus est*

$$\mathrm{FAR}_B = \frac{1}{2^t}\mathrm{FAR}_{Bk} + \frac{2^t - 1}{2^t}\mathrm{FAR}_{Bu}.$$

Tests sur une base d'empreinte digitale. L'objectif est d'étudier si la transformation de biohashing se comporte conformément au théorème précédent. On a réalisé l'expérience sur la base FVC 2002 avec les mêmes paramètres ($n = 512$, $m = 128$) que précédemment, en donnant à l'attaquant la pleine connaissance du secret s à l'exception de t bits, avec t variant entre 0 et 6. Les performances sont décrites sur la figure 3.2. On y voit qu'avec 5 bits inconnus, l'EER est sous les 1% comme prévu. On remarque que l'EER est légèrement différent de celui de la figure 3.1, du au caractère aléatoire de la génération des secrets. Les performances de la transformation biohashing se comportent comme prévu par le théorème 6.

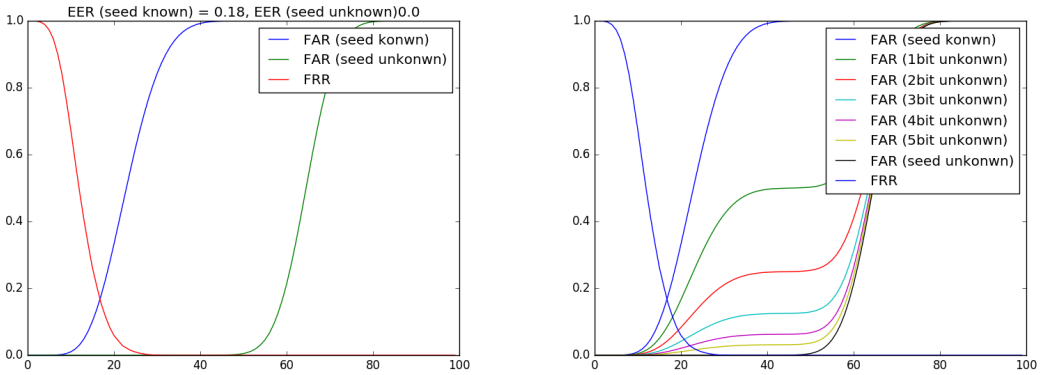


FIGURE 3.2 – FAR avec t bits inconnus dans le secret

Préimage réutilisable par un algorithme génétique. Dans cette section, on va décrire une attaque pour retrouver une préimage réutilisable sur la transformation précédente. L'approche présentée ici utilise un algorithme génétique (classe d'algorithmes initialement proposée par J. H. Holland en début des années 1970 [71]).

On suppose que l'on possède $u = \mathcal{T}(sp, x)$ et le secret sp, et $u' = \mathcal{T}(sp', x')$ et le secret sp' avec x et x' provenant du même individu. On cherche une seconde préimage réutilisable x^*, c'est-à-dire qui vérifie $D_B(u, \mathcal{T}(sp, x^*))) \leq \tau_B$ et qui minimise la fonction d'évaluation $F(x^*) = D_B(u', \mathcal{T}(sp', x^*))$. On considère un candidat x^* comme un vecteur de n valeurs réelles, provenant d'une population initiale de N candidats possible. Cette population est modifiée à chaque itération, pendant un nombre fixé d'itération, ou lorsqu'elle vérifie une condition d'arrêt. Plus précisément, à chaque itération, on réalise les opérations suivantes sur les candidats :

1. Selection : on ne conserve qu'un pourcentage des candidats dans la population.
2. Croisement : les candidats restants génèrent de nouveaux candidats.
3. Mutation : chaque caractéristique des nouveaux candidats a une probabilité de muter (de manière fixe ou adaptative).

Implémentation de l'attaque. Les expériences sont réalisées sur la même base d'empreintes digitales FVC 2002 que précédemment avec un seuil τ_B de 17 (voir figure 3.1) et les mêmes paramètres pour la transformation ($n = 512$, $m = 128$). Pour la sélection, on choisit de conserver 50% de la population des candidats x^* qui vérifient la contrainte $D_B(u, \mathcal{T}(sp, x^*))) \leq \tau_B$ et qui minimisent la fonction d'évaluation $F(x^*) = D_B(u', \mathcal{T}(sp', x^*))$. Pour le croisement, chaque paire (x_1^*, x_2^*) produit deux nouveaux candidats (x_3^*, x_4^*), pour maintenir la taille de la population constante. Plus précisément, on utilise $x_3^* = ax_1^* + (1-a)x_2^*$, $x_4^* = (1-a)x_1^* + ax_2^*$ avec a un réel entre 0 et 1. Pour la mutation, on obtient les meilleurs résultats avec une probabilité importante de muter légèrement pour chaque caractéristique.
On voit figure 3.3 que l'algorithme se stabilise avec $F(x^*) = 7$, ce qui est largement inférieur au seuil d'acceptation τ_B du système (et un peu meilleur que dans la publication d'origine [94], avec une fonction de mutation différente).

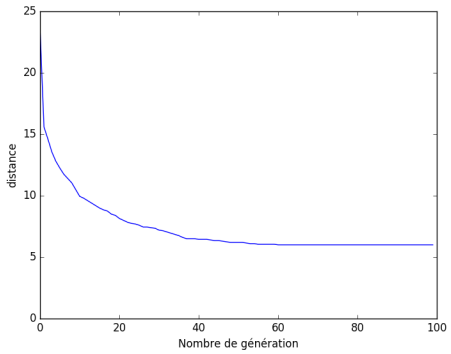


FIGURE 3.3 – Distance obtenue en fonction du nombre d'itérations

Algorithmes génétiques et transformations biométriques. Une approche similaire a été proposée en 2013 sur l'iris par Galbally *et al.* dans [53], la même année que la publication de cette attaque. D'autres attaques utilisant des algorithmes génétiques ont été proposées depuis celle-ci, par exemple par A. Rozsa, A. E. Glock et T. E. Boult en 2015 [123] et par X. Dong, Z. Jin et A. B. J. Teoh en 2019 [44]. Remarquons aussi que ce type d'approche avait déjà été étudié en dehors du contexte de la biométrie. Par exemple, M. Mitchell, J. H. Holland et S. Forrest comparaient, dès les années 90, les performances entre les algorithmes génétiques et les méthodes d'optimisation Hill Climbing [101]. De manière générale, on pourra se référencer vers le livre de M. Mitchell [100] pour une vision globale des algorithmes génétiques.

3.3 Transformations GRP-IoM et URP-IoM

On décrit ici deux transformations récentes appelées GRP-IoM et URP-IoM, présentées en 2018 par Z. Jin, J. Y. Hwang, S. Kim Y.-L. Lai et A. B. J. Teoh dans [78], où l'acronyme IoM signifie Index-of-Max. Si v est un vecteur, $IoM(v)$ correspond à l'indice de la composante maximale de v (si v contient plusieurs composantes maximales, $IoM(v)$ retourne le premier indice). GRP signifie Gaussian Random Projection et URP signifie Uniformly Random Permutation.

L'algorithme d'extraction E, qui prend en entrée une donnée biométrique et retourne un vecteur de caractéristiques utilisés par les auteurs, est décrite dans [79]. La distance D_A (qui en fait n'en est pas une) est calculée par $D_A(x, x') = \sum_{i=1}^{n} x_i x'_i \; / \; \sum_{i=1}^{n} x_i^2 + x_i'^2$, où x et x' sont considérés comme venant de la même personne si $D_A(x, x')$ est supérieure à un seuil donné.

Génération de paramères pour GRP-IoM. L'algorithme $\mathcal{G}$ pour la génération de paramètres, prend en entrée la clé secrète s, la taille n des vecteurs de caractéristique et la taille m des templates biométriques et un paramètre k :

1. Pour $i = 1, \ldots, m$, on génère une matrice Gaussienne $\mathcal{W}^i = [w_1^i \cdots w_k^i]$ où chaque colonne w_j^i est un vecteur de taille n, généré par $w_j^i \leftarrow_{\$} \mathcal{N}(0, I_n)$.
2. Retourner les paramètres sp formés par ces m matrices.

Tranformation des données pour GRP-IoM. L'algorithme $\mathcal{T}$ prend en entrée sp, $x \in \mathcal{M}_A \subset \mathbb{R}^n$, et un paramètre k et retourne $u \in \mathcal{M}_B \subset \mathbb{Z}_k^m$:

1. Pour $i = 1, \ldots, m$:
   a) $v_i \leftarrow x\mathcal{W}^i$.
   b) $u_i \leftarrow IoM(v_i)$
2. Retourner le template $u = (u_1, \ldots, u_m) \in (\mathbb{Z}_k)^m$.

Vérification pour GRP-IoM. L'algorithme de vérification $\mathcal{V}$ prend en entrées $u' \in \mathcal{M}_B \subset \mathbb{Z}_k^m$ et $u \in \mathcal{M}_B \subset \mathbb{Z}_k^m$ ainsi qu'un seuil τ_B, et retourne $True$ ou $False$:

1. $d \leftarrow D_{\mathcal{H}}(u', u)$, où $D_{\mathcal{H}}(,)$ est la distance de Hamming.
2. Si $d \leq \tau_B$ retourner $Vrai$, sinon retourner $Faux$.

Génération de paramères pour URP-IoM. L'algorithme $\mathcal{G}$ pour la génération de paramètres, prend en entrée la clé secrète s, la taille n des vecteurs de caractéristique et la taille m des templates biométriques et les paramètres k et p :

1. Soit S_n le groupe des permutations et $S_{n,k}$ le sous ensemble de S_n où les permutations ne sont évaluées que sur les k premières composantes (avec $k \leq n$). On note $\mathcal{S}_{n,k}$ l'ensemble des représentations matricielles de $S_{n,k}$, dont les $n - k$ dernières colonnes sont donc nulles.
2. Pour $i = 1, \ldots, m$ on génère p permutations $\mathcal{P}_1^i \leftarrow\!\!\$\, \mathcal{S}_{n,k}$, $\ldots$, $\mathcal{P}_p^i \leftarrow\!\!\$\, \mathcal{S}_{n,k}$.
3. Retourner l'ensemble sp composé par ces $m \times p$ permutations.

Tranformation des données pour URP-IoM. L'algorithme $\mathcal{T}$ prend en entrée sp, $x \in \mathcal{M}_A \subset \mathbb{R}^n$, deux paramètres k et p et retourne $u \in \mathcal{M}_B \subset \mathbb{Z}_k^m$:

1. Pour $i = 1, \ldots, m$:
   a) Pour $j = 1, \ldots, p$: $v_j \leftarrow \mathcal{P}_j^i(x)$
   b) $v \leftarrow v_1 \odot \ldots \odot v_p$ (produit de Hadamard d'ordre p)
   c) $u_i \leftarrow IoM(v)$.
2. Retourner le template $u = (u_1, \ldots, u_m) \in (\mathbb{Z}_k)^m$.

L'algorithme de vérification $\mathcal{V}$ pour URP-IoM est le même que pour GRP-IoM.

Performance des transformations précédentes. Les expériences sont réalisées directement sur la base de vecteurs de caractéristiques fournie par les auteurs (qui provient d'une partie de la base FVC 2002, après extraction). Elle est constituée de 5 données biométriques pour 100 personnes, soient 500 vecteurs de $\mathbb{R}^n$ avec $n = 299$. La figure 3.4 donne un EER_A de 3% (avec $\tau_A \simeq 0.3$) avec la distance euclidienne et proche de 0% (avec $\tau_A \simeq 0.13$) avec la distance de [79], en prenant le premier vecteur comme donnée d'enrollement et les quatres autres pour la vérification.

Après transformation, l'EER_B obtenu dépend bien sûr des paramètres choisis. Dans le cas de GRP-IoM, en prenant $m = 300$, $k = 16$, on obtient un EER_B de 0.2% (avec $\tau_B = 0.89m$). Dans le cas URP-IoM, en prenant $k = 128$, $p = 2$ and $m = 600$, on obtient un EER_B de 0.4% (avec $\tau_B = 0.89m$). On remarque que les

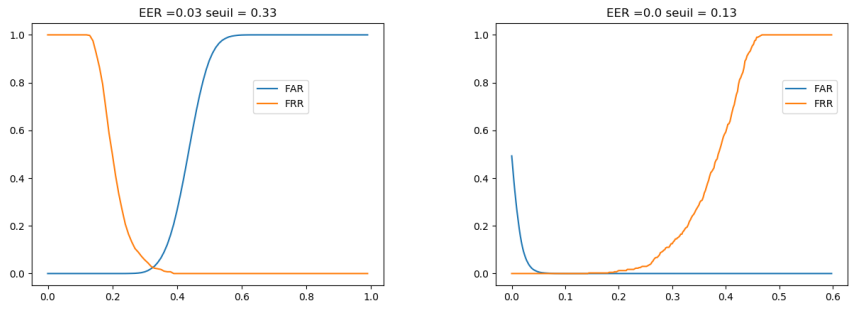


FIGURE 3.4 – EER de la base sans transformation

performances ont été améliorées par une fonction déterministe dans le cas où la distance euclidienne est utilisée avant transformation.

Attaques sur GRP-IoM. Les attaques décrites ci-dessous ont encore comme hypothèse que la clé est connue de l'attaquant. On commence par une attaque par authentification et par préimage réutilisable. On note $A = \{\mathcal{W}_i \in \mathbb{R}^{k\times n}, 1 \leq i \leq m\}$, l'ensemble des matrices gaussiennes qui sont connues de l'attaquant par hypothèse. Pour chaque matrice $\mathcal{W}_i$, on note $\mathcal{W}_{i,j}$ la j-eme ligne avec $j = 1, \ldots, k$. Le template biométrique $u = (u_1, u_2, \ldots, u_m) \in (\mathbb{Z}_k)^m$ est calculé et connu de l'attaquant pour $i = 1, \ldots, m$ par $u_i = \text{argmax}_{l \text{ avec } 1\leq l\leq k}\langle \mathcal{W}_{i,l}, x\rangle$. Celui-ci en déduit un ensemble de $(k-1)m$ inégalités linéaires à n inconnues, de la forme :

$$\{\langle \mathcal{W}_{i,j}, x'\rangle \leq \langle \mathcal{W}_{i,u_i}, x'\rangle\}_{1\leq i\leq m, 1\leq j\leq k}.$$

Ce système est sur-déterminé pour les paramètres proposés et un attaquant peut déterminer une solution x' du système.

Mise en oeuvre. L'attaque a été implémentée en python avec la fonction `linprog` de la librairie `Scipy`. On conserve les paramètres précédents ($m = 300$, $k = 16$). La probabilité de succès d'une attaque par authentification avec la méthode ci-dessus est de 100%. Dans le cas d'une attaque par préimage réutilisable, la distance moyenne obtenue est de $0.55m$ en moyenne et de $0.83m$ dans le pire des cas, soit encore un taux de succès de 100% (on a $\tau_B = 0.89m$).

N	1	2	3	4
Matching Score – pire cas (%)	17	27	33	38
Matching Score – cas moyen (%)	45	50	53	56

TABLE 3.1 – Attaques par préimage réutilisable sur GRP-IoM

Si on se place dans le cas où un attaquant possède N templates biométriques, avec $2 \leq N \leq 4$, on améliore les distances obtenues, surtout dans le pire cas, comme présenté ci dessus dans la table 3.1. Remarque : cette table donne le nombre de coordonnées identiques (*matching score*) en pourcentage.

Attaque par inversion. L'attaque précédente peut aussi servir d'attaque par inversion. Si on l'utilise directement, on obtient une probabilité de succès de 2%, soit

moins que $\mathrm{EER_A}$ dans le cas de la distance euclidienne, ce qui n'est pas suffisant par rapport à la définition 5. On peut par contre utiliser le solveur linéaire `solvers.lp` de la librairie `cvxopt` et rajouter des fonctions objectifs. On réalise les tests avec trois fonctions différentes : $\min \|x\|_2^2$, $\min \|x - v_m\|_2^2$ (où v_m est un vecteur composé par la moyenne de chaque caractéristique de la base), $\min \|x - v_r\|_2^2$ (où v_r est un vecteur de l'attaquant tiré aléatoirement). Les résultats de la table 3.1 précédents ne sont pas améliorés avec l'utilisation de ces fonctions, par contre l'attaque par inversion est considérablement renforcée, avec un taux de succès respectivement de 63%, 74%, 98% (pourcentage des vecteurs reconstruits dont la distance euclidienne avec le vecteur cible est inférieure à $\tau_A = 0.3m$). Le cas de la distance de [79] n'est pas décrit ici : les résultats sont différents, mais l'attaque par inversion fonctionne encore (la dernière fonction objectif est particulièrement bonne).

Attaques sur URP-IoM. On commence par présenter les attaques par authentification et par préimage réutilisable. On note $A = \{\sigma_{1,i}, \ldots, \sigma_{p,i} \in (S_{n,k})^p, 1 \leq i \leq m\}$, l'ensemble des permutations qui sont connues de l'attaquant par hypothèse. Le template biométrique $u = (u_1, u_2, \ldots, u_m) \in (\mathbb{Z}_k)^m$ avec

$$u_i = \underset{l \text{ avec } 1 \leq l \leq k}{\operatorname{argmax}} \{x_{\sigma_{1,i}(l)} \cdot \ldots \cdot x_{\sigma_{p,i}(l)}\} \text{ for } i = 1 \ldots m,$$

où k et p sont connus de l'attaquant. Celui-ci en déduit l'ensemble d'inégalités (de degré p) suivant

$$\left\{ x_{\sigma_{1,i}(j)} \cdot \ldots \cdot x_{\sigma_{p,i}(j)} \leq x_{\sigma_{1,i}(u_i)} \cdot \ldots \cdot x_{\sigma_{p,i}(u_i)} \;\middle|\; \begin{array}{l} 1 \leq i \leq m \\ 1 \leq j \leq k \\ j \neq u_i \end{array} \right\}.$$

On obtient un système à $m(k-1)$ contraintes linéaires en passant par le logarithme :

$$\left\{ \begin{array}{l} \log x_{\sigma_{1,i}(j)} + \log x_{\sigma_{2,i}(j)} \\ - \left(\log x_{\sigma_{1,i}(u_i)} + \log x_{\sigma_{2,i}(u_i)}\right) \leq 0 \end{array} \;\middle|\; \begin{array}{l} 1 \leq i \leq m \\ 1 \leq j \leq k \\ j \neq u_i \end{array} \right\}.$$

Le logarithme ajoute n contraintes supplémentaires : pour $i = 1, \ldots, n$, on doit avoir $x_i > 0$. La seconde préimage cherchée x^* est retrouvée en prenant l'exponentielle des composantes de la solution de ce système. Notons que les composantes x_i du vecteur de caractéristiques sont dans $\mathbb{R}$, tandis que celles de notre seconde préimage x^* sont toutes positives. L'attaque par inversion ne va donc pas bien fonctionner.

Mise en oeuvre. L'attaque par authentification est mise en oeuvre avec les paramètres proposés par les auteurs : $k = 128$ (taille des fenêtres), $m = 600$ (nombre de permutations) et $p = 2$ (nombre de produits de Hadamard). Le taux de succès est de 100% comme pour la transformation précédente.

Dans le cas d'une attaque par préimage réutilisable, la distance moyenne obtenue est de $0.75m$ en moyenne et de $0.97m$ dans le pire des cas, soit encore un taux de succès de 100% en moyenne (rappel : $\tau_B = 0.89m$).
Si on se place dans le cas où un attaquant possède N templates biométriques, avec $2 \leq N \leq 4$, on améliore les distances obtenues, surtout dans le pire cas, comme

N	1	2	3	4
Matching Score – pire cas (%)	3	12	14.7	14.8
Matching Score – cas moyen (%)	25	28	29	31

TABLE 3.2 – Attaques par préimage réutilisable sur URP-IoM

présenté ci dessus dans la table 3.2. Remarque : cette table donne le nombre de coordonnées identiques (*matching score*) en pourcentage.

Attaque par associativité sur URP-IoM. On utilise l'attaque précédente pour obtenir une attaque par associativité. Cela peut aussi être utilisé sur GRP-IoM, mais la présence d'une attaque par inversion la rend moins intéressante. Celle-ci obtient toutefois un taux de succès de 97% pour un attaquant n'utilisant pas de fonction objectif lors du calcul des préimages (non décrit). Par ailleurs, la récente stratégie proposée dans [60] n'a pas été retenue car celle-ci était peu adaptée à notre contexte.

L'attaquant possède deux templates $u = (u_1, \dots u_m)$ et $u' = (u'_1, \dots u'_m)$, ainsi que les secrets s et s' correspondant et veut déterminer si u et u' viennent de la même personne. L'algorithme prend en entrée u, u', s et s' et retourne vrai si u et u' viennent de la même personne et faux sinon. Il s'effectue comme ci-dessous :

1. De u, u', s et s', on calcule x et x' comme dans la section précédente.
2. On calcule le coefficient de Pearson $\rho(x, x')$ de x et x'. Si celui-ci est inférieur à un certain seuil τ, l'algorithme retourne vrai, et faux autrement.

Pour savoir si cet algorithme fonctionne correctement, il faut déterminer le seuil optimal τ et le taux de succès obtenu avec ce seuil.

Expériences. L'expérience est réalisée sur la même base (100 individus, avec 5 empreintes par individus). On commence par générer 500 templates biométriques à l'aide de 500 secrets différents, et on reconstruit une seconde préimage pour chaque template comme l'attaque précédente pour former une base B de 500 secondes préimages (dont les caractéristiques sont toutes positives). Dans un second temps, on met en oeuvre le jeu suivant pour une certaine valeur de τ et $N = 10000$:

1. $c_1, c_2 \leftarrow 0, 0$
2. Pour i entre 1 et N :
   a) On tire au hasard x et x' dans B de la même personne (avec $x \neq x'$)
   b) Si $|\rho(x, x')| \geq \tau : c_1 \leftarrow c_1 + 1$.
   c) On tire au hasard y et y' dans B de deux personnes différentes
   d) Si $|\rho(y, y')| \leq \tau : c_2 \leftarrow c_2 + 1$.
3. Retourner c_1/N et c_2/N

Le calcul du coefficient de Pearson retourne une valeur réelle entre -1 et 1. On a choisit la valeur $\tau = 0.18$, car elle permet de retourner la même valeur pour c_1 et c_2 à la fin du jeu précédent. Dans ce cas on obtient $c_1/N \simeq c_2/N \simeq 0.83$. L'attaque ci-dessus a été réalisée plusieurs fois (nouveaux templates, nouveaux secrets) en retournant un taux de succès similaire de 83% à chaque fois. Notons que le temps

de calcul est négligeable comparé à la génération des 500 templates biométriques (de l'ordre de 1 seconde sur un PC standard en python). Le choix du coefficient de Pearson pour le calcul de corrélation a été choisi arbitrairement et a été conservé compte tenu des bons résultats produits. Cela peut être modifié si besoin.

Conclusion. Nous avons présenté des attaques différentes sur plusieurs transformations biométriques. L'utilisation de la programmation linéaire avait déjà été mise en oeuvre pour attaquer le biohashing en 2010 dans [104], puis en 2014 par Y. C. Feng, M.-H. Lim et P. C. Yuen, en la combinant par une attaque dite par *masquerade* dans [48], où l'attaquant doit posséder énormément de données. En 2016, B. Topcu, C. Karabat, M. Azadmanesh et H. Erdogan attaquent aussi le biohashing dans [140] en ajoutant certaines hypothèses. L'approche logarithmique et l'attaque par associativité utilisée dans URP-IoM et de manière générale la formalisation proposée sont nouvelles. Nous n'avons pas utilisé de méthodes de programmation quadratique ou géométrique car celles-ci demandent que les matrices définissant les contraintes soient définies positives. Ce point mériterait d'être creusé en perspective.

Perspectives. Une première suite à cette partie consiste à formaliser la notion d'attaquant. Lors d'un calcul du taux de fausse acceptation, l'attaquant est tiré aléatoirement parmi les autres empreintes (vecteurs caractéristiques) de la base. Cela donne un attaquant fort que l'on peut considérer comme *ayant accès au système*. On peut aussi tirer aléatoirement chaques caractéristiques du vecteur de l'attaquant dans un certain intervalle. Ce type d'attaquant est plus faible : si on calcule le FAR, celui-ci est beaucoup plus bas que le précédent. C'est celui-ci qui est considéré dans l'attaque avec les algorithmes génétiques. On peut le considérer comme un attaquant *ayant connaissance du système*. Un troisième type d'attaquant serait un attaquant ayant connaissance du système et ayant accès à un autre système (d'acquisition), ce qui demanderait une seconde base d'empreinte provenant des mêmes individus.

D'autres perspectives directes de ces travaux concernent la généralisation des attaques décrites dans ce chapitre, que ce soit au niveau des paramètres ou au niveau des transformations considérées. De manière générale, il y a trop peu d'études de sécurité sur ces transformations. Une des raisons est l'absence de base de référence, pouvant servir d'une certaine manière de vecteurs tests. Ces attaques doivent aussi prendre en compte la notion de quantité d'information contenue dans les vecteurs de caractéristiques. Faire le lien entre la sécurité de ces transformations et des études beaucoup plus théoriques comme celle de T. Ignatenko et F. M. Willems en 2015 [75] sur des transformations génériques serait aussi intéressant.

Chapitre 4

Analyse forensique de dumps

Sommaire

4.1 Détection des données aléatoires 44
4.2 Analyse textuelle des dumps 46
4.3 Recherche de dates 49

Les cartes à puce contiennent de nombreuses données personnelles liées au porteur, à son identité ou à l'utilisation de la carte. Un dump de mémoire est formé de l'ensemble des données mémoires à un instant donné. Dans cette partie, on considère les données non volatiles contenues dans l'EEPROM qui sont récupérées en communicant avec l'API de la carte (il est donc possible que les dumps considérés ne soient pas exactement la copie de la mémoire). Les dumps mémoires de 371 cartes ont été récupérés, incluant notamment des cartes de transport, des forfaits de ski, des cartes de paiement EMV, des cartes vitales ou encore des passeports électroniques.

L'analyse forensique de ce type de données diffère fortement de celle de la recherche de fichier dans la mémoire d'un disque, à cause de la quantité de données et de l'absence de méta données [120]. L'analyse automatique d'environnements non-standards, tels que la mémoire non volatile des cartes à puce reste un challenge en forensique [56]. Un rare exemple d'analyse automatique de dumps a été réalisé en 2011 par T. van Deursen, S. Mauw et S. Radomirovic dans le cadre des applications de transport, mais celui-ci est loin d'être complet [40].

Cette analyse a été réalisée pendant la thèse de Thomas Gougeon entre 2014 et 2017, sous la direction de Christophe Rosenberger et Gildas Avoine, avec Morgan Barbier [62]. Trois directions différentes ont été prises pour cette analyse. La première a consisté à séparer les données non aléatoires des données aléatoires (comme des clés). La seconde a été de repérer les données textuelles de la carte, tandis que la dernière a consisté à retrouver les dates présentes dans les dumps mémoires. Chacun de ces points a fait l'objet d'une publication [63, 64, 65].

4.1 Détection des données aléatoires

La première approche a consisté à séparer les données aléatoires (à priori cryptographiques) des données non aléatoires des dumps. Il n'est à priori pas possible d'utiliser directement sur nos données des tests statistiques comme ceux du NIST [125], car celles-ci sont globalement trop peu nombreuses et surtout elles contiennent des séquences courtes qui ne sont pas uniformes (des données aléatoires peuvent succéder à des données textuelles). De plus, comme les champs de données ne sont pas bien définis, il faut tester les données bit par bit. Certaines études se rapprochent de cette problématique, comme la recherche de clés RSA en 1999 par A. Shamir et N. van Someren [128], ou encore celle de clés AES en 2009 par J. A Halderman *et al.* [68] dans le contexte des attaques par démarrage à froid (*cold boot attacks*). Ces travaux sont toutefois très ciblés et s'avèrent peu adaptés à notre contexte.

Pour tester si un bit est aléatoire ou non, on réalise des tests sur des séquences de longueur l, en fenêtres glissantes de longueur s, qui vont retourner des p-valeurs. Comme $s \leq l$, chaque bit est testé dans plusieurs séquences et un score est calculé à l'aide de ces p-valeurs (par exemple la moyenne des p-valeurs) pour déterminer la classe de chaque bit (aléatoire ou pas). L'objectif est de déterminer les tests statistiques et les paramètres de tests les plus pertinents.

Boosting et AdaBoost. Ces tests statistiques sont considérés comme des classifieurs faibles et sont combinés ensemble pour produire un classifieur fort à l'aide d'une technique appelé *boosting*, étudié par R. Schapire en 1990 [126]. L'algorithme de boosting choisi est l'algorithme adaptatif AdaBoost proposé en 1997 par Y. Freund en R. Schapire [49]. L'algorithme, tel que décrit par Y. Freund, R. Schapire et N. Abe dans [50], part d'un ensemble d'apprentissage $A = \{(x_i, y_i), i \in \{1, \ldots, n\}\}$, où x_i est un vecteur de caractéristiques décrivant l'objet i et $y_i \in \{-1, +1\}$ sa classe. Le principe consiste à maintenir à jour une distribution de poids sur chaque élément de A (initialisée à l'identique) en appliquant à chaque tour t entre 1 et T un classifieur faible et en renforçant les poids des éléments mal classifiés. Le classifieur fort correspond à un vote majoritaire (avec poids) des classifieurs faibles.

Phase d'apprentissage. On génère un dump synthétique de 100000 bits, contenant approximativement 35% de données aléatoires, avec des séquences entre 80 et 300 bits, pour chaque classe. Ce dump synthétique est coupé en deux, pour obtenir un dump d'apprentissage et un dump de validation. Les tests statistiques considérés ici contiennent 8 tests du NIST, ainsi qu'un test d'auto-corrélation et deux tests sur les courtes séquences ST et TBT (F. Sulak, 2013 [136] et P. M. Alcover, A. Guillamon et M. del Carmen Ruiz, 2013 [2]). L'ensemble de ces tests représente environ 10000 tests en prenant en compte les différents paramètres (longueur des séquences entre 32 et 256, des décalages entre 10% et 100% de la longueur des séquences, quatres fonctions de score, autres paramètres liés aux tests).

Le nombre optimal T de classifieurs faibles à utiliser a été déterminé de manière expérimentale : pour $T = 14$, on obtient un taux de reconnaissance de $94,14\%$ pour le dump d'apprentissage et 91.3% pour le dump de validation. Pout T plus grand, ce taux augmente sur le dump d'apprentissage, mais n'augmente plus sur

celui de validation (phénomène de sur-apprentissage). Notons qu'une légère variation du dump d'apprentissage retourne un classifieur final différent, avec un taux de reconnaissance sur le dump de validation assez similaire. Celui-ci (les 14 tests) n'est donc pas décrit ici car il dépend trop des données d'apprentissage.

Expérimentation. Le classifieur entrainé sur le dump synthétique est ensuite appliqué sur des données provenant de cartes réelles, pour lesquelles la vérité terrain est connue. En pratique, cela représente 133 dumps différents contenant en tout plus de 1300000 bits dont environ 180000 sont aléatoires. La table 4.1 affiche des taux de réussite supérieurs à ceux sur le dump synthétique. L'explication la plus probable est que le dump synthétique (validation) était généré pour être le plus représentatif de tout ce qui peut se trouver dans une carte, tandis que l'expérience sur de vrais dumps a été réalisée sur seulement 6 applications qui sont peut-être moins représentatives. La figure 4.1 montre le résultat sur un dump d'une carte EMV avec le code couleur suivant : vert pour un bit non aléatoire bien classé, bleu pour un bit aléatoire bien classé, rouge pour un bit non aléatoire mal classé et violet pour un bit aléatoire mal classé. Les bits mal classés sont soit à la frontière entre deux grands ensembles, soit des séquences courtes (≤ 100) au sein d'un ensemble important (≥ 1000).

FIGURE 4.1 – Reconnaissance de bits aléatoires dans un dump EMV [62]

Une deuxième série d'expérience est réalisée sur ces 133 dumps à l'aide d'un classifieur final utilisant un seul test statistique (*approximate entropy* sur des séquences de longueur 256 bits avec un décalage de 128 bits entre chaque séquence). On voit dans la table 4.1 que les résultats sont proches du classifieur précédent et que la technique de boosting n'apporte pas un gain si important sur les 133 dumps testés

(augmentation des faux positifs de 2% mais baisse des faux négatifs de 0.5%). Elle peut s'avérer néanmoins intéressante sur un nombre de dumps plus important.

Appl.	Dumps	Aléa	Taux (14)	Taux (1)	Non aléa	Taux (14)	Taux (1)
EMV	27	135336	93,8%	91,4%	381344	96,4%	98%
Passeport	6	28912	94,5%	93,7%	40076	96,5%	91,2%
Vitale2	4	17360	94,1%	93,4%	253824	99,5%	99,6%
Vitale1	3	0	-	-	87630	99,7%	100%
Calypso	88	0	-	-	420456	99,9%	100%
Moneo	5	0	-	-	24128	100%	100%
Total	133	181608	94%	92%	1207458	98,5%	99%

TABLE 4.1 – Taux de reconnaissance (1 ou 14 classifieurs faibles)

Analyse multi-dumps. Les données contenues dans des dumps provenant de la même application sont stockées au même endroit. Ce n'est pas forcément le cas quand les cartes proviennent d'opérateurs différents (par exemple, les forfaits de ski provenant de stations différentes stockaient les données avec le même encodage, mais pas au même endroit). On combine les résultats par un vote majoritaire, pour obtenir un taux de reconnaissance qui s'approche de 100% pour toutes les applications, à partir de trois dumps, même dans le cas particulier où le classifieur final n'est composé que d'un seul test statistique.

4.2 Analyse textuelle des dumps

L'objectif de cette seconde partie est de retrouver les informations textuelles dans les dumps, typiquement, les noms et prénoms du propriétaire de la carte, de la société qui la fournie ou encore d'adresses physiques ou électroniques. Un outil, appelé `Bulk Extractor`, a déjà été développé en 2013 sur cette problématique par S.L. Garfinkel [57]. Neanmoins, les fonctions de décodage et les décalages sont limitées et il y a énormément de faux positifs générés.

```
1. 702361214F07A0000000421010500243428701019F120E5452414E53414354494F4E204342
2. 7021611F4F07A0000000031010500A5649534120444542495487010 29F1204 56495341
3. 70615F201A4A4F484E2F534D4954482E4D52202020202020202020202020205F300202018C1B9F02069F03069F1A0
   295055F2A029A039C019F37049F45029F4C088D1A8A029F02069F03069F1A0295055F2A029A039C019F37049F4C08
4. 70369F0702FF008E0E0000000000000000020301031F009F0D059800B420009F0E050050480000 9F0F05B820B4F80
   05F280202509F4A0182
5. 70329F080200028F01069F32010392243459245 1B87DA8C05BA7F1DE5DC802BF59D394D6CC034A046F46995E0245E
   437AED7B899
6. 00000000135040025009781003 2600

ICC     00 00 00 00 00 00 00 04 00 71 B3 00 00 00 00 00 01 B8 B2 4A 02 50 00 33 01 1A 13 43 00
Holder1 04 00 98 E5 94 C8 02 0D 60 C9 65 C7 D5 90 00 00 00 00 00 00 00 19 75 07 10 92 82 D2 CF
Holder2 F3 6A 68 88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
EvLog1  09 0E E5 92 04 20 60 86 60 00 00 00 00 1C D6 DD 56 40 00 01 C0 00 00 51 08 66 E0 00 00
EvLog2  09 0E E5 7A 04 20 60 86 60 00 00 00 00 1C D6 DD 56 40 00 01 80 00 00 11 08 66 E0 00 00
EvLog3  09 0E E5 5A 04 20 60 86 60 00 00 00 00 1C D6 DD 56 40 00 01 40 00 00 91 08 66 E0 00 00
ConList 11 2B 40 01 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

FIGURE 4.2 – Exemple de dumps anonymisés (EMV et Calypso) [62]

La figure 4.2 donne deux exemples de types de données que l'on à récupéré (à l'aide du logiciel `Cardpeek`). Dans le premier dump (EMV), la première ligne comporte la chaîne *Transaction CB* en ASCII. La seconde ligne deux fois les mots *Visa*, puis *Debit* en ASCII. La troisième ligne contient le nom du porteur de carte en ASCII (*John Smith. Mr*). D'autres données se trouvent dans ce dump (date, transaction ou une partie de la clé publique sur la sixième ligne). Les données soulignées du second dump contiennent la date de naissance du porteur ainsi que son nom, sur deux lignes, en encodage 5-bits, ne respectant pas les octets. Notons que le standard Calypso laisse le type d'encodage utilisé à la discrétion de l'opérateur du transport public qui fournit ces cartes (qui ont donc du être décodées manuellement pour avoir une vérité de terrain pour nos expériences).

Analyse lexicale. La première partie, appelée analyse lexicale, prend une fonction de décodage en entrée (code ASCII, codage 5-bits, ..), génère les caractères correspondants à partir du fichier binaire et les assemble en chaînes de caractères. Notons que l'ensemble des données d'un dump n'est pas nécessairement encodé de la même façon. De plus, le bit de poids fort de l'information n'est pas forcément aligné sur les octets. Ainsi, il est nécessaire d'encoder l'ensemble des données binaires en prenant en compte tout les décalages possibles (entre 1 et 8 pour un codage ASCII).

Cette première partie va donc générer énormément de faux positifs. De plus, sans la connaissance de la longueur des données, les données textuelles vont être bruitées, même si la bonne fonction de décodage est appliquée. Par exemple, dans une carte Calypso, la date de naissance du porteur est située à côté de son nom. Si on applique l'encodage 5-bits à la suite `0x19750710` correspondant à la date de naissance d'un certain `james`, l'analyse lexicale retourne `lwjaxprjames` au lieu de `james`. Ces données vont donc devoir être retravaillées afin d'en extraire l'information.

Analyse syntaxique. L'objectif de cette phase est d'identifier parmi toutes les chaînes générées celles qui contiennent une sous chaîne ayant un sens pour un être humain (dans ce cas, la chaîne est appelée information). Bien sûr, une chaîne textuelle encodée avec une mauvaise fonction de décodage ne sera pas considérée comme une information (appelée non information). Notons qu'il n'est pas impossible qu'une chaîne qui ne soit pas une information et qui est décodée avec une mauvaise fonction de décodage puisse aparaître comme une information (pas courant mais possible avec de courtes chaînes). Ce genre de cas ne pourra pas être traité dans l'analyse syntaxique, mais l'analyse multi-dump éliminera la plupart de ces cas particuliers.

La première étape consiste à générer un corpus de chaîne contenant l'ensemble des informations textuelles possibles, afin de vérifier pour toute chaîne si une sous chaîne appartient à ce corpus. Il est néanmoins difficile de créer un dictionnaire exhaustif de noms propres et communs (particulièrement en plusieurs langues). Ce corpus va donc être complété par un classifieur bayésien naïf pour déterminer si une information a un sens, même si elle n'est pas dans le corpus. Ce classifieur va calculer deux scores de confiance, le premier sur l'hypothèse que la chaîne soit une information et l'autre sur l'hypothèse que la chaîne ne soit pas une information.

Pour une chaîne x, on définit un vecteur de caractéristique $(x_1, \dots x_n)$ où x_i est une sous chaîne de x. On note $P(x_i|I)$ la probabilité conditionnelle qu'une information contienne x_i, calculée à l'aide d'un corpus d'apprentissage. On calcule la probabilité $P(x|I)$ qu'une chaîne x soit une information par le produit $\prod_{i=1}^{n} P(x_i|I)$, hypothèse d'indépendance qui fait que le classifieur Bayesien est appelé naïf. Si cette probabilité est supérieure à $P(I)$, alors la chaîne x est marquée comme information Dans le calcul du produit précédent, si l'une des probabilité conditionnelle est nulle, celle-ci est ignorée pour éviter que le produit ne soit nul.

Validation du protocole. On génère un ensemble S de 10000 chaînes où la moitié sont des informations, qui ont été retirées pour le coup du corpus d'entrainement, et l'autre moitié ne le sont pas. Le corpus d'entrainement est formé de 400000 mots, provenant du dictionnaire français de John the Ripper (150000) et d'un second ensemble contenant 250000 mots, principalement des noms et des prénoms américains. D'autres corpus ont été testés, sans donner de meilleurs résultats. Les 5000 chaînes de non informations sont formées par des caractères alphabétiques aléatoires, concaténés dans des chaînes de longueur entre 4 et 15, dont on a vérifié la non appartenance au corpus d'apprentissage.

La première partie consiste à vérifier si une chaîne contient un mot dans le corpus. Si la taille de ce mot est trop petite, cela retourne trop de faux positifs. Une longueur optimale a été évaluée expérimentalement à 5 ou plus. Cette partie reconnaît plus de 76% des informations, sans générer de faux positifs (non information reconnue comme information). La seconde partie consiste à appliquer le classifieur bayesien naïf sur S, avec une probabilité $P(I) = 0.5$. On obtient environ 98% de reconnaissance des informations sans augmenter réellement les faux positifs ($\simeq 1.5\%$).

Expérimentations. Les expérimentations sur les 371 dumps ont pris en compte plusieurs fonctions de décodage, dont l'ASCII (8 bits) et les décodages 5-bits et 5-bits-8 (ce dernier utilisant un octet pour stocker l'information). L'encodage 5-bits est bien présent dans les cartes Calypso et dans les cartes vitales. L'analyse lexicale va générer 190093 chaînes de caractères qui sont confrontées à la vérité terrain. La table 4.2 donne les taux de reconnaissance des informations (RR_I) et des non-informations ($\mathrm{RR}_{\bar{I}}$) pour les trois fonctions de décodage les plus courantes.

Décodage	RR_I	$\mathrm{RR}_{\bar{I}}$
ASCII	99,7%	93,12%
5-bits	100%	96,7%
5-bits-8	98,7%	96,9%

TABLE 4.2 – Taux de reconnaissance après l'analyse syntaxique

Notons aussi la présence très limitée de l'encodage uuencode (Unix-to-Unix) sur 6 bits dans les cartes vitales. Sans surprise, les faux positifs sont des mots courts ou des mots longs contenant au milieu des chaînes courtes ayant un sens. Par ailleurs les encodages ASCII-7 et Base64 n'ont pas été trouvés dans nos dumps, mais ont générés respectivement $2,6\%$ et 3% de faux positifs.

Analyse multi-dumps. Les données de cartes provenant de la même application ont souvent des structures de mémoire similaires. Ainsi, on peut espérer qu'une chaîne, stockée à un indice donné, appartienne à la même classe. Notons qu'une information n'aura pas toujours la même longueur pour deux dumps différents, par exemple le nom du porteur. L'analyse multi-dump a été testée sur 287 dumps à l'aide d'un vote majoritaire, les autres dumps appartenant à des applications pour laquelle on ne possédait qu'une ou deux cartes. Le taux de reconnaissance de l'information passe à 100%, avec une baisse des faux positifs (moins de deux par dumps).

Ces expériences ont été réalisées sans la recherche des données aléatoires de la section précédente. Si on applique ce premier filtre, les faux positifs sont réduits après l'analyse syntaxique, mais les résultat deviennent identiques après l'analyse multi-dumps. Par contre, les faux négatifs de la section précédente seront perdus.

4.3 Recherche de dates

La dernière partie de cette analyse est la recherche automatique de dates. Cela inclut la date de naissance du porteur de carte, la date de création ou d'activation de la carte, ou encore des dates de transactions. N'importe quelle chaîne de caractère peut potentiellement définir une date, représentée en mode compteur, ce qui implique l'importance du contexte de la recherche.

Phase de décodage. La première étape consiste à appliquer toutes les fonctions de décodage possibles sur les dumps. Il existe de nombreuses fonctions pour les dates, regroupées en deux types : format et compteur. Les fonctions de type format représentent une date par une suite de chiffre composée par exemple du jour, du mois et de l'année au format ASCII ou BCD. Les fonctions de type compteur comptent la date à partir d'une date initiale (temps UNIX, date de création de la carte) en fonction d'une certaine unité de temps (seconde, jour,...).

La fonction de décodage prend en entrée une chaîne de m bits et sort une date existante dans le calendrier grégorien ou trois valeurs 0, 1 et -1 où les deux premières valeurs correspondent respectivement à une entrée composée uniquement de 0 ou de 1 et -1 correspond à une sortie en dehors du calendrier grégorien (par exemple si la fontion de décodage est de type format en ASCII ne correspond pas à un chiffre).

Analyse multi-dumps. On élimine tous les candidats qui sont dans l'une des deux configurations suivantes : si la fonction de décodage a retourné -1 sur l'un des dumps, ou si la fonction de décodage a retourné 0 ou 1 pour tous les dumps (à priori une zone mémoire non utilisée par l'opérateur). Les sorties à 0 ou 1 sont toutefois autorisées si elles apparaissent sur un nombre réduit de candidat, car elles peuvent représenter des zones prévues pour des dates qui n'auraient pas encore été remplies (voyage pour une carte de transport, transaction pour une carte de paiement).

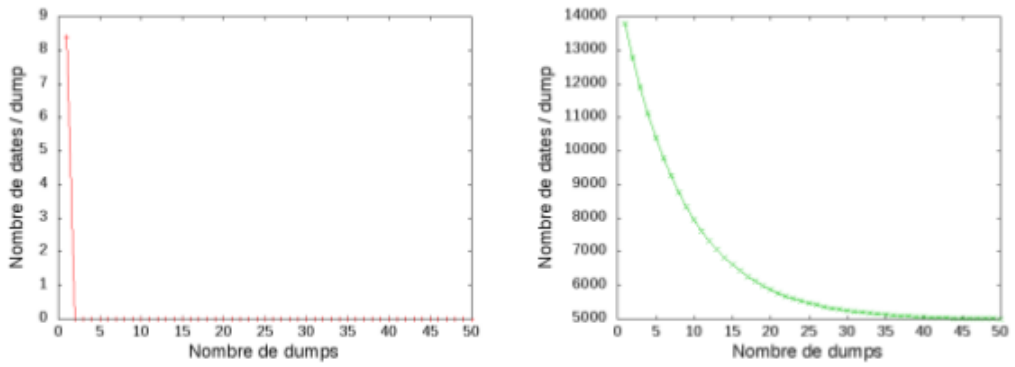
Ainsi, cette analyse multi-dumps ne peut pas générer des faux négatifs (éliminer des vraies dates). On verra dans nos expériences que l'analyse multi-dumps appliquée aux dates de type format sera très performante pour une dizaine de dumps seulement, mais éliminera peu de faux positifs pour des dates de type compteurs.

Expérimentations. Afin d'évaluer ces deux premières phases de l'analyse, nous avons comparé les résultats entre un dump totalement aléatoire et deux dumps provenant de cartes de transport (Calypso et OV-chipkaart). Le dump aléatoire est composé de 5000 bits générés par `/dev/urandom/`, celui de Calypso contient aussi environ 5000 bits, tandis que celui d'OV-chipkaart contient 512 bits. Plusieurs fonctions de décodage sont utilisées (par exemple des formats de type ASCII, BCD et BCD-8, où ce dernier est BCD complété jusqu'à 8 bits avec 2 bits à zéro). Leur description précise se trouve dans [62]. Le nombre de dates potentielles générées par la phase de décodage est décrite table 4.3. Sans surprise, elles sont nombreuses ...

Application	Nombre de dates	Dont format	Dont compteur
Aléatoire	13805	9	13796
Calypso	5555	70	5485
OV	1379	11	1368

TABLE 4.3 – Nombre de dates générées par la phase de décodage

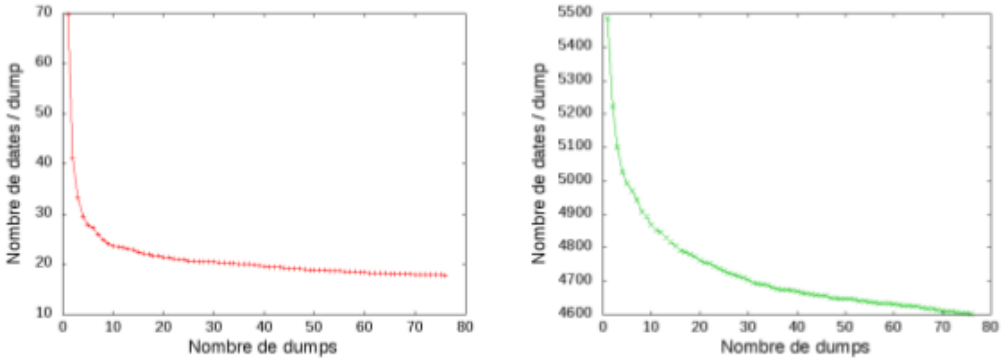
La performance de la phase multi-dumps sur le dump aléatoire est décrite figure 4.3. On remarque qu'il reste 5000 fausses dates de type compteur après l'analyse multi-dumps réalisée sur 50 dumps. Cela vient en partie du fait que certaines fonctions de décodage génèrent des dates qui sont toujours dans le calendrier grégorien (par exemple un compteur sur 14 bits qui compte le nombre de jours depuis le 1er janvier 1990). Les performances de la phase multi-dumps sur les dumps Calypso est décrite figure 4.4 et est similaire au dump aléatoire. L'analyse multi-dump n'est pas suffisante pour retrouver des dates dans les applications testées : il y a trop de faux positifs restants (par exemple, plusieurs milliers pour Calypso).



(a) Dates de type *format*. (b) Dates de type *compteur*.

FIGURE 4.3 – Analyse multi-dumps sur un dump aléatoire [62]

Analyse contextuelle. Une personne réalisant une analyse forensique sur des dumps de cartes peut avoir connaissance du contexte d'utilisation de cette carte et éliminer les candidats qui ne respectent pas ce contexte (typiquement un intervalle de temps). Dans la pratique, si on recherche une date dans un ensemble de données, c'est justement en fonction d'un contexte.



(a) Dates de type *format*. (b) Dates de type *compteur*.

FIGURE 4.4 – Analyse multi-dumps sur les dumps Calypso [62]

Par exemple, la recherche de dates de naissance dans les dumps Calypso retourne 189 résultats, ce qui est encore important. Mais en considérant qu'il est peu probable qu'un opérateur encode une date de naissance en mode compteur (en notant qu'un compteur démarrant en 1970 ou 1990 ne permet pas de coder toutes les dates de naissance), il ne reste que deux dates qui sont vraiment utilisées par l'opérateur pour encoder la date de naissance du porteur. D'autres dates ont également été trouvées comme des dates de déplacements, de transactions EMV, des dates de validité. Un autre exemple concerne la carte Vitale qui enregistre la date de naissance du porteur, celle de ses éventuels enfants, mais aussi le cas échéant les dates de début d'événements médicaux comme les maladies de longue durée.

Conclusion. Une analyse forensique doit combiner une recherche automatique efficace avec une recherche plus ciblée. Ainsi, la recherche automatique de dates montre que l'on obtient des résultats pertinents si et seulement si on prend en compte des éléments de contexte. Par ailleurs, le travail présenté dans ce chapitre a demandé un gros travail pour établir la vérité terrain, ce qui illustre très bien l'importance de l'automatisation de la recherche de ces données.

Notons que les résultats de cette section concernent avant tout la démarche et peuvent ne pas être adaptés à un autre contexte. Par exemple, la liste et le nombre des tests statistiques utilisés dans la première partie dépend fortement des données d'apprentissage. Notons enfin que l'accès aux données n'était que très rarement protégé (sauf l'accès aux données biométriques dans un passeport et de la photo dans les cartes vitales) et que ces données étaient stockées en clair.

Perspectives. Des applications potentielles de ces travaux concernent l'étude ou l'adaptation des techniques présentées dans cette section à d'autres cas d'usage. Par exemple, la détection d'anomalies dans la génération de données aléatoires, l'analyse de traces réseaux dans le cas où le protocole est inconnu (dans le cas contraire des outils comme `Scapy` ou `Wireshark` font très bien le travail).

Chapitre 5

Authentification des personnes

Sommaire

5.1 Attaque sur un CAPTCHA 54

5.2 Robustesse des mots de passe 57

Ce chapitre traite de deux aspects de l'authentification des personnes : l'attaque d'un Captcha et l'évaluation de la robustesse des mots de passe. Notons qu'un Captcha n'est pas directement lié à l'authentification des personnes, dans le sens où il ne cherche pas à authentifier un individu mais cherche à savoir si une entité est un individu. CaptchaStar est un nouveau type de Captcha, proposé par M. Conti, C. Guarisco et R. Spolaor à ACNS 2016 [35], qui est interactif, et où l'utilisateur bouge un curseur afin de reconnaître une forme. Dans cette section, on présente une attaque sur ce Captcha, basée sur la concentration des pixels dans la grille, avec un taux de succès de 96% en 12 secondes. Cette attaque a été réalisée en 2017 avec Thomas Gougeon en parallèle de sa thèse et a été présentée dans [66].

Dans une seconde partie, je me suis intéressé à l'évaluation des énumérateurs des mots de pase avec la thèse de Mathieu Valois, que j'ai co-encadré avec Jean-Marie Le Bars depuis septembre 2016. En effet, la stratégie d'évaluation commune de ces énumérateurs ne correspond pas bien à la réalité, en particulier parce que le temps d'énumération des candidats n'est pas pris en compte. Cette évaluation pose toutefois des problèmes car cela doit être le plus indépendant possible de la machine, et la complexité des algorithmes dépend de la base d'apprentissage. Nous avons notamment montré que les énumérateurs académiques ont des résultats relativement proche des logiciels spécialisés (JTR ou hashcat) dans le cas de fonctions de hachage standards et bien meilleurs dans le cas de fonctions de hachage lentes (de type Bcrypt ou Argon2). Cette étude a été publiée en 2019 dans [143].

5.1 Attaque sur un CAPTCHA

Un Captcha est un acronyme pour *Completely Automated Public Turing test to tell Computers and Humans Aparts*, proposé par L. von Ahn, M. Blum, N. J. Hopper et J. Langford en 2003 [144]. C'est un test qui doit être facile à réaliser pour un humain, mais difficile pour une machine. Les Captchas sont très variés, des bouts de texte tordus à écrire à de véritables jeux basés sur des images. Néanmoins, la plupart des Captchas sont résolus avec un taux de succès très important.

Les Captchas basés sur du texte étaient les plus répandus dans un premier temps car ceux-ci sont simples à comprendre. Des attaques ont vu le jour rapidement sur ce type de Captchas, basées sur des heuristiques très simples ou sur des approches tirées du *Machine Learning* [149, 23, 55]. Des alternatives, basées sur la reconnaissance d'images ou de données audio, n'étaient pas plus résistantes que les autres [59, 24, 131]. Un autre problème auquel sont sujet la plupart des Captchas sont les attaques par relais [103]. Pour se prémunir de ces attaques, on peut rendre le Captcha intéractif, par exemple sous forme de jeux ou comme le présent Captcha.

Présentation de CaptchaStar. CaptchaStar est un récent Captcha interactif où l'utilisateur bouge un curseur afin de reconnaître une forme. Les pixels de l'écran bougent avec le curseur et si celui-ci est proche d'une position précise, la forme apparaît. Dans ce cas, l'utilisateur n'a plus qu'à cliquer pour envoyer la position du curseur au serveur qui vérifie si cette position est suffisament proche de la bonne. Une démo est disponible sur le site internet des auteurs [36]. Avec les paramètres proposés par la démo, les auteurs estiment qu'un être humain est capable de résoudre le challenge en moins de 30 secondes avec un taux de succès de 87%. La figure 5.1 illustre les deux phases du processus pour la résolution du challenge (notons que l'image du milieu n'est pas suffisante pour résoudre le test).

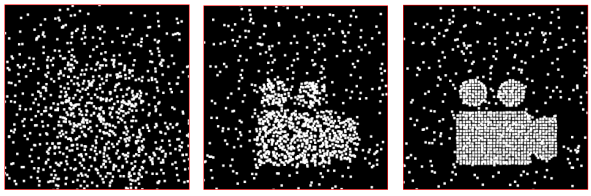


FIGURE 5.1 – Résolution d'un challenge par un humain sur [36].

CaptchaStar est formé d'une grille de 300×300 pixels où sont placés un certain nombres de carrés blancs de 5×5 pixels appelés *stars*. Lors de la génération d'un challenge, le serveur choisi aléatoirement une image à partir d'une base de données locale (5000 images en noir et blanc dans la démo, mais cela peut se changer). Cette image est discrétisée en transformée en *star* et ces *stars* sont mélangées de manière pseudo-aléatoire, dépendant d'une position dans la grille qui est la solution du challenge. L'algorithme utilisé n'est pas décrit ici car il n'est pas utilisé dans l'attaque, mais est décrit dans [35]. Il génère en plus un certain nombre de *stars*, appelées bruit, disposées aléatoirement dans la grille (par défaut 70% de bruit).

Notons $\mathbf{S} = \{s^i, 1 \leq i \leq n\}$ l'ensemble des n *stars* générées par l'algorithme à partir de l'image originale, où chaque s^i est definie par deux coordonnées $\left(s^i_x, s^i_y\right)$ dans la grille. Par ailleurs, $\psi \times n$ *stars* bruitées sont rajoutées aléatoirement dans la grille (par défaut $\psi = 0.7$, mais ψ peut être plus grand que 1). La solution exacte du challenge est la paire $sol = (sol_x, sol_y)$ où chacune des deux coordonnées est tirée aléatoirement dans l'intervalle $[5, 295]$. La solution du challenge est indépendante de l'image originale : deux images identiques offrent deux solutions différentes.

Heuristique choisie. La concentration des pixels est calculée de la manière suivante : pour chaque état S^k de la grille, on découpe celle-ci en un ensemble T^k de carrés de $\ell \times \ell$ pixels. Par convention, on considère qu'un pixel $t_{i,j} = 0$ si celui-ci est noir et $t_{i,j} = 1$ s'il est blanc. On calcule la concentration de pixels pour chaque carré $t^k \in T^k$, par le nombre de pixels blancs dans le carré :

$$\text{score}(t^k) = \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} t^k_{i,j}.$$

Pour chaque état S^k, on stocke de manière ordonnée l'ensemble de ces scores (il y en a autant que de carrés). Le score final de chaque état S^k est calculé en sommant les n_{max} valeurs maximales de cet ensemble. Par exemple, si on choisit $\ell = 10$ et $n_{max} = 20$ (valeurs choisies dans notre attaque), notre heuristique calcule le nombre de pixels blancs de chaque carré de taille 10×10 pixels, et les 20 scores les plus élevés sont pris en compte et sommés entre eux.

En pratique, cela fait $290 \times 290 = 84100$ états possibles pour lequel on doit calculer la concentration maximale, réalisé en calculant les scores de chacun des $30 \times 30 = 900$ carrés (si $\ell = 10$) et en sommant les $n_{max} = 20$ scores les plus élevés. C'est tout à fait possible à faire en un temps raisonnable, mais nous avons choisi une approche plus efficace pour des raisons de performance.

Méthodologie. L'attaque est décomposée en deux parties : dans un premier temps, on parcourt grossièrement la grille en ne calculant les scores que pour un sous ensemble d'états S^k. Dans un second temps, on considère l'état qui a donné le score le plus élevé dans la première phase et on parcourt ensuite tous les état possibles autour de cet état. Cela revient d'une certaine manière à parcourir la grille comme le fait un être humain pour résoudre le challenge.

Plus formellement, la première phase calcule le score pour n_s états en découpant la grille en n_s carrés (on utilise le centre comme coordonnée). Dans un second temps, on calcule les score pour tous les états autour de la solution de la première phase, où le curseur se situe dans un carré de taille $\ell_2 \times \ell_2$. Il y a donc seulement $n_s + \ell_2^2$ scores à calculer. Nous avons choisi les valeurs $n_s = 2500$ et $\ell_2 = 20$ pour notre attaque (2900 scores à calculer au lieu de 84100 ci-dessus).

La figure 5.2 illustre notre attaque avec les paramètres précédents, c'est-à-dire $\ell = 10$, $n_{max} = 20$, $n_s = 2500$ et $\ell_2 = 20$. On y distingue en particulier la solution approximative de la première phase.

FIGURE 5.2 – Les deux phases de l'attaque sur le site [36].

Implémentation. L'attaque est implémentée en Javascript (Chromium sur Ubuntu 14.04) Le code est directement exécuté sur la console du navigateur après connection au site internet de la demo. La vérification (comme la génération du challenge) est effectuée coté serveur, qui calcule la distance entre les coordonnées du curseur calculée par cette attaque et la solution du Captcha. L'attaque a été exécutée sur 1000 challenges, avec un taux de réussite de 96% et un temps d'exécution de 12 secondes en moyenne (à comparer avec les 87% de réussite en 30 secondes pour un individu légitime). Les quelques challenges qui n'ont pas été retrouvés correctement proviennent généralement d'images comportant un grosse zone de blanc.

Choix des paramètres. Le choix des paramètres (ℓ, n_{max}, n_s, et ℓ_2) a été réalisé expérimentalement sans recherche exhaustive du meilleur choix. La première raison est que plus ℓ_2 et n_s augmentent, plus le taux de réussite est élevé, tout comme le temps de calcul. La seconde raison est que l'ensemble des paramètres optimaux est directement lié à l'implémentation de CaptchaStar (notamment aux paramètres de génération du challenge). Une recherche exhaustive n'a donc que peu d'intérêt. Il est aussi possible de choisir les paramètres de manière dynamique, par exemple en fonction du nombre de *stars* dans le challenge proposé, afin d'améliorer le taux de réussite, même sans avoir la connaissance des paramètres de génération.

Variantes. Nous avons aussi testé l'attaque précédente en modifiant la génération du challenge de deux manières (assez proches en pratique) :

1. En augmentant la quantité de bruit lors de la génération,
2. En augmentant le nombre d'image dans le challenge (non décrit ici).

Nous avons ajouté 200% de *stars* supplémentaires (bruit), en plus des 70% par défaut, ce qui est supérieur au taux $\psi = 250\%$ envisagé par les auteurs. Nous avons lancé l'attaque sur 100 challenges et avons obtenu un taux de réussite qui tombe à 91%. Ce taux est toutefois toujours supérieur au taux de réussite d'un individu légitime qui baisse à 75% quand $\psi = 250\%$ selon les auteurs. Le taux de réussite de la seconde variante est similaire et même supérieur en général à la première variante car le bruit ajouté est généralement inférieur à 200%.

Conclusion. L'attaque proposée dans cette partie obtient un meilleur taux de réussite et un temps de calcul plus court qu'un individu légitime. Il semble difficile d'imaginer des contres-mesures efficaces. De plus, l'attaque présentée ici déplace le curseur de manière similaire à un être humain ce qui rend très difficile sa détection par une analyse de comportement.

5.2 Robustesse des mots de passe

La sécurité des mots de passe, liée à l'importante utilisation de mots de passe faibles est étudiée depuis longtemps [102, 47, 19], mais la discipline a vraiment progressée depuis une dizaine d'années, avec le développement de plusieurs énumérateurs très performants. Ces algorithmes apprennent des probabilités d'occurences sur une base d'apprentissage et énumèrent les mots probables à partir de cette base, comme proposé dès 2005 par A. Narayanan et V. Shmatikov [105]. Bien que ce ne soient pas les seuls, on considère ici l'énumérateur basé sur un modèle de Markov appelé OMEN [46] et celui basé sur une grammaire probabilistique, appelée PCFG [145]. Notons qu'une version adaptative et multimodale de ces énumérateurs a été récemment proposée par J. Galbally, I. Coisel et I. Sanchez dans dans [51, 52]. De tels énumérateurs sont souvent évalués par le rapport entre le nombre de mots de passes qui sont effectivement dans une base à attaquer et le nombre de mots de passes qui ont été énumérés. Une telle métrique, appelée *guess number*, correspond donc au taux de succès de l'énumérateur [85, 97, 142].

Cette métrique ne prend toutefois pas en compte l'efficacité d'énumération des candidats testés. C'est particulièrement un problème si on les compare avec des énumérateurs réputés pour leur rapidité de traitement que l'on peut trouver comme John The Ripper [110] (utilisé ici en mode Markov) ou Hashcat [133]. Ce phénomène est amplifié si une fonction de hachage rapide, comme SHA-256 ou SHA-3, est utilisée, tandis qu'il est amoindri avec une fonction lente comme Bcrypt [121], Scrypt [113] ou Argon2 [10, 18], à tel point que les auteurs d'OMEN ne conseillent d'utiliser leur énumérateur que pour ces fonctions lentes [45]. L'utilisation de ces fonctions est pourtant loin de se généraliser, comme le montre la figure 5.3 qui a été réalisée à partir des mots de passe fuités du site [73].

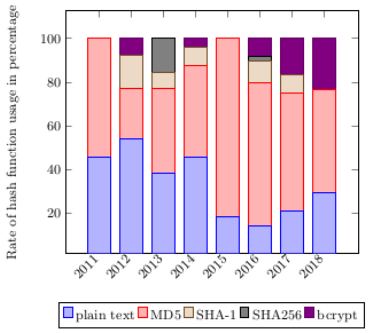


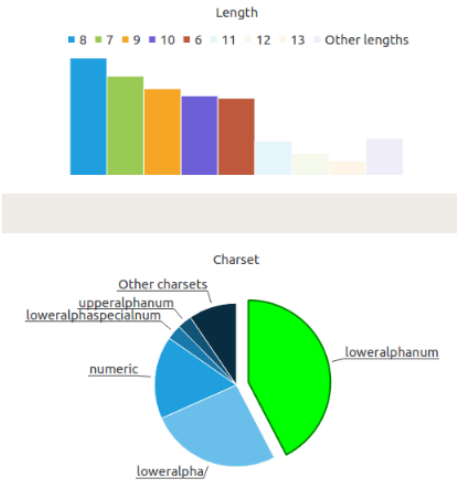
FIGURE 5.3 – Proportion des fonctions de hachage [143] (la date correspond au *leak*)

Bases de mots de passe. Les bases de mots de passe utilisées dans cette partie sont la base RockYou (32 millions de mots de passe dont 14 millions uniques, fuitée en clair en 2009) et la base Linkedin (160 millions de mots de passe dont 60 millions uniques, fuitée entre 2012 et 2016 où les mots sont hachés avec SHA-1). Des tests ont aussi été réalisés sur d'autres bases plus importantes tirées de [73]. Remarque : les bases ci dessus ne sont pas salées. Ce n'est pas nécessairement représentatif de la réalité. Ainsi, on ne considère pas les attaques dites par tables arc-en-ciel [108, 109].

Développement d'un outil d'analyse de mots de passe. Il existe peu d'outils d'analyse statistique de bases de mots de passe disponible. Le logiciel opensource `PACK` [82], écrit en python, est le plus utilisé, mais celui-ci n'est pas très efficace. Ainsi, pour une base de 500 millions de mots (soit une taille d'environ 5Go), l'analyse met environ 45 minutes pour se terminer. Nous avons donc décidé de développer notre propre outil d'analyse en C++, sur la base d'un projet étudiant à l'Ensicaen [12]. Le code source est disponible sous git et peut être utilisé soit en ligne de commande, soit avec une interface graphique. Il fournit différentes statistiques sur les jeux de caractères et sur des masques simples ou plus avancés. Par exemple, pour le mot P@ssword123 de taille 11, le jeu de caracère est { lowercase, uppercase, digit, special}, le masque simple est USLD et le masque avancé est USL6D3.

Cet outil offre la possibilité de filtrer les mots de la base avec des expressions régulières avant de les analyser. Par exemple, la base Linkedin possède de nombreux mots de taille 15 composés de chiffres, qui sont sans doute dus à des robots et qui biaisent les statistiques. L'utilisations de threads POSIX a aussi permis d'améliorer sensiblement les performances avec un gain allant jusqu'à 400% entre 1 et 8 threads (au delà de 8 threads, le gain est presque nul car le travail de chaque thread ne compense plus la lecture du fichier et la fusion des résultats). Une autre amélioration proposée par rapport à PACK est la gestion des caractères UTF-8 (notamment les accents en français). Notons qu'il faut savoir si un même caractère avec et sans accent doit être considéré comme deux caractères différents ou non.

L'interface graphique proposée est réalisée en Qt5 et requiert la librairie Qtcharts (facilement installable sur la version 18.04 d'Ubuntu). Cette interface est toutefois optionnelle. Au niveau des performances sur une base de 500 millions de mots de passe, quand PACK mettait 45 minutes pour son analyse, notre outil met 7 minutes avec un thread et moins de deux minutes avec 8 threads. La figure 5.4 montre la sortie de l'analyse sur une base d'environ 500 millions de mots de passe en mode graphique et en mode ligne de commande (extraits).




```
Statistics relative to charsets:
        loweralphanum:   45,91%      (228049679)
            loweralpha:  24,36%      (120999045)
              numeric:   8,605%      (42741330)
         mixedalphanum:   6,799%     (33772341)

Statistics relative to simplemasks:
          lowerdigit:    30,47%      (151370975)
               lower:    24,36%      (120999045)
               digit:    8,605%      (42741330)
          digitlower:    6,013%      (29868932)
       lowerdigitlower:  3,767%      (18712738)

Statistics relative to advancedmask:
      ?1?1?1?1?1?1?1?1?1?1?1:  15,06%      (74847808)
                  othermasks:  11,68%      (58034271)
?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1:  2,143%     (10644579)
```

FIGURE 5.4 – Outil d'analyse de mots de passe (496672991 mots) [12]

Motivation. Pour illustrer cette problématique, on considère la comparaison faite par les auteurs d'OMEN entre John the Ripper (JtR) en OMEN sur la base RockYou. La figure 5.5 correspond au Guess Number (à droite) et à une évaluation du temps de l'attaque à gauche en mode réel. Pour cela, on calcule simplement $N(T_E + T_H)$ où N est le guess number, T_E le temps moyen d'énumération et T_H le temps de la fonction de hachage. On prend $T_E = 10^{-6}$ secondes pour OMEN et 5×10^{-8} secondes pour JtR pour un PC standard, tandis que $T_H = 8 \times 10^{-11}$ secondes pour SHA-1. On y voit que le Guess number n'est pas suffisant pour évaluer un algorithme d'énumération. Par contre, si on évalue les performances de ces énumérateurs avec Bcrypt (pour un facteur de coût de 10), $T_H = 1,4 \times 10^{-4}$ secondes, l'évaluation du temps de l'attaque devient similaire au schéma avec le Guess Number.

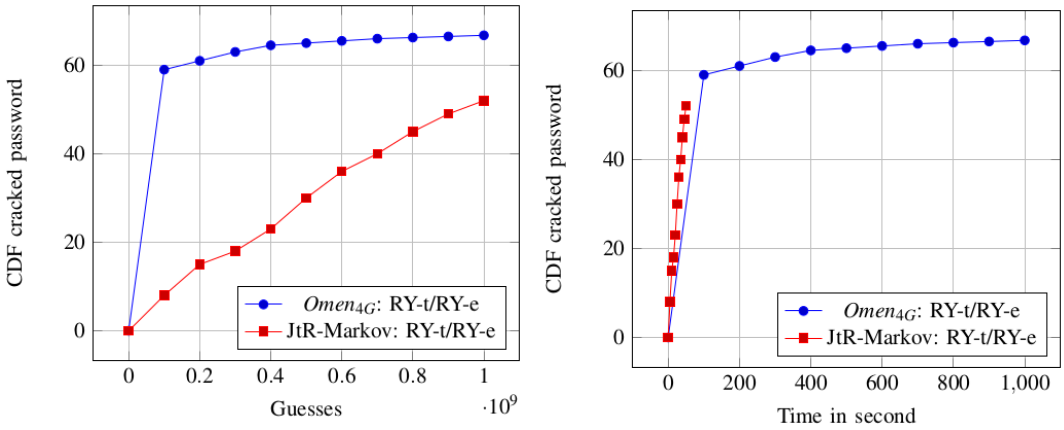


FIGURE 5.5 – Comparaison Guess number [46] / temps d'attaque réel (SHA-1)

Les performances données ci-dessus pour les fonctions de hachage sont évaluées avec une carte graphique GTX 1080 Ti et tirées de [61], où le temps de Bcrypt est adapté pour un facteur 10. Remarquons que certains matériels spécialisés dans le calcul de fonctions de hachage pour le minage de cryptomonnaies ont des performances bien supérieures. Par exemple, l'Antminer S17 Pro affiche entre 36 et 56×10^{12} calculs de SHA-256 en une seconde. De même, comme certaines cryptomonnaies utilisent des fonctions de hachage lentes pour le minage, il est possible que les performances de ces fonctions évoluent. Par exemple, l'ASIC Bitmain AntMiner L3 [4] affiche des performances de 596 millions d'empreintes hachées par seconde pour la fonction lente Scrypt, utilisée par les cryptomonnaies Litecoin et Dogecoin.

Evaluation de la performance. On note qu'il faudrait normalement prendre en compte le temps de recherche du candidat testé dans la base qui est attaquée. Néanmoins, ce temps peut être rendu négligeable à l'aide de techniques de type filtre de Bloom et n'est donc pas pris en compte. L'hypothèse d'un temps constant pour le calcul de la fonction de hachage est plutôt réaliste, contrairement au temps d'énumération qui est particulièrement variable, ce qui nécessite une étude plus approfondie. Enfin, une évaluation de la complexité algorithmique de ces énumérateurs est rendue difficile car celle-ci dépend énormément de la base où la phase d'apprentissage a été faite, on va donc se tourner vers une solution approchée.

Mesurer le temps de génération de chaque candidat est un processus trop compliqué. De même, mesurer le nombre de candidats générés sur une période de temps donnée demande un nombre trop important d'appels à une fonction `clock()` qui modifie le résultat. On va plutôt mesurer le temps de génération de n candidats. On choisit comme valeur de n le nombre approximatif de candidats générés en 0.1 secondes en début d'algorithme. Ainsi, n est différent pour chaque énumérateur (de 10^9 pour la force brute à 10^4 pour PCFG dans notre cas). Cette valeur de 0.1 seconde est choisie comme un bon compromis entre constance du temps d'énumération et contraintes (appels à la fonction `clock()`, stockage des résultats intermédiaires).

Expérimentations. La performance de chaque énumérateur est calculée sur chaque période de temps. Ces performances sont ainsi présentées figure 5.6 et 5.7 sur les bases Linkedin et Rockyou (en pourcentage de mots de passe crackés par rapport au nombre de mots de passe de la base). On y voit notamment que PCFG se comporte bien, même avec une fonction de hachage rapide comme SHA-1. Les sauts dans les courbes d'OMEN sont directement liés au fonctionnement de l'énumérateur. En effet, celui-ci génère des candidats par longueur, et les sauts correspondent à des changements de longueur. D'autres tests ont été réalisés en croisant les bases d'apprentissage, sans obtenir des résultats particulièrement différents. Les test utilisent les implémentations des auteurs, même si l'on posède notre propre version de PCFG.

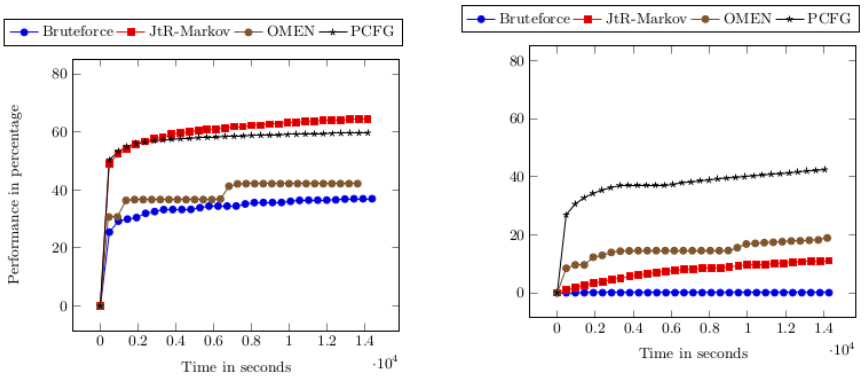


FIGURE 5.6 – Performance sur Linkedin avec SHA-1 (gauche) et Bcrypt (droite)

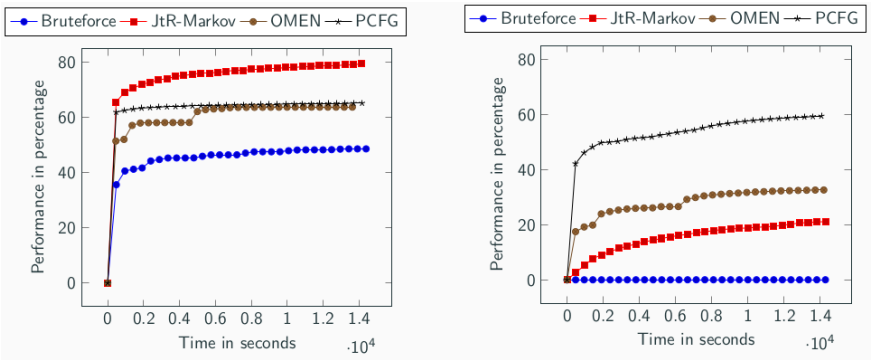


FIGURE 5.7 – Performance sur Rockyou avec SHA-1 (gauche) et Bcrypt (droite)

Politiques de sécurité. On présente ici comment se comportent les énumérateurs précédents (JtR, PCFG et OMEN) dans le cas où les mots de passe sont soumis a une politique de sécurité donnée. Les tests sont réalisés sur la base Myspace (308875476 de mots) qui est soumise à l'évaluateur de Dropbox `zxcvbn` (cinq niveaux de 0 à 4) [146], qui est reconnu comme l'évaluateur de mot de passe le plus abouti, en retirant les mots de passe de la base jugés faibles par `zxcvbn`. Au niveau 1, il reste 95.9 % de la base, au niveau 2, 42.4 %, au niveau 3, 14.6% et au niveau 4, 0.7% des mots de la base. Les résultats sont donnés figure 5.8.

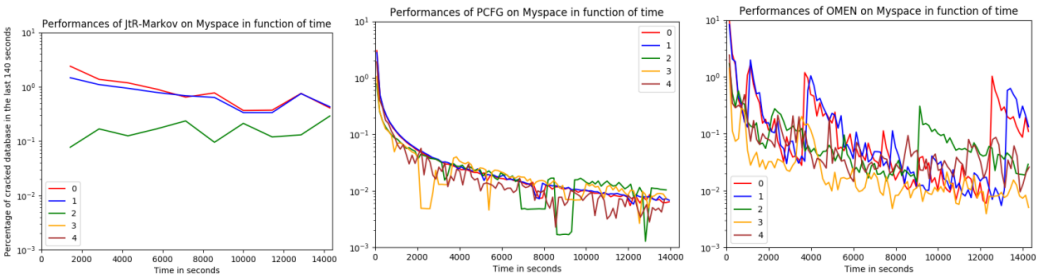


FIGURE 5.8 – Performance sur Myspace sous zxcvbn (0 à 4) avec Bcrypt

On y voit notamment qu'un logiciel comme John the Ripper est directement affecté à partir des niveaux 3 et 4, contrairement à PCFG et OMEN, dont les performances ne semblent pas directement liées au niveau de difficulté de `zxcvbn` (sur les bases RockYou et Linkedin, c'est moins vrai). Notons que le temps d'attaque sur la figure 5.8 s'arrête un peu avant 4h car la suite n'apporte pas d'informations supplémentaires. L'incorporation d'énumérateurs comme PCFG ou OMEN dans un logiciel spécialisé réduirait fortement l'intérêt de `zxcvbn`, notamment si on considère comme facilité d'utilisation du système que le pourcentage de mots de passe de la base restant correspond à la probabilité d'être accepté par un utilisateur. Cette partie sur le comportement des énumérateurs est en cours de publication.

Conclusion. Un des principal problème dans la sécurité des mots de passe est l'absence de standardisation des énumérateurs. La dernière version du NIST SP 800-63-3 qui date de 2019 se contente d'une politique vague, mais il n'y a ni algorithme ni implémentation de référence pour les énumérateurs. Leur comportement diffère pourtant énormément selon la base d'apprentissage. Hashcat a d'ailleurs récemment ajouté une interface logicielle permettant d'attacher des énumérateurs avancés.

Perspectives. Une suite directe à ce travail sur la sécurité des mots de passe concerne les règles de réécritures (typiquement a $\rightarrow$ @). Celles-ci pourraient être appliquées à chaque candidat généré par l'énumérateur avant la génération du candidat suivant. Il serait alors intéressant d'analyser le comportement des énumérateurs dans ce contexte car de telles règles permettraient de sortir en partie de la forte dépendance de la base d'apprentissage et de lisser le temps d'énumération. Une autre perspective, qui est en fait un travail en cours, consiste à analyser le comportement des énumérateurs en étudiant l'évolution des structures de données qu'ils utilisent lors de l'énumération des mots de passe (taille de la file de priorité dans le cas de PCFG, nombre de noeuds dans un arbre dans le cas d'OMEN).

Conclusion

Plusieurs travaux touchant à la cryptographie et à la sécurité informatique ont été présentés dans ce manuscrit. Certains d'entre eux ont été réalisés seul ou pendant ma thèse avec mon directeur Philippe Langevin, comme ceux présentés au chapitre 1 sur la correction du biais dans le cadre de la génération de données aléatoires et sur la construction d'un engagement flou avec un code de Kerdock ou encore sur l'étude du comportement de l'algorithme de Biohashing du début du chapitre 3.

D'autres travaux proviennent du co-encadrement des thèses d'Aude Plateaux, de Thomas Gougeon et de Mathieu Valois sur la protection des données de e-santé décrit au chapitre 2, sur la forensique des dumps mémoires du chapitre 4 et sur la protection des mots de passe du chapitre 5, qui ont été dirigées par Christophe Rosenberger, Kumar Murty, Gildas Avoine et Jean-Marie Le Bars.

Enfin, plusieurs travaux sont le fruit d'une collaboration avec d'autres chercheurs, comme ceux de la seconde partie du chapitre 2 dans le cadre de l'ANR Lyrics sur la protection des données personnelles, ceux sur les attaques sur les transformations de données biométriques avec Christophe Rosenberger et Estelle Cherrier pour l'algorithme biohashing, et avec Kevin Atighehchi, Loubna Ghammam et Koray Karabina pour la fin du chapitre 3. L'attaque sur le CAPTCHA de la partie 5 entre aussi dans ce cadre car il a été réalisé avec Thomas Gougeon, sans lien avec sa thèse.

Il y a plusieurs perspectives à ces travaux, notamment dans le cas de la sécurité des transformations de données biométriques. Je vais encadrer une nouvelle thèse en septembre 2019 sur ce domaine car il est à la fois important au niveau de l'actualité (RGPD) mais aussi car il reste encore trop peu étudié. Dans le même domaine, mais avec des outils complètement différents, je souhaite aussi étudier l'adéquation des réseaux euclidiens pour la protection des données biométriques. J'ai eu l'occasion de découvrir la cryptographie homomorphe basée sur les réseaux dans le cadre du FUI ATELYN dont j'étais le responsable pour le GREYC. Ces objets mathématiques pourraient permettre la mise en place de systèmes biométriques où la sécurité est basée sur des problèmes difficiles. D'autres prolongements aux travaux décrits dans ce manuscrit sont aussi possibles, comme la création de tests statistiques à la volée pour la génération de données aléatoires ou un possible regroupement entre attaques sur des mots de passe ciblés et forensique. De manière générale, je suis attaché à conserver un certain équilibre entre recherche fondamentale et applications.

Bibliographie

[1] M. Abdalla, S. Belaid, D. Pointcheval, S. Ruhault, and D. Vergnaud. Robust pseudo-random number generators with input secure against side-channel attacks. In *International Conference on Applied Cryptography and Network Security (ACNS)*, volume 9092 of *LNCS*, pages 635–654. Springer, 2015.

[2] P. M. Alcover, A. Guillamon, and M. del Carmen Ruiz. A new randomness test for bit sequences. *Informatica*, 24(3) :339–356, 2013.

[3] N. Alon, O. Goldreich, J. Hastad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms*, 3(3) :289–304, 1992.

[4] Antminer. Bitmain, 2019. https ://www.asicminervalue.com/miners/bitmain.

[5] G. Arfaoui. *Design of privacy preserving cryptographic protocols for mobile contactless services*. PhD thesis, Université d'Orléans, 2015.

[6] G. Arfaoui, G. Dabosville, S. Gambs, P. Lacharme, and J.-F. Lalande. A privacy-preserving NFC mobile pass for transport systems. *ICST Transactions on Mobile Communications Applications*, 2(5), 2014.

[7] G. Arfaoui, S. Gambs, P. Lacharme, J.-F. Lalande, R. Lescuyer, and J. C. Paillès. A privacy-preserving contactless transport service for NFC smartphones. In *Mobile Computing, Applications, and Services (MobiCASE)*, volume 130 of *LNICST*, pages 282–285. Springer, 2013.

[8] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information System Security (TISSEC)*, 9(1) :1–30, 2006.

[9] K. Atighehchi, L. Ghammam, K. Karabina, and P. Lacharme. Cryptanalysis of two cancelable biometric schemes based on index-of-max hashing, 2020. preprint.

[10] J.P. Aumasson. Password Hashing Competition and our recommendation for hashing passwords : Argon2. https ://password-hashing.net/.

[11] T. Baignères, P. Junod, and S. Vaudenay. How far can we go beyond linear cryptanalysis ? In *ASIACRYPT*, volume 3329 of *LNCS*, pages 432–450. Springer, 2004.

[12] Y. Bass, J.-B. Jorand, M. Valois, and P. Lacharme. C++ implementation of pack with additionnal features, 2018. https ://git.unicaen.fr/passwords/cpack.

[13] A. Beimel. Secret-sharing schemes : a survey. In *International Workshop on Coding and Cryptology (IWCC)*, pages 11–46, 2011.

[14] R. Belguechi. *Sécurité des systèmes biométriques :révocabilité et protection de la vie privée*. PhD thesis, Ecole Nationale Supérieure d'Informatique, ESI, Algérie, 2015.

[15] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Crypto*, volume 263 of *LNCS*, pages 213–222. Springer, 1987.

[16] T.D. Bending and D. Fon-Der-Flaass. Crooked functions, bent functions, and distance regular graphs. *Electronical Journal of Combinatorics*, 5(1), 1998.

[17] J. Bierbrauer, K. Gopalakrishnan, and D. Stinson. Orthogonal arrays, resilient functions, error-correcting codes and linear programming bounds. *SIAM Journal on Discrete Mathematics*, 9(3) :424–452, 1996.

[18] A. Biryukov, D. Dinu, and D. Khovratovich. Argon2 : new generation of memory-hard functions for password hashing and other applications. In *IEEE European Symp. on Security and Privacy (EuroS&P)*, pages 292–302, 2016.

[19] M. Bishop and D. V. Klein. Improving system security via proactive password checking. *Computers and Security*, 14(3) :233–249, 1995.

[20] G.R. Blakley. Safeguarding cryptographic keys. In *National Computer Conference (NCC)*, pages 313–317. American Federation of Information Processing Societies (AFIPS), 1979.

[21] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *ACM Confernce on Computer and Communications Security (CCS)*, pages 132–145, 2004.

[22] J. Bringer, H. Chabanne, G. D. Cohen, B. Kindarji, and G. Zémor. Optimal iris fuzzy sketches. In *IEEE International Conference on Biometrics : Theory, Applications, and Systems (BTAS)*, pages 1–6, 2007.

[23] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell. The end is nigh : Generic solving of text-based captchas. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2014.

[24] E. Bursztein, R. Beauxis, H. S. Paskov, D. Perito, C. Fabry, and J. C. Mitchell. The failure of noise-based non-continuous audio captchas. In *IEEE Symposium on Security and Privacy (S&P)*, pages 19–31, 2011.

[25] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, volume 3152 of *LNCS*, pages 56–72. Springer, 2004.

[26] S. Canard, B. Schoenmakers, M. Stam, and J. Traoré. List signature schemes. *Discrete Applied Mathematics*, 154(2) :189–201, 2006.

[27] A. Canteaut, C. Carlet, P. Charpin, and C. Fontaine. On cryptographic properties of the cosets of R(1, m). *IEEE Trans. on Info. Theory*, 47(4), 2001.

[28] C. Carlet. *Boolean Functions for Cryptography and Error-Correcting Codes.* Cambridge University Press, 2010.

[29] C. Carlet. *Vectorial Boolean Functions for Cryptography.* Cambridge University Press, 2010.

[30] D. Chaum and E. Van Heyst. Group signatures. In *EUROCRYPT*, volume 547 of *LNCS*, pages 257–265. Springer, 2001.

[31] B. Chor, J. Friedman, 0. Goldreich, J. Hastad, S. Rudich, and R. Smolensky. The bit extraction problem or t-resilient functions. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 396–407. IEEE Computer Society, 1985.

[32] B. Chor and E. Kushilevitz. Secret sharing over infinite domains. *Journal of Cryptology*, 6 :87–95, 1993.

[33] CNIL. La loi informatique et libertés, 2019. https ://www.cnil.fr/fr/la-loi-informatique-et-libertes.

[34] T. Connie, A. J. B. Teoh, M. Goh, and D. Ngo. Palmhashing : A novel approach to cancelable biometrics. *Info. Processing Letters*, 93(1) :1–5, 2004.

[35] M. Conti, C. Guarisco, and R. Spolaor. Captchastar ! a novel captcha based on interactive shape discovery. In *Applied Cryptography and Network Security (ACNS)*, volume 9696 of *LNCS*, pages 611–628. Springer, 2016.

[36] M. Conti, C. Guarisco, and R. Spolaor. Captchastar demo, 2016. http ://captchastar.math.unipd.it/demo.php.

[37] B. De Decker, M. Layouni, H. Vangheluwe, and K. Verslyp. A privacy-preserving ehealth protocolcompliant with the belgian healthcare system. In *Euro PKI*, volume 5057 of *LNCS*, pages 118–133. Springer, 2008.

[38] P. Delsarte. Four fundamental parameters of a code and their combinatorial significance. *Information and Control*, 23(5) :407–438, 1973.

[39] Y. Desmedt and Y. Frankel. Threshold crypto-systems. In *CRYPTO*, volume 435 of *LNCS*, pages 307–315. Springer, 1989.

[40] T. Van Deursen, S. Mauw, and S. Radomirovic. mcarve : Carving attributed dump sets. In *USENIX Security Symposium*, pages 107–121, 2011.

[41] J. Devigne. *Protocoles de re-chiffrement pour le stockage de données*. PhD thesis, Université de Caen Basse Normandie, 2013.

[42] M. Dichtl. Bad and good ways of post-processing biased physical random numbers. In *Fast Software Encryption (FSE)*, volume 4593 of *LNCS*, page 127–152. Springer, 2007.

[43] J. Dillon. *Elementary Hadamard Difference sets*. PhD thesis, University of Maryland, 1974.

[44] X. Dong, Z. Jin, and A. B. J. Teoh. A genetic algorithm enabled similarity-based attack on cancellable biometrics, 2019.

[45] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane. Omen implementation. https ://github.com/RUB-SysSec/OMEN.

[46] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane. Omen : Faster password guessing using an ordered markov enumerator. In *International Symposium on Engineering Secure Software and Systems*, 2015.

[47] D. C. Feldmeier and P. R. Karn. UNIX password security - ten years later. In *CRYPTO*, volume 403 of *LNCS*, pages 44–63. Springer, 1989.

[48] Y. C. Feng, M.-H. Lim, and P. C. Yuen. Masquerade attack on transform-based binary-template protection based on perceptron learning. *Pattern recognition*, 47(9) :3019–3033, 2014.

[49] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1) :119–139, 1997.

[50] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5) :771–780, 1999.

[51] J. Galbally, I. Coisel, and I. Sanchez. A new multimodal approach for password strength estimation-part i : Theory and algorithms. *IEEE Transactions on Information Forensics and Security*, 12(12) :2829–2844, 2017.

[52] J. Galbally, I. Coisel, and I. Sanchez. A new multimodal approach for password strength estimation-part ii : Experimental evaluation. *IEEE Transactions on Information Forensics and Security*, 12(12) :2845–2860, 2017.

[53] J. Galbally, A. Ross, M. Gomez-Barrero, J. Fierrez, and J. Ortega-Garcia. Iris image reconstruction from binary templates : An efficient probabilistic approach based on genetic algorithms. *Computer Vision and Image Understanding*, 117(10) :1512–1525, 2013.

[54] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156 :3113–3121, 2008.

[55] H. Gao, J. Yan, F. Cao, Z. Zhang, L. Lei, M. Tang, P. Zhang, X. Zhou, X. Wang, and J. Li. A simple generic attack on text captchas. In *Network and Distributed System Security Symposium, (NDSS)*. The Internet Society, 2016.

[56] S. L. Garfinkel. Digital forensics research : The next 10 years. *digital investigation*, 7 :64–73, 2010.

[57] S. L. Garfinkel. Digital media triage with bulk data analysis and bulk extractor. *Computers & Security*, 32 :6–72, 2013.

[58] I. Goldberg and D. Wagner. Randomness and the Netscape browser. *Dr.Dobbs Journal*, 1996.

[59] P. Golle. Machine learning attacks against the Asirra CAPTCHA. In *ACM Conf. on Computer and communications security (CCS)*, pages 535–542, 2008.

[60] M. Gomez-Barrero, J. Galbally, C. Rathgeb, and C. Busch. General framework to evaluate unlinkability in biometric template protection systems. *IEEE Transactions on Information Forensics and Security*, 13(6) :1406–1420, 2018.

[61] J. Gosney. 8x Nvidia GTX 1080 Ti Hashcat Benchmarks. Accédé en 2019.

[62] T. Gougeon. *Analyse forensique de la mémoire des cartes à puce*. PhD thesis, Université de Caen Basse Normandie, 2017.

[63] T. Gougeon, M. Barbier, P. Lacharme, G. Avoine, and C. Rosenberger. Memory carving in embedded devices : Separate the wheat from the chaff. In *International Conference on Applied Cryptography and Network Security (ACNS)*, volume 9696 of *LNCS*, pages 592–608. Springer, 2016.

[64] T. Gougeon, M. Barbier, P. Lacharme, G. Avoine, and C. Rosenberger. Memory carving can finally unveil your embedded personal data. In *Int. Conf. on Availability, Reliability and Security (ARES)*, pages 1–18, 2017.

[65] T. Gougeon, M. Barbier, P. Lacharme, G. Avoine, and C. Rosenberger. Retrieving dates in smart card dumps is as hard as finding a needle in a haystack. In *IEEE Workshop on Info. Forensics and Security (WIFS)*, pages 1–6, 2017.

[66] T. Gougeon and P. Lacharme. How to break CaptchaStar. In *International Conf. on Info. Systems Security and Privacy (ICISSP)*, pages 41–51, 2018.

[67] The Guardian. Us government hack stole fingerprints of 5.6 million federal employees, 2015.

[68] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember : cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5) :91–98, 2009.

[69] F. Hao, R. Anderson, and J. Daugman. Combining crypto with biometrics effectively. *IEEE Transactions on Computers*, 55(9) :1081–1088, 2006.

[70] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs : Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, pages 205–220, 2012.

[71] J. H. Holland. *Adaptation in Natural and Artificial Systems*. 1975.

[72] V.C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (ABAC) definition and considerations, 2014. NIST Special Publication 800-162.

[73] Troy Hunt. Have i been pwned, 2017. https ://haveibeenpwned.com.

[74] T. Ignatenko and F.M.J. Willems. Information leakage in fuzzy commitment schemes. *IEEE Transactions on Information Forensics and Security*, 5(2) :337–348, 2010.

[75] T. Ignatenko and F.M.J. Willems. Fundamental limits for privacy-preserving biometric identification systems that support authentication. *IEEE Transactions on Information Theory*, 61(10) :5583–5594, 2015.

[76] M. Inuma and A. Otsuka. Relations among security metrics for template protection algorithms. In *IEEE International Conference on Biometrics : Theory, Applications, and Systems (BTAS)*, 2013.

[77] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *IEEE Global Communications Conference (GLOBECOM)*, pages 99–102, 1987.

[78] Z. Jin, J. Y. Hwang, S. Kim Y.-L. Lai, and A. B. J. Teoh. Ranking-based locality sensitive hashing-enabled cancelable biometrics : Index-of-max hashing. *IEEE Transactions on Information Forensics and Security*, 13(2), 2018.

[79] Z. Jin, M.-H. Lim, A. B. J. Teoh, B.-M. Goi, and Y.H. Tay. Generating fixed-length representation from minutiae using kernel methods for fingerprint authentication. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, 46(10) :1415–1428, 2016.

[80] A. R. Hammons Jr., P. V. Kumar, A. R. Calderbank, N. J. A. Sloane, and P. Solé. The z_4-linearity of Kerdock, Preparata, Goethals, and related codes. *IEEE Transactions on Information Theory*, 40(2) :301–319, 1994.

[81] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *ACM Conference on Computer and Communication Security (CCS)*, pages 28–36, 1999.

[82] P. Kacherginsky. Password analysis and cracking kit, 2014. https ://github.com/iphelix/pack.

[83] W.M. Kantor. An exponential number of generalized Kerdock codes. *Information and Control*, 53 :74–80, 1982.

[84] E. J. C. Kelkboom, J. Breebaart, T. A. M. Kevenaar, I. Buhan, and R. N. Veldhuis. Preventing the decodability attack based cross-matching in a fuzzy commitment scheme. *IEEE Transactions on Information Forensics and Security*, 6(1) :107–121, 2011.

[85] P.G. Kelley, S. Komanduri, M.L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L.F. Cranor, and J. Lopez. Guess again (and again and again) : Measuring password strength by simulating password-cracking algorithms. In *IEEE Symposium on Security and Privacy (S&P)*, pages 523–537, 2012.

[86] A. M. Kerdock. A class of low rate non-linear codes. *Information and Control*, 20 :182–187, 1972.

[87] W. Killmann and W. Schindler. A proposal for : Functionality classes for random number generators, 2011. BSI, version 2.0.

[88] A. Kumar, R. Meka, and A. Sahai. Leakage-resilient secret sharing. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 25, 2018.

[89] P. Lacharme. Efficient construction of resilient functions for a random generator. In *Yet Another Conference in Cryptography (YACC)*, 2006.

[90] P. Lacharme. *Générateur vraiment aléatoire dans un composant sécurisé*. PhD thesis, Université de Toulon, 2007.

[91] P. Lacharme. Post-processing functions for a biased physical random number generator. In *Fast Software Encryption (FSE)*, volume 5086 of *LNCS*, pages 334–342. Springer, 2008.

[92] P. Lacharme. Analysis and construction of correctors. *IEEE Transactions on Information Theory*, 55(10) :4742–4748, 2009.

[93] P. Lacharme. Revisiting the accuracy of the biohashing algorithm on fingerprints. *IET Biometrics*, 2(3) :130–133, 2013.

[94] P. Lacharme, E. Cherrier, and C. Rosenberger. Reconstruction attack on biohashing. In *International Conference on Security and Cryptography (SECRYPT)*, pages 363–370. SciTePress, 2013.

[95] P. Lacharme, A. Röck, V. Strubel, and M. Videau. The linux pseudorandom number generator revisited, 2012. IACR Cryptology ePrint Archive.

[96] R. Lescuyer. *Outils cryptographiques pour les accréditations anonymes*. PhD thesis, Université Paris VII Diderot, 2012.

[97] J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. In *IEEE Symposium on Security and Privacy (S & P)*, pages 689–704, 2014.

[98] D. Maio, D. Maltoni, R. Cappelli, J.L. Wayman, and A. K. Jain. FVC2002 : Second fingerprint verification competition. In *International Conference on Pattern Recognition (ICPR)*, pages 811–814. IEEE Computer Society, 2002.

[99] A. Meneghetti, M. Sala, and A. Tomasi. A weight-distribution bound for entropy extractors using linear binary codes, 2014. CoRR abs/1405.2820.

[100] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, 1998.

[101] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing ? *Advances in Neural Information Processing Systems*, (6), 1994.

[102] R. Morris and K. Thompson. Password security : A case history. *Communication to the ACM*, 22 :594–597, 1979.

[103] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re : Captchas-understanding captcha-solving services in an economic context. In *USENIX Security Symposium*, volume 10, pages 435–462, 2010.

[104] A. Nagar, K. Nandakumar, and A. K. Jain. Biometric template transformation : A security analysis. In *SPIE, Electronic Imaging, Media Forensics and Security*, 2010.

[105] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *ACM conference on Computer and communications security (CCS)*, pages 364–372, 2005.

[106] J. Von Neumann. Various techniques used in connection with random digits. *Applied Math. series*, 12 :36–38, 1951.

[107] A. W. Nordstrom and J. P. Robinson. An optimum nonlinear code. *Information and Control*, 11 :613–616, 1967.

[108] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO*, volume 2729 of *LNCS*, pages 617–630. Springer, 2003.

[109] P. Oechslin. Les compromis temps-mémoire et leur utilisation pour casser les mots de passe windows. In *Symposium sur la sécurité des technologies de l'information et des communications (SSTIC)*, 2004.

[110] OpenWall. John the ripper, 2017. http ://www.openwall.com/john.

[111] H. Park, Y. Yeom, and J.-S. Kang. A lightweight BCH code corrector of trng with measurable dependence. *Security and Communication Networks*, 2019.

[112] T. Pederson. A threshold crypto-system without a trusted dealer. In *EUROCRYPT*, volume 547 of *LNCS*, pages 522–526. Springer, 1991.

[113] C. Percival. Stronger key derivation via sequential memory-hard function. In *BSD Conference*, 2009.

[114] A. Plateaux. *Solutions opérationnelles d'une transaction électronique sécurisée et respectueuse de la vie privée.* PhD thesis, Université de Caen Basse Normandie, 2013.

[115] A. Plateaux and P. Lacharme. Organisation d'une architecture de santé respectueuse de la vie privée. In *Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information (SARSSI)*, 2012.

[116] A. Plateaux, P. Lacharme, V. Coquet, S. Vernois, G. Frey, and A. Gouriou. Protection de la vie privée dans les modèles de paiement en ligne. In *Conf sur la Sécurité des Architectures Réseaux et des Systèmes d'Info (SARSSI)*, 2013.

[117] A. Plateaux, P. Lacharme, V. Coquet, S. Vernois, K. Murty, and C. Rosenberger. An e-payment architecture ensuring a high level of privacy protection. In *Int. Conf. on Security and Privacy in Com. Networks (SecureComm)*, 2013.

[118] A. Plateaux, P. Lacharme, C. Rosenberger, and K. Murty. A contactless e-health information system with privacy. In *International Wireless Communications & Mobile Computing Conference (IWCMC)*, 2013.

[119] A. Plateaux, P. Lacharme, S. Vernois, V. Coquet, and C. Rosenberger. A comparative study of card-not-present e-commerce architectures with card schemes : What about privacy ? *J. Inf. Sec. Appl.*, 40 :103–110, 2018.

[120] R. Poisel and S. Tjoa. A comprehensive literature review of file carving. In *Int Conf on Availability, Reliability and Security (ARES)*, pages 475–484, 2013.

[121] N. Provos and D. Mazieres. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91, 1999.

[122] I. S. Reed. A class of multiple-error-correcting codes and the decoding scheme, 1954. Technical Report N 44.

[123] A. Rozsa, A. E. Glock, and T. E. Boult. Genetic algorithm attack on minutiae-based fingerprint authentication and protected template fingerprint systems. In *IEEE Computer Vision and Pattern Recognition Workshops (CVPR)*, 2015.

[124] S. Ruhault. *Security analysis for pseudo-random number generators.* PhD thesis, Ecole normale supérieure - ENS PARIS, 2015.

[125] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical Report Special Publication 800-22, NIST, 2010.

[126] R.E. Schapire. The strength of weak learnability. *Machine Learning*, 5 :197–227, 1990.

[127] A. Shamir. How to share a secret. *Com. of the ACM*, 22 :612–613, 1979.

[128] A. Shamir and N. Van Someren. Playing "hide and seek" with stored keys. In *Financial Crypto.*, volume 1648 of *LNCS*, pages 118–124. Springer, 1999.

[129] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. on Info. T.*, 30(5) :776–780, 1984.

[130] K. Simoens, P. Tuyls, and B. Preneel. Privacy weaknesses in biometric sketches. In *IEEE Symp on Security and Privacy (S&P)*, pages 188–203, 2009.

[131] S. Sivakorn, I. Polakis, and A. D. Keromytis. I am robot : (deep) learning to break semantic image CAPTCHAs. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 388–403, 2016.

[132] E. H. Spafford. The internet worm incident. In *European Software Engineering Conference (ESEC)*, pages 446–468, 1989.

[133] J. Steube. Hashcat implementation. https ://github.com/hashcat/hashcat.

[134] D. Stinson. An explication of secret sharing schemes. *Designs, Codes and Cryptography*, 2 :357–390, 1992.

[135] D. Stinson and J.L. Massey. An infinite class of counterexamples to a conjecture concerning nonlinear resilient functions. *Journal of Cryptololy*, 8, 1995.

[136] F. Sulak. A new statistical randomness test : Saturation point test. *International Journal of Information Security Science*, 2(3) :81–85, 2013.

[137] B. Tams. Decodability attack against the fuzzy commitment scheme with public feature transforms, 2014. CoRR abs/1406.1154.

[138] A. B. J. Teoh, T. Connie, D. Ngo, and C. Ling. Remarks on biohash and its mathematical foundation. *Information Processing Letters*, 100 :145–150, 2008.

[139] A. Tomasi, A. Meneghetti, and M. Sala. Code generator matrices as RNG conditioners. *Finite Fields and Their Applications*, 47 :46–63, 2017.

[140] B. Topcu, C. Karabat, M. Azadmanesh, and H. Erdogan. Practical security and privacy attacks against biometric hashing using sparse recovery. *EURASIP Journal on Advances in Signal Processing*, 2016.

[141] A. Umar, I. Gurulian, K. Mayes, and K. Markantonakis. Tokenisation blacklisting using linkable group signatures. In *International Conference on Security and Privacy in Communication Systems (SecureComm)*, pages 182–198, 2016.

[142] B. Ur, S.M. Segreti, L. Bauer, N. Christin, L.-F. Cranor, S. Komanduri, D. Kurilova, M.L. Mazurekand W. Melicher, and R. Shay. Measuring real-world accuracies and biases in modeling password guessability. In *USENIX Security Symposium*, pages 463–481, 2015.

[143] M. Valois, P. Lacharme, and J.-M. Le Bars. Performance of password guessing enumerators under cracking conditions. In *IFIP Information Security Conference & Privacy Conference (SEC)*, pages 67–80, 2019.

[144] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA : Using hard AI problems for security. In *EUROCRYPT*, volume 2656 of *LNCS*, pages 294–311. Springer, 2003.

[145] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *IEEE Symposium on Security and Privacy (S & P)*, pages 391–405, 2009.

[146] D. L. Wheeler. Zxcvbn : Low-budget password strength estimation. In *USENIX Security Symposium*, 2016.

[147] F.J. Mac Williams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library, 1977.

[148] G. Xiao and J.L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Trans. on Information Theory*, 34 :569–571, 1988.

[149] J. Yan and A. S. El Ahmad. A low-cost attack on a microsoft CAPTCHA. In *ACM Conf. on Computer and Com. Security (CCS)*, pages 543–554, 2007.

[150] H. Zhou and J. Bruck. Linear extractors for extracting randomness from noisy sources. In *Int. Symp. on Information Theory (ISIT)*, pages 1738–1742, 2011.

Résumé

La sécurité informatique est une discipline qui a évolué parallèlement aux autres domaines de l'informatique. Ainsi, la cryptographie moderne, qui a commencé dans les années 70, a proposé de nouvelles solutions pour la protection de la vie privée à partir des années 80, ou plus récemment pour la protection des données biométriques avec l'arrivée de la cryptographie homomorphe. L'utilisation de ces primitives cryptographiques ne résout toutefois pas tous les problèmes de sécurité et de nombreuses applications ou attaques emploient des techniques venant d'autres disciplines.

Les travaux décrits dans ce manuscrit sont liés à des domaines très différents, comme la cryptographie, la protection de la vie privée, la forensique, la protection des données biométriques et l'authentification des personnes. Plusieurs aspects y sont étudiés comme la génération de données aléatoires, l'analyse automatique de mémoires de cartes à puce et la sécurité des transformations non inversibles de données biométriques. La plupart de ce travaux ont donné lieu à la fois à une analyse théorique et à une mise en oeuvre sur des cas d'usage précis.

Resume

Computer security is a discipline that has evolved in parallel with other topics of computer science. Modern cryptography, which began in the middle of the 1970s, proposed new solutions for privacy in the 1980s, or more recently for biometric protection with homomorphic encryption. However, these cryptographic primitives will not resolve all security problems and numerous applications or attacks are based on techniques from various domains.

This thesis is interested by several topics, that are very different, as cryptography, privacy, forensics, biometric data protection and authentication. These works typically expose random data generation, automatical research of fata in memory dumps or practical attacks on non invertible transformation of biometric data. Most of these works provide both a theorical analysis and a pratical implementation.

Mots-clés : cryptographie appliquée, sécurité informatique

ENSICAEN - UNICAEN - CNRS - GREYC UMR 6072, F-14050 Caen, France