Thèse de doctorat de l'université Pierre et Marie Curie
Spécialité Informatique

EDITE de Paris

présentée par
Stéphane Jacob

sous la direction de
Anne Canteaut

pour obtenir le grade de
Docteur de l'université Pierre et Marie Curie

Protection cryptographique des bases de données : conception et cryptanalyse

Soutenance le 08/03/2012

Jury :

Anne Canteaut	INRIA Paris-Rocquencourt	Directrice de thèse
Luc Bouganim	INRIA Paris-Rocquencourt	Rapporteur
Pierre Loidreau	DGA MI et Université de Rennes 1	Rapporteur
Jean-Claude Bajard	Université Pierre et Marie Curie	Examinateur
Matthieu Finiasz	CryptoExperts	Examinateur
Pierre-Alain Fouque	École normale supérieure	Examinateur
François Morain	École polytechnique	Examinateur

Stéphane Jacob

Protection cryptographique des bases de données : conception et cryptanalyse

Équipe-Projet SECRET
INRIA Paris-Rocquencourt
Domaine de Voluceau, BP 105
78153 Le Chesnay Cedex
France

Remerciements

Je tiens à remercier toutes les personnes qui m'ont accompagnées au long de cette thèse.

Tout d'abord, je souhaite remercier Anne Canteaut, ma directrice de thèse, pour avoir su me conseiller tout au long de cette thèse, pour la qualité de sa relecture, pour m'avoir supporté pendant ces trois années et aussi pour avoir réussit à convaincre son équipe de reperdre si rapidement après la fessée infligée à Chambéry.

Je remercie ensuite Pierre Loidreau et Luc Bouganim d'avoir acceptés d'être mes rapporteurs. Luc a en plus le mérite d'avoir toujours accepté dans la bonne humeur qu'Anne et moi lui apportions la contradiction.

Je tiens aussi à remercier les autres membres de mon jury qui ont tous apportés à leur façon à cette soutenance. Jean-Claude Bajard a gentiment trouvé une salle pour la soutenance. François Morain m'a fait découvrir la cryptographie et m'y a donné goût. Pierre-Alain Fouque m'a aussi enseigné la cryptographie. Finalement, Matthieu Finiasz m'a énormément apporté tout au long de cette thèse. Ses intérêts pour les Gentoo, les Nikon, et plus généralement pour l'informatique, la photographie, la cryptographie, la musique... auront été la source de discussions sans fin au cours de ces trois années.

Je tiens aussi à remercier l'intégralité des personnes ayant fréquenté l'Équipe-projet SECRET durant la durée de ma thèse. Tout d'abord les permanents : Nicolas qui, après m'avoir offert 5 mois très agréables mois chez Matthew Parker (à qui je dois l'envie de faire une thèse), m'a invité au projet ; JP pour ses matchs de tennis toujours très disputés, et son support moral, Daniel pour sa curiosité informatique sans limite et Pascale pour ses opinions politiques. Mais le projet ne serait pas ce qu'il est sans ses stagiaires, thésards, post-docs et invités, que je ne citerai point, de peur d'en oublier. Je souhaite juste remercier les trois autres membres du bureau 12 de m'avoir si bien accueilli dans ce bureau à deux Mat(t)hieu et deux Stéphane. Les Los Pumas m'auront aussi grandement influencé.

Ensuite, ces trois années n'auraient pas été possibles sans le soutient de mes amis des autres projets, Mathieu et Pilki, à qui s'est ajouté ensuite Jonathan qui aurait même pu se retrouver dans mon bureau si les vils desseins de Pilki eurent pris forme.

Mais je n'ai pas passé tout mon temps à l'INRIA ces trois années. Les restaurants japonais avec Pierrot, les foots du dimanche matin avec Constantin, les concerts avec Cyril et les dîners improvisés chez Pitou et Nelly auront été des moments marquants de cette période de ma vie.

Finalement, ma famille m'a grandement aidé à passer ces trois années, tout particulièrement Chrysanthi qui me supporte quotidiennement désormais depuis 2 ans et un jour.

b13f65276d7a463eef31e41fb171558e39a1694565b2819e8dc966bb8e2e7a1d
bdb67102afa605cf124643311d6c35a63fcbb0dfcbee37f23d7d63ab7c2

Introduction

Depuis la loi du 4 mars 2002 relative aux droits des malades et à la qualité du système de santé[1], dite loi « Kouchner », le Code de la santé publique autorise le stockage externalisé de données de santé :

> « Les professionnels de santé ou les établissements de santé ou la personne concernée peuvent déposer des données de santé à caractère personnel, recueillies ou produites à l'occasion des activités de prévention, de diagnostic ou de soins, auprès de personnes physiques ou morales agréées à cet effet.[2] »

Néanmoins, cet hébergement est encadré juridiquement de façon précise, et premièrement par cette même loi inaugurale. Par exemple, il ne peut avoir lieu qu'avec le consentement exprès de la personne concernée :

> « Cet hébergement de données, quel qu'en soit le support, papier ou informatique, ne peut avoir lieu qu'avec le consentement exprès de la personne concernée.[2] »

Gérer et/ou héberger ce type de données sensibles est complexe car encadré par de strictes règles juridiques et limité par les normes techniques existantes. En particulier, concernant les problèmes de sécurité, les solutions techniques actuelles ne sont pas toujours en adéquation avec les dispositifs légaux les encadrant.

Peu après le début de ma thèse, j'ai commencé à participer aux travaux du projet DEMOTIS[3,4]. Ce projet[5] réunit « informaticiens et juristes qui expérimente[nt] de nouvelles méthodes de conception multidisciplinaire pour les grands systèmes d'information pour lesquels [existent] des contraintes juridiques, sociales et techniques importantes, comme les systèmes d'information de santé. » Les partenaires, qui viennent de divers horizons, à la fois

[1] JORF n° 54 du 5 mars 2002, page 4118, texte n° 1
[2] Article L.1111-8, alinéa 1 du Code de la santé publique
[3] DEMOTIS : Définir, Évaluer et MOdéliser les Technologies de l'Information de Santé
[4] http://www.demotis.org/
[5] Contrat ANR-08-SEGI-007

du monde universitaire et de celui de l'entreprise, sont les suivants. Sopinspace[6] développe des applications collaboratives dans le but d'aider le débat public, la démocratie participative et la coopération. Les équipes-projets SECRET et CACAO de l'INRIA travaillent essentiellement à la conception et à l'analyse de la sécurité d'algorithmes cryptographiques. C'est en tant que doctorant dans l'équipe-projet SECRET que j'ai participé au projet DEMOTIS. L'équipe-projet SMIS a, elle, pour premier objectif de concevoir et de valider de nouvelles techniques de gestion de données embarquées compatibles avec les contraintes matérielles imposées par ces types de calculateurs. Finalement, le CECOJI (Centre d'Études sur la COopération Juridique Internationale) est une unité mixte de recherche associant l'Université de Poitiers et le CNRS. Il est constitué de plusieurs équipes, dont celle « Normativité et nouvelles technologies » menée par Stéphanie Lacour, qui travaille sur les normes qui s'appliquent ou s'appliqueront aux nouvelles technologies, sur leurs modes de régulation et sur les domaines juridiques qui les touchent.

Le projet DEMOTIS étudie principalement deux types de systèmes : prioritairement l'infrastructure du dossier médical personnalisé (DMP) et secondairement celles des dossiers des réseaux de soins liés à certaines affections, notamment au SIDA. Ces choix sont motivés par l'importance de leurs enjeux, à la fois économiques et sociaux, ainsi que par leur pertinence vis-à-vis de l'actualité politique et technologique.

DEMOTIS comporte deux volets interdépendants, juridique avec le droit de la santé, des données personnelles et de la propriété intellectuelle, et informatique avec la sécurité des bases de données et en particulier les techniques cryptographiques à utiliser pour les protéger. Ces volets ont été étudiés conjointement via un corpus de textes juridiques et normatifs qui a fait l'objet de lectures croisées de juristes et d'informaticiens. Notre but était de confronter les textes juridiques à l'état de l'art en l'informatique et d'aboutir à une identification claire des besoins attendus, des limitations que l'informatique impose actuellement, et des pistes émergentes dans les domaines de la cryptographie et des bases de données pertinentes dans ce cadre.

Ma thèse s'inscrit donc dans ce contexte, et explore l'aspect cryptographique de la sécurisation des systèmes d'hébergement et de gestion de données sensibles. Je me suis principalement intéressé au problème du chiffrement des bases de données qui est central dans cette problématique.

Ainsi, les problèmes mis en évidence par le projet DEMOTIS dans le domaine de la cryptographie dépassent largement le simple cadre du dossier médical personnalisé (DMP). Le chiffrement des bases de données est en effet un réel problème d'actualité. Les services informatiques tendent de plus en plus à être externalisés, de plus en plus de données sont « *on the cloud* », c'est-à-dire hébergées, administrées et gérées par des personnes qui

[6] http://www.sopinspace.com/

ne sont pas leurs propriétaires. Les intérêts d'une telle externalisation sont multiples. En effet, le coût de développement, de mise en œuvre et de maintenance d'applications pour gérer des données est vite prohibitif pour une qualité de service rarement exceptionnelle. Pour cela, il est souvent préférable de partager les coûts entre plusieurs entités, ou de faire appel à un tiers spécialisé. Par exemple, les webmails de type Gmail ont souvent une meilleure qualité de service, de meilleures fonctionnalités et sont aussi plus robustes à la plupart des attaques que les serveurs mails gérés par de petites entités. Mais cela a un coût : l'accès aux données n'est plus restreint à leur propriétaire et celui-ci doit ainsi faire confiance à une entité tierce.

Cela pose donc le problème de la confidentialité et de l'intégrité des données ainsi externalisées. Pour certaines applications, comme celles concernant les données de santé, il est nécessaire de les protéger contre leur hébergeur. La première façon de faire cela est évidemment le contrôle d'accès, aspect auquel que je ne m'intéresse point dans ma thèse. Mais cela n'est pas suffisant car cela ne protège que contre des attaquants situés en dehors du système d'information. Il faut aussi protéger les données contre une fuite, voire contre les administrateurs de la base externalisée. La protection appropriée est le chiffrement de la base de données ; solution qui, bien qu'évidente, reste un problème ouvert. Ce manuscrit vise donc à étudier, analyser et proposer des solutions à ce problème.

En plus du compromis habituel entre la protection des données (confidentialité et intégrité) et le coût en temps et en mémoire imposé par le chiffrement, le chiffrement des bases de données requiert le maintien d'une certaine structure autour des données. En effet, il est impératif de garder la possibilité de faire des requêtes efficaces sur la base.

C'est pourquoi les méthodes naïves ne sont, malheureusement, pas adaptées. Si l'on se concentre sur l'aspect de la protection des données, un chiffrement « naïf » des fichiers contenant la base consiste à les chiffrer, par exemple, avec un AES en mode CBC. Malgré la sécurité évidente de ce procédé, celui-ci est inintéressant car il requiert le chiffrement presque total de la base à chaque mise à jour, ce qui n'est évidemment pas acceptable.

D'un autre côté, certains chiffrements ont été proposés dans le but d'être utilisés dans le contexte de la protection de bases de données, comme par exemple le chiffrement conservant l'ordre (OPE : *Order Preserving Encryption*). Ses atouts sont clairs. Un OPE assure que l'on peut encore faire des recherches dans la base ainsi chiffrée. Mais il donne aussi de nombreuses informations sur les données claires : leur ordre et en particulier les cas d'égalité sont ainsi révélés. Même si cela peut être acceptable dans certains cas particuliers, cette propriété est en général à proscrire.

Une solution plus prometteuse est l'emploi d'un algorithme de chiffrement complètement homomorphique, qui permet d'obtenir le résultat d'un calcul

sur les clairs en effectuant les opérations sur les chiffrés. Le premier chiffrement ayant cette fonctionnalité a été conçu par G. Gentry [Gen09]. Même si des améliorations ont été proposées récemment, tous les algorithmes de ce type sont à l'heure actuelle beaucoup trop lents pour être utilisés dans le contexte de grandes bases de données.

En l'absence de solutions satisfaisantes qui permettent d'effectuer des requêtes sur des données chiffrées, il est nécessaire de construire des structures de données additionnelles permettant d'accélérer les requêtes, typiquement les index. De nombreuses questions se posent alors. Faut-il construire l'index sur les données chiffrées ou chiffrer un index construit sur des données en clair ? Garder un index non chiffré, construit sur des données non chiffrées, est évidemment inenvisageable car le chiffrement de la base deviendrait alors complètement inutile.

Les travaux réalisés lors de ma thèse sont essentiellement les suivants. Il m'a d'abord fallu appréhender ce problème du chiffrement des bases de données, et identifier les contraintes inhérentes [GJ09]. Les problèmes particuliers liés aux données de santé et notamment aux solutions cryptographiques adaptées au cadre juridique ont été détaillés dans le livrable 5 du projet DEMOTIS [ABC+10].

Ensuite, notre étude des solutions proposées, notamment au sein de la communauté des bases de données, nous a conduit à nous focaliser sur un chiffrement proposé par Ge et Zdonik en 2007 appelé *Fast Comparison Encryption (FCE)* [GZ07], dont la particularité est d'effectuer des comparaisons rapides pour deux chiffrés, c'est-à-dire des comparaisons où il n'est pas nécessaire de déchiffrer complètement les deux chiffrés à comparer. J'ai proposé dans [Jac10a] une cryptanalyse de ce chiffrement qui infirme la sécurité annoncée par les auteurs qui prétendaient qu'attaquer FCE revenait à attaquer le chiffrement par bloc sous-jacent, typiquement l'AES. J'ai en effet proposé un algorithme permettant de retrouver la clef, à partir d'une demi-page de texte clair et de la demi-page chiffrée correspondante, soit environ 2^{15} appels au chiffrement par bloc sous-jacent et 2^{25} multiplications sur 16 bits, ce qui prend moins de 10 minutes sur un PC standard.

À la suite de cette publication, les auteurs de FCE nous ont indiqué une variante de leur algorithme [Ge10] où les calculs s'effectuent dans $GF\left(2^{\kappa}\right)$ au lieu de $\mathbb{Z}/2^{\kappa}\mathbb{Z}$, mais notre attaque s'adapte à ce cas, bien que certaines des optimisations ne soient alors plus possibles. Ce résultat est décrit dans [Jac10b]. Dans ce rapport, figurent aussi les prémisses de nos travaux sur la question du chiffrement de l'index, et, en particulier, la distinction entre le chiffrement d'un index effectué sur les données claires et le calcul d'un index sur des données chiffrées.

Finalement, nous avons analysé la sécurité de méthodes de chiffrement proposées dans [ABG, Guo11] pour chiffrer des bases de données, et proposé

des solutions de chiffrement sûres.

Ce manuscrit se divise en cinq chapitres. Tout d'abord, nous étudierons les moyens cryptographiques qui sont appropriés pour protéger des informations sensibles stockées dans des bases de données. Nous verrons, en particulier, que les outils de chiffrement « classiques » ne suffisent pas et nous passerons donc en revue les outils adaptés. Nous profiterons aussi de ce chapitre introductif pour exposer les notions de bases de données dont nous aurons besoin dans la suite de cette thèse.

Le second chapitre est l'occasion d'aborder la protection des bases de données externalisées, tant du point de vue du contrôle d'accès que du point de vue juridique. Nous prendrons particulièrement soin à expliquer les rôles des différents acteurs et à illustrer ces modèles par des exemples de systèmes de bases de données de santé externalisées. Ce chapitre se conclura sur les différents niveaux de centralisation que peut prendre le stockage de ces données, et leurs implications.

Le troisième chapitre se focalisera sur les moyens existants, commerciaux ou « open-source », pour protéger les bases de données. Pour cela, il faut préalablement distinguer les trois niveaux où le chiffrement peut être appliqué à une base de donnée :

- au niveau du système de gestion des fichiers, des partitions, des disques durs,
- au niveau de la base de données,
- au niveau applicatif.

Cela nous amènera à un état de l'art des moyens de chiffrements actuellement disponibles, que ce soit des cryptoprocesseurs (TPM, HSM...), des logiciels de chiffrement de disques durs, ou des modules de chiffrement au sein des systèmes de gestion des bases de données. À partir de cette étude, nous pourrons alors facilement déduire les fonctionnalités manquantes mais pourtant requises, ou néanmoins importantes pour atteindre notre but.

L'avant-dernier chapitre nous permettra d'évaluer en détail la sécurité des solutions proposées actuellement dans les domaines de recherche conjoints au monde des bases de données et de la cryptographie. En particulier, nous pourrons proposer des améliorations, et surtout effectuer des propositions réalistes, à défaut d'être parfaites.

Finalement, le dernier chapitre portera sur les cryptanalyses de l'algorithme de chiffrement *Fast Comparison Encryption (FCE)* proposé par Ge et Zdonik en 2007 [GZ07], et de sa variante [Jac10a].

Chapitre 1

Protection cryptographique des données externalisées

Pour diverses raisons, de plus en plus de données sont externalisées. Cela peut être dans le but de rendre ces données accessibles depuis n'importe quel terminal informatique comme dans le cas du « cloud computing », ou simplement à cause d'un besoin de profiter de compétences externes. Mais externaliser ainsi des données ne signifie pas toujours que l'on souhaite les divulguer aux personnes les gérant pour nous. Il se peut aussi que l'on désire être capable de s'assurer de leur intégrité. Ces situations nécessitent donc une protection cryptographique des données.

Un des aspects prépondérant pour les utilisateurs de telles solutions est néanmoins la facilité d'accéder à ces données externalisées. En particulier, il est rare qu'un utilisateur souhaite accéder à toutes ses données. Généralement, il voudra seulement en récupérer ou en modifier une partie. Par exemple, lorsque l'on regarde ses emails, il est fréquent de ne vouloir lire que les emails non lus, sans pour autant devoir télécharger l'intégralité de tous nos emails. Il faut donc un chiffrement ayant une granularité relativement fine. Un exemple similaire est le cas du chiffrement de disque dur où il n'est pas acceptable de déchiffrer l'intégralité du disque pour chaque lecture ou écriture. C'est pour cela que dans ce cas, le chiffrement est généralement effectué au niveau de chaque secteur du disque. Mais cette granularité ne doit pour autant pas être la source de nouvelles attaques. En particulier, un attaquant ne doit pas être capable de permuter des secteurs chiffrés, de déterminer si des secteurs sont identiques, ou de permettre la détection de fichiers tatoués (« watermarked »). Ces nouvelles contraintes ont donné lieu à de nouveaux concepts ; notamment les chiffrements par blocs adaptables, ou des modes opératoires avec protection d'intégrité.

Après une présentation de ces techniques cryptographiques, nous décrirons quelques particularités caractéristiques des bases de données qui vont conditionner le choix de la solution cryptographique utilisée pour les proté-

ger. En particulier, les bases de données agrègent des données selon une forte structure particulière, qui jouera nécessairement un rôle important lors du chiffrement, et ont aussi une fonctionnalité supplémentaire qui est la possibilité d'effectuer des requêtes efficaces sur les données. Nous décrirons donc le fonctionnement de ces mécanismes.

1.1 Cryptographie

1.1.1 Propriétés classiques requises

Avant de décrire les propriétés de sécurité requises pour les chiffrements symétriques, il convient de définir dans un premier temps ce type d'algorithme. Un schéma de chiffrement symétrique est constitué d'une paire d'algorithmes : l'algorithme de chiffrement $\mathcal{E}$ et l'algorithme de déchiffrement $\mathcal{D}$. Étant donné un message M et une clef K, son chiffré se déduit comme ceci :

$$C = \mathcal{E}(K, M) = \mathcal{E}_K(M).$$

Pour retrouver le message clair à partir du chiffré C, de la clef K et de l'algorithme de déchiffrement $\mathcal{D}$, on procède comme suit :

$$M = \mathcal{D}(K, C) = \mathcal{D}_K(C).$$

Ces systèmes s'appellent aussi chiffrements à clef secrète, car l'unique clef utilisée est le secret partagé entre les utilisateurs. $\mathcal{D}$ est nécessairement déterministe, c'est-à-dire qu'à un couple (C, K) il associe un unique message M. Par contre, rien n'impose a priori à $\mathcal{E}$ de l'être, et, comme nous le verrons plus tard, ce ne sera généralement pas le cas. C'est-à-dire qu'il est possible que plusieurs chiffrés correspondent à un couple (M, K) fourni à $\mathcal{E}$, dans le cas où il serait randomisé ou pourvu d'un état interne. Dans ce cas, il réalise une expansion sur la taille du chiffré.

Il existe de nombreux niveaux de sécurité en cryptographie s'appliquant à ce type d'algorithmes de chiffrement. Les principaux sont les suivants.

Définition 1 (Non-inversibilité, voir par exemple [Poi02], OW : one-wayness)**.** Soient un algorithme de chiffrement non-inversible $\mathcal{E}$ et un message clair M. Il est impossible de trouver M à partir de $\mathcal{E}(K, M)$ sans connaître K.

Définition 2 (Sécurité sémantique [GM84], IND : indistinguishability)**.** Soit un adversaire ayant une puissance de calcul polynomiale, c'est-à-dire qu'il ne peut faire qu'un nombre polynomial de calculs, et un chiffré. Un chiffrement sûr sémantiquement assure que l'attaquant ne peut déduire aucune information sur le clair à partir du chiffré.

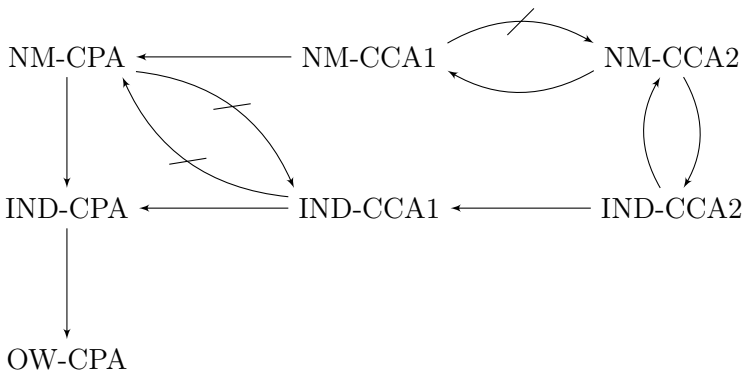
La notion de sécurité sémantique transpose donc la notion de « sécurité parfaite » de Shannon au contexte où l'adversaire a une puissance de calcul limitée.

Définition 3 (Non-malléabilité [DDN00], NM : non-malleability)**.** Soient $\mathcal{E}$ un algorithme de chiffrement non-malléable, M un message clair, et $C = \mathcal{E}(K, M)$ le chiffré de M. Alors aucun attaquant polynomial ne peut dériver un deuxième chiffré $C' = \mathcal{E}(K', M')$ tel que M et M' soient reliés.

Les propriétés précédentes peuvent être étudiées dans différents contextes, en fonction du type de données (clairs, chiffrés correspondants) contrôlées par l'attaquant. On distingue classiquement plusieurs modèles d'adversaires :

- attaque à clairs choisis (CPA : chosen plaintext attack) : une attaque à clairs choisis est une attaque où l'attaquant peut obtenir le chiffré de tout message clair de son choix, soit grâce à une clef publique, soit via un oracle de chiffrement ;
- attaque à chiffrés choisis (CCA : chosen ciphertext attack) : une attaque à chiffrés choisis est une attaque où l'attaquant a accès à l'oracle de déchiffrement et peut ainsi demander le déchiffrement de tout chiffré de son choix ; il existe deux variantes d'attaques à chiffrés choisis :
  - attaques non-adaptatives (CCA1) : l'accès à l'oracle est uniquement autorisé avant que l'attaquant reçoive le challenge,
  - attaques adaptatives (CCA2) : l'accès à l'oracle est ici illimité dans le temps.

FIGURE 1.1 – Relations entre les notions de sécurité [BDPR98]



Légende : $\rightarrow$: implication prouvée
$\nrightarrow$: non implication prouvée

La figure 1.1 présente les relations entre ces notions de sécurité [BDPR98]. Il en découle que le niveau de sécurité minimum acceptable est IND-CPA, alors que le meilleur est IND-CCA2 (que nous noterons dorénavant IND-CCA) et NM-CCA2. Dans le but de bien clarifier le contexte, nous allons donc redéfinir précisément les notions IND-CPA et IND-CCA.

Définition 4 (IND-CPA [BR05])**.** Considérons un adversaire qui choisit deux messages de même longueur en un temps polynomial et après un nombre de requêtes polynomial à l'algorithme de chiffrement. Seulement l'un des deux messages clairs est chiffré et son chiffré est donné à l'adversaire. Ce dernier a le droit d'interroger l'algorithme de chiffrement avec n'importe quel message.

Le schéma de chiffrement est considéré IND-CPA si cet adversaire ne peut pas décider avec une probabilité significativement plus grande que $\frac{1}{2}$ lequel des deux messages a été chiffré.

Cette définition implique que le chiffrement doit être non déterministe, c'est-à-dire que deux messages identiques chiffrés successivement n'auront pas le même chiffré. Il y a essentiellement deux façons d'atteindre ce but. La première est de rendre le chiffrement aléatoire. Pour cela l'algorithme de chiffrement doit générer une valeur aléatoire qui est utilisée lors du chiffrement. Elle sera nécessaire pour le déchiffrement et induit donc une expansion du message chiffré par rapport au message clair. L'autre solution consiste à considérer une fonction de chiffrement avec un état interne, c'est-à-dire que la fonction a un état interne qui se met à jour après chaque chiffrement utilisant la même clef. La valeur de cet état étant aussi nécessaire au déchiffrement, cette solution implique aussi une expansion.

Définition 5 (IND-CCA [BR05])**.** Considérons un adversaire qui choisit deux messages chiffrés de même longueur en un temps polynomial. Seulement l'un des deux messages chiffrés est déchiffré et son clair est donné à l'adversaire. Il a le droit d'interroger l'algorithme de déchiffrement avec n'importe quel message, hormis les deux qu'il a choisis.

Le schéma de chiffrement est considéré IND-CCA si cet adversaire ne peut pas décider avec une probabilité significativement plus grande que $\frac{1}{2}$ lequel des deux messages a été déchiffré.

En cryptographie symétrique, les modèles d'adversaire peuvent être raffinés. Il existe des variantes de ces attaques, par exemple dans lesquelles l'adversaire a accès à l'algorithme de chiffrement tout au long de l'attaque, mais n'a accès au déchiffrement que lors de la phase des choix des deux messages clairs. Les relations entre ces variantes ont été établies par Katz et Yung [KY00a]. Dans le cadre de la cryptographie symétrique, Bellare et al. [BDJR97] ont redéfini l'indistinguabilité par d'autres objectifs de sécurité (LOR, FTG et ROR).

Définition 6 (Real-or-random (ROR) [BDJR97])**.** L'attaquant envoie un message clair à l'oracle. Celui-ci génère un message clair aléatoire de la même longueur et renvoie le chiffré de ces deux messages.

Le schéma de chiffrement est considéré ROR si cet adversaire ne peut pas décider facilement lequel des deux messages chiffrés correspond à celui qu'il a envoyé à l'oracle.

Définition 7 (Find-then-guess (FTG) [BDJR97])**.** Cette notion de sécurité se décompose en deux phases :

- find (chercher) : l'attaquant peut faire des requêtes de chiffrement à l'oracle. Il doit ensuite donner deux messages m_0 et m_1 à l'oracle.
- guess (deviner) : l'oracle lui retourne le chiffré de l'un de ces deux messages. L'attaquant peut continuer à effectuer des requêtes à l'oracle, mais doit finir par deviner quel message ce dernier a chiffré.

Le schéma de chiffrement est considéré FTG si cet adversaire ne peut pas décider facilement lequel des deux messages a été chiffré.

Comme pour le cas de IND-CPA, il est clair qu'un chiffrement déterministe ne peut pas atteindre ce niveau de sécurité car l'attaquant a ici le droit de demander le chiffrement des deux messages qu'il a choisis.

Définition 8 (Left-or-right (LOR) [BDJR97])**.** L'attaquant formule des requêtes du type (m, m'), où les deux messages ont la même longueur, à l'oracle. Celui-ci lui retourne toujours le chiffré du même côté (droit ou gauche).

Le schéma de chiffrement est considéré LOR si cet adversaire ne peut pas décider facilement si l'oracle lui retourne les chiffrés du message de gauche ou de celui de droite.

On peut remarquer que l'attaquant peut aussi demander le chiffré d'un message donné en envoyant (m, m) à l'oracle.

1.1.2 Primitives de chiffrement symétrique

Les algorithmes de chiffrement à flot et par blocs constituent les deux grandes classes de primitives symétriques. La distinction entre ces deux classes n'est pour autant pas toujours aisée. Habituellement, le chiffrement à flot désigne les algorithmes opérant sur des blocs de clair de taille relativement petite (généralement un bit, un octet ou un mot) au moyen d'une transformation qui varie au cours du temps. Par opposition, un algorithme de chiffrement par blocs applique la même fonction à différents blocs dérivés du clair, qui sont de taille plus conséquente, typiquement 64, 128 ou 256 bits [MvOV97].

Nous distinguerons donc deux briques de base du chiffrement symétrique :

- les générateurs pseudo-aléatoires,

- les chiffrements par blocs.

Ce sont deux objets mathématiques distincts qui sont utilisés de diverses manières. Par exemple, il est possible d'utiliser un chiffrement par blocs pour fabriquer un générateur pseudo-aléatoire, grâce à des modes opératoires adéquats que nous verrons plus loin.

Intuitivement, ces deux types de chiffrement nous seront utiles. En effet, les bases de données contiennent des champs de taille variable, pouvant être aussi petits qu'un bit ou tout aussi bien en faire plusieurs centaines. Cela nous incite à considérer ces deux types de chiffrement comme étant d'intérêt pour atteindre notre but.

Chiffrement à flot

Dans la suite de ce document nous limiterons l'utilisation du terme chiffrement à flot aux seuls algorithmes synchrones, ceci étant en partie motivé par le fait que les chiffrements à flot auto-synchronisants sont plus ou moins tombés en désuétude.

Définition. Un algorithme de chiffrement à flot synchrone consiste à combiner le clair avec une suite binaire de même longueur, la suite chiffrante. Notons $\boldsymbol{s} = (s_t)_{t\geq 0}$ cette suite qui est engendrée par un automate à états finis, appelé générateur pseudo-aléatoire, de façon indépendante à la fois du clair et du chiffré. Le rôle de ce générateur est de produire à chaque moment t un bloc de m bits, s_t, qui est fonction de son état interne $\boldsymbol{x}_t$. Le générateur peut se décomposer en trois fonctions, comme cela est décrit à la figure 1.2 :

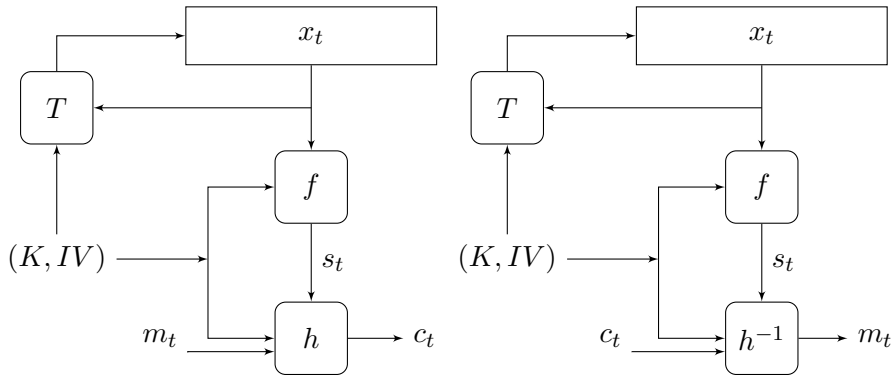
- *une procédure d'initialisation* qui, à partir de la clef secrète et d'un vecteur d'initialisation public, calcule l'état initial du générateur, $\boldsymbol{x}_0$. Cette étape peut parfois être scindée en deux phases. La première, le chargement de clef, consiste en un calcul d'une valeur ne dépendant que de la clef. La seconde, appelée injection d'IV ou de re-synchronisation, détermine alors l'état initial du générateur à partir de la valeur obtenue précédemment et de l'IV. Ce découpage permet de gagner du temps lors d'un changement de l'IV sans changement de clef, ce qui est fréquent, par exemple lors de l'utilisation d'un protocole de communication pour lequel la longueur des paquets échangés est relativement petite, comme par exemple pour les communications téléphoniques, ou dans le contexte des bases de données où les champs peuvent être très petits et les IV varient d'un champ à l'autre et d'une mise à jour à l'autre.

- *une fonction de transition*, notée T, qui fait passer l'état interne du générateur de l'instant t à l'instant $(t+1)$. En général, cette fonction est fixe, mais il se peut qu'elle varie avec la clef, l'IV, voire même avec le temps.

- *une fonction de filtrage*, notée f, qui retourne le bloc de suite chiffrante à partir de l'état interne courant, $\boldsymbol{x}_t$. Comme la fonction de transition, la fonction de filtrage est généralement fixe pour des raisons de simplicité et d'encombrement pour les implémentations matérielles.

Pour chiffrer, il suffit alors de combiner la suite chiffrante au clair à l'aide d'une fonction h inversible, qui associe un bloc de ℓ bits de texte chiffré à un couple de ℓ bits de suite chiffrante et de clair. Dans l'immense majorité des cas, la fonction h correspond au XOR bit à bit, c'est-à-dire à l'addition modulo 2, d'où le nom de chiffrement synchrone additif.

FIGURE 1.2 – Chiffrement à flot synchrone et déchiffrement associé



Les générateurs pseudo-aléatoires. Il y a de nombreuses familles de générateurs pseudo-aléatoires, mais seules les constructions dédiées, au sens où elles sont spécialement conçues pour cet usage, permettent d'obtenir un très haut débit dans un environnement logiciel ou une mise en œuvre très peu coûteuse dans un contexte matériel ; ces deux objectifs étaient d'ailleurs ceux du projet eSTREAM [ECR05]. Ainsi, ni les générateurs sûrs d'un point de vue calculatoire, comme par exemple [ACGS88], dont la sécurité repose sur la difficulté de certains problèmes mathématiques, ni les générateurs inconditionnellement sûrs selon certains modèles, comme par exemple [Mau92, AR99, Rab05], ne respectent ces contraintes malgré le grand intérêt qu'elles présentent du point de vue de la sécurité.

Les contraintes imposées précédemment rendent la conception de tels générateurs très complexe, d'où leur rareté. Parmi les plus anciens figurent SNOW 2.0 [EJ02], SNOW 3G et MUGI [Hit01] qui ont été pris en compte dans la norme internationale de chiffrement ISO/IEC 18033-4 [ISO05]. Plus récemment, le projet eSTREAM [ECR05] du réseau européen ECRYPT a recommandé deux listes de générateurs pseudo-aléatoires : une pour une implémentation logicielle, l'autre pour une implémentation matérielle. Après

plusieurs révisions, les générateurs actuellement recommandés pour être utilisés en logiciel sont HC-128, Rabbit, Salsa20/12 et SOSEMANUK et ceux pour une implémentation matérielle sont Grain v1, MICKEY v2 et Trivium. Finalement, la norme ISO/IEC 18033-4 [ISO05] a été amendée en 2009 pour intégrer Rabbit et DECIM.

Générateurs pseudo-aléatoires dédiés. Classer ces algorithmes dédiés est difficile à cause de leur grande disparité mais on peut toutefois les séparer en trois grandes familles.

En terme d'implémentation, les chiffrements à transition linéaire s'imposent comme le choix naturel car le plus simple, mais il faut bien faire attention à ce que la fonction de filtrage f ne soit ni linéaire, ni proche d'une fonction linéaire. Les chiffrements utilisant des registres à décalage à rétroaction linéaire (LFSR) sont prépondérants dans cette catégorie à la fois parce que leur implémentation a un coût très faible et que l'on dispose de nombreux résultats théoriques sur les propriétés statistiques des suites chiffrantes qu'ils produisent. De plus ils peuvent être utilisés dans un environnement matériel (registres à décalage à rétroaction linéaire binaires) et dans un environnement logiciel, mais dans ce cas on opère sur un alphabet plus grand que le bit, en général sur un octet ou un mot de processeur. Les générateurs à base de LFSR les plus connus sont E0, déployé dans la norme Bluetooth, A5/1 et son cousin A5/2 utilisés dans la norme GSM pour les communications avec des téléphones portables, SNOW 2.0 qui est inclus dans la dernière norme ISO/IEC 18033 et son grand frère SNOW 3G qui est destiné aux téléphones 3G, ou encore SOSEMANUK un des chiffrements retenu par le projet eSTREAM. Lors de la conception de tels chiffrements il faut faire très attention aux attaques désormais classiques telles que les attaques algébriques, les cryptanalyses linéaires et différentielles, ainsi qu'à la perpétuelle émergence de nouvelles attaques.

La raison d'être des chiffrements à transition non-linéaire est d'éviter les faiblesses potentielles dues au caractère linéaire de la fonction de transition. Toutefois, la fonction de transition choisie doit garantir que les états internes du générateur ne forment pas une suite de faible période, et ce quelle que soit la valeur de l'état initial, ce qui est amplement plus difficile à prouver que pour des fonctions linéaires. Un moyen de contourner ce problème, dont l'exemple le plus connu est RC4, est de s'autoriser une taille d'état interne très grande, ce qui entraîne généralement des problèmes d'implémentation. Mais même ce contournement n'est pas complètement satisfaisant car les chiffrements ainsi créés sont particulièrement vulnérables aux attaques par canaux cachés qui exploitent l'analyse du comportement du cache. Sans une grande taille d'état interne, il faut des résultats théoriques sur la période de la fonction de transition que peu de fonctions vérifient. Un exemple de chiffrement à transition non-linéaire sont les FCSRs (Feedback with Carry

Shift Register) [KG97]. Ce modèle a été utilise dans un des candidats retenus par le projet eSTREAM, F-FCSR. Ce chiffrement a ensuite été retiré du portfolio à cause de certaines attaques [HJ08].

Finalement, entre ces deux types de transition, il existe des conceptions hybrides. En effet, il est possible de diviser l'état interne d'un générateur en deux parties, l'une étant mise à jour par une fonction linéaire, l'autre par une fonction non-linéaire. Si cette seconde partie est sensiblement plus petite que l'autre, elle est souvent assimilée à une mémoire interne et le générateur ainsi formé est alors classé comme un générateur à transition linéaire avec mémoire, comme par exemple pour SNOW 2.0, SNOW 3G et E0. Toutefois, il existe des générateurs dans lesquels les deux parties sont de tailles similaires, comme par exemple Grain v1 [HJM05] un des chiffrements recommandés par eSTREAM pour les implémentations matérielles.

Utilisation des chiffrements à flot. Les avantages des algorithmes de chiffrements à flot découlent de la petite taille de bloc utilisée, ce qui est d'autant plus vrai dans le cas des chiffrements où le bloc est réduit à un unique bit. Cette faible taille permet une réduction des délais et de la taille de la mémoire-tampon nécessaire pour stocker le message avant l'obtention d'un bloc complet. À cela s'ajoute le fait que ces chiffrements à flot n'ont évidemment pas besoin de remplissage, ce qui est fortement désirable lorsque la bande passante est faible ou que le protocole utilisé nécessite l'utilisation de paquets courts. L'emploi de blocs de petite taille limite aussi la propagation d'erreurs de transmission lors du déchiffrement. Ces propriétés sont aussi particulièrement adaptées au chiffrement de bases de données, dans le cas où les champs sont de petite taille.

Lorsque les ressources sont limitées, souvent parce qu'il faut restreindre la consommation électrique du circuit électrique dédié au chiffrement, ou qu'il est nécessaire de pouvoir chiffrer et déchiffrer très rapidement, on utilise généralement des algorithmes de chiffrement à flot, par exemple sur les systèmes embarqués.

Comparaison. Comparer les vitesses des différents chiffrement n'est pas aisé. La première difficulté est qu'il faut comparer à la fois les vitesses en logiciel et en matériel. Ensuite, pour un même algorithme de chiffrement, les performances varient grandement en fonction des architectures utilisées. La répartition des temps varie aussi d'un algorithme à un autre : la mise à la clef, la resynchronisation, la production de la suite chiffrante prennent des temps proportionnellement variables. Les implémentations ont aussi un grand rôle dans ces comparaisons. Il est donc impossible de classer une fois pour toutes les différents algorithmes, comme on le comprend bien à la vue des graphiques proposés par D. J. Bernstein[1].

[1] http://cr.yp.to/streamciphers/timings.html

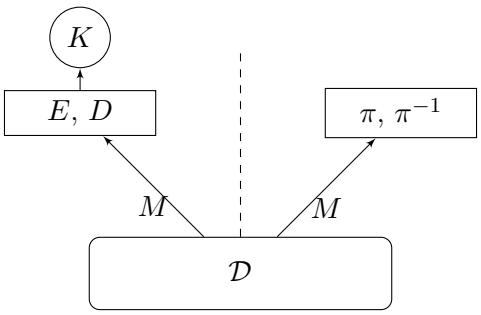
Chiffrement par blocs

Définition. Un algorithme de chiffrement par blocs opère sur un bloc de taille donnée fixe, qui peut valoir typiquement 64, 128, 160 ou 256 bits. Ce bloc est alors transformé par une fonction E_K inversible spécifiée par une clef K. Plus généralement, si k est la taille de la clef et b la taille du bloc, on a $E : \{0,1\}^k \times \{0,1\}^b \to \{0,1\}^b$ et $D : \{0,1\}^k \times \{0,1\}^b \to \{0,1\}^b$ tels que, pour tout bloc M de taille b et toute clef K de taille k, $E(K, D(K, M)) = M$.

Sécurité des chiffrements par blocs. Le rapport [ECR10], décrit les propriétés désirées pour un bon chiffrement par bloc. La première est l'absence d'attaque connue permettant de récupérer la clef en moins de 2^n opérations de chiffrement. Une notion plus avancée est celle d'indistinguabilité dont la description est la suivante. Un attaquant reçoit deux mots de b bits. L'un est un message clair M, l'autre est soit le chiffré de M, $C = E(K, M)$ pour une clef inconnue K, soit un mot aléatoire de b bits. Si cet attaquant ne peut pas réussir à déterminer lequel des messages il a reçu avec une probabilité meilleure qu'un demi, le chiffrement sera considéré comme indistinguable d'une fonction aléatoire. Cela laisse présager que le chiffrement ne donnera pas d'information sur le message clair.

De façon plus formelle, la sécurité d'un algorithme de chiffrement par blocs est l'avantage maximal qu'a un attaquant pour distinguer la permutation $M \mapsto E_K(M)$ pour une clef K inconnue choisie aléatoirement, d'une permutation $M \mapsto \pi(M)$ choisie uniformément au hasard dans l'ensemble des permutations possibles de $\{0,1\}^b$, en un temps t et avec q évaluations de la permutation et de son inverse. Cela est décrit à la figure 1.3, où $\mathcal{D}$ est le distingueur.

FIGURE 1.3 – Indistinguabilité d'un algorithme de chiffrement par blocs



Cela n'est évidemment pas suffisant, et il existe d'autres modèles d'attaques. En particulier, les attaques à clefs liées [Bih94] et les attaques à clefs connues [KR07] sont très étudiées, et peuvent aussi être des scénarios perti-

nents lorsque le chiffrement par blocs est utilisé dans un contexte particulier, par exemple pour construire une fonction de hachage.

Constructions. Tout algorithme de chiffrement par blocs est constitué d'un certain nombre d'itérations d'une fonction f. f diffère d'un tour à l'autre car elle n'utilise pas directement la clef maître K, mais des sous-clefs K_i déduites de K par un algorithme appelé cadencement de clef. De plus, il est fréquent que plusieurs paramètres (constantes...) de f changent d'un tour à l'autre. Il y a principalement deux façons de construire un telle fonction f. La première est un réseau de substitution-permutation où la fonction f est constituée de substitutions et de permutations de bits du bloc. Une généralisation de cette construction utilise une permutation linéaire du bloc et non simplement une permutation de ses bits. C'est le cas, par exemple de l'AES. L'autre méthode consiste à utiliser un schéma de Feistel, comme décrit à la figure 1.4. Pour cela, on découpe le bloc M en deux morceaux de tailles égales L_0 et R_0. f transforme alors (R_0, L_0) en (R_1, L_1) de la façon suivante :

$$\begin{cases} R_1 &= L_0 \oplus F(K, R_0) \\ L_1 &= R_0. \end{cases}$$

On remarque qu'un tel système est facilement inversible et que le déchiffrement s'en déduit aisément :

$$\begin{cases} R_0 &= L_1 \\ L_0 &= R_1 \oplus F(K, L_1). \end{cases}$$

Cela permet d'utiliser le même circuit pour chiffrer et déchiffrer.

FIGURE 1.4 – Schéma de Feistel

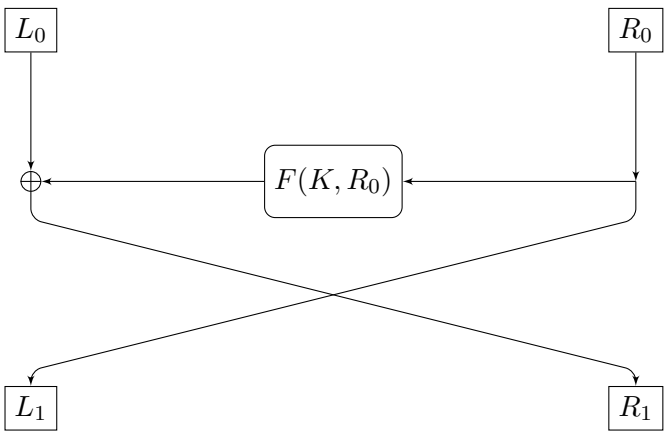
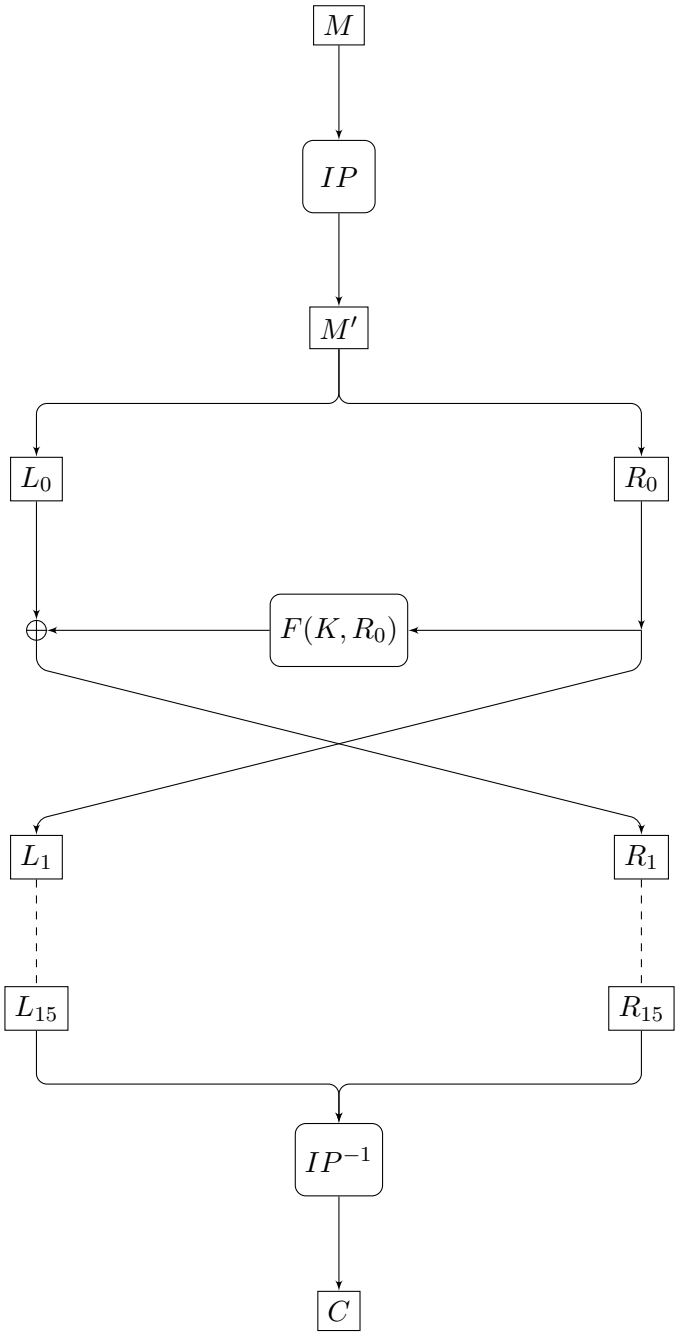


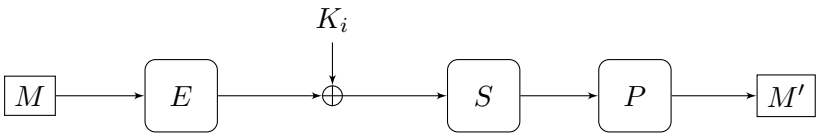
FIGURE 1.5 – Schéma du DES



Exemples de chiffrements par blocs. Le standard du DES [FIP77] fut publié le 15 janvier 1977. DES est un système de chiffrement par blocs de 64

bits avec une clef de 56 bits qui est stockée avec 8 bits de parité sur 64 bits. Il utilise un schéma de Feistel à 16 tours décrit à la figure 1.5. Dans ce schéma, IP désigne une permutation définie dans le standard et F est la fonction définie dans la figure 1.6, où E est une expansion de message de 32 à 48 bits, P une permutation et S les boîtes S 6×4 (tables de substitution) du DES. Les sous-clefs $\{K_i\}_{i=0...15}$ sont dérivées de la clef K via deux permutations $PC1$ et $PC2$ et des décalages donnés par une table.

FIGURE 1.6 – Fonction F du DES



Il existe plusieurs types d'attaques qui permettent de casser le DES sans tester toutes les clefs possibles. La première est celle datant de 1991 par Eli Biham et Adi Shamir [BS91] ; à partir de 2^{47} couples clairs-chiffrés choisis en utilisant leur méthode appelée cryptanalyse différentielle, ils retrouvent la clef. Mitsuru Matsui [Mat94] fait de même en 1993 à partir de 2^{43} couples clairs-chiffrés connus avec sa méthode, la cryptanalyse linéaire. D'autres attaques existent, mais la plus efficace en pratique reste certainement l'attaque exhaustive. En 1998, l'Electronic Frontier Fondation [EFF98] a fait fabriquer une machine dédiée pour environ 250 000 $ qui retrouve une clef en seulement 3 jours. Ce temps a été amélioré en utilisant une machine à base de FPGA par COPACOBANA (COst-optimized PArallel COdeBreaker) et réduit à moins d'une journée[2]. Cela a eu pour résultat le retrait du standard du DES (FIPS 46-3) par le NIST en 2005.

Pour cela, il a fallu améliorer DES. La première idée, faire un double DES, n'apporte guère plus de sécurité car à partir de $C = DES_{K_1}(DES_{K_2}(M))$, on peut faire une attaque dans le milieu en précalculant les $DES_{K_2}(M)$, puis en cherchant seulement une clef K_1 donnant une égalité à l'équation $DES^{-1}_{K_1}(C) = DES_{K_2}(M)$. Donc le coût d'une attaque sur un double DES est seulement 2^{57}. Il faut donc utiliser un triple DES avec 112 bits de clef, dont l'attaque dans le milieu a pour complexité 2^{113} :

$$C = DES_{K_1}\left(DES^{-1}_{K_2}(DES_{K_1}(M))\right).$$

Normalisé officiellement le 26 novembre 2001 sous le nom AES [FIP01], le chiffrement Rijndael, développé par Joan Daemen et Vincent Rijmen, est le vainqueur de la compétition AES organisée par le NIST pour trouver un successeur au DES. Cet algorithme est un réseau de substitution-permutation

[2] http://www.sciengines.com/company/news-a-events/74-des-in-1-day.html

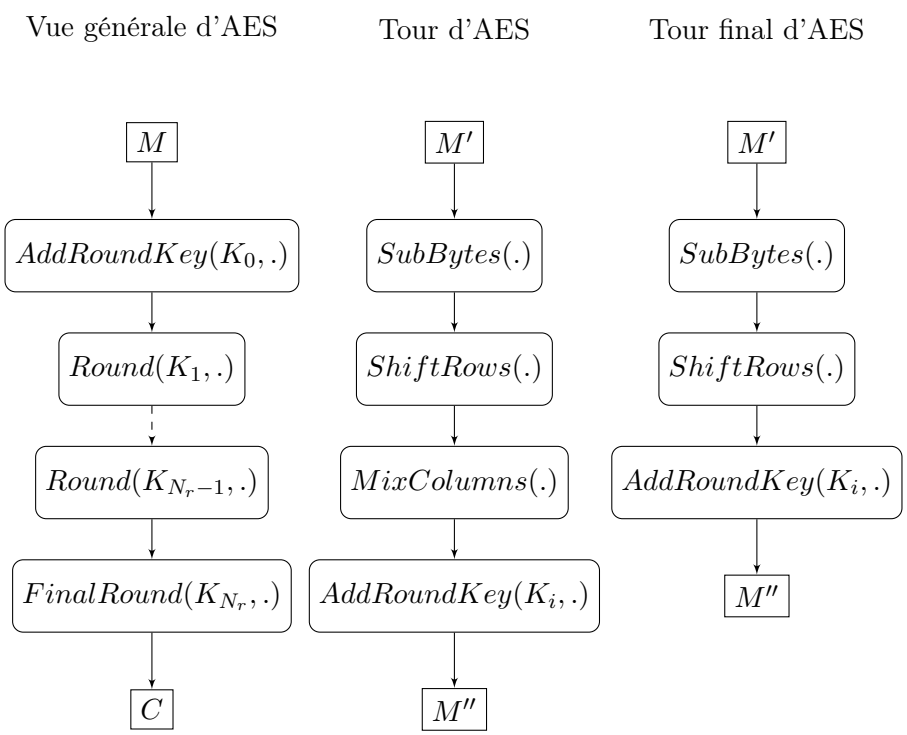
au sens large. La taille de bloc d'AES est 128 bits, contrairement à Rijndael qui supporte tous les multiples de 32 supérieurs à 128 bits. Les deux autorisent des clefs de 128, 192 et 256 bits. Le nombre de tours dépend de la taille du bloc et de la taille de la clef comme décrit dans le tableau 1.1.

TABLE 1.1 – Tours de Rijndael en fonction des tailles de clef et de bloc

clef / bloc	128	196	256
128	10	12	14
196	12	12	14
256	14	14	14

Le déroulement d'AES est décrit dans la figure 1.7, où N_r est le nombre de tours comme précisé dans le tableau 1.1, et où les fonctions de tour et du tour final sont détaillées.

FIGURE 1.7 – Advanced Encryption Standard (AES)



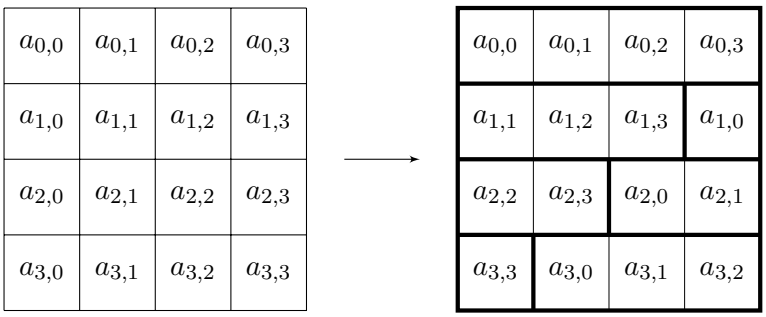
Le plus simple pour décrire les fonctions internes d'AES-128 est de représenter un bloc interne de 128 bits comme un carré de 4×4 cases contenant 8 bits, comme montré à la figure 1.8. *ByteSub* consiste à appliquer une S-

Box aux octets de l'état interne, c'est-à-dire à transformer chaque $a_{i,j}$ en $S(a_{i,j})$. $ShiftRows$ est elle décrite à la figure 1.9, c'est un décalage appliqué aux lignes de l'état interne. $MixColumns$ mélange les différents octets situés sur une même colonne en appliquant une transformation linéaire aux 4 octets d'une colonne. Finalement, l'ajout de la clef de tour se fait grâce à $AddRoundKey$ qui met à jour $a_{i,j}$ en $a_{i,j} \oplus k_{i,j}$, où $k_{i,j}$ est l'octet correspondant dans la sous-clef du tour concerné.

FIGURE 1.8 – État interne d'AES

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

FIGURE 1.9 – Fonction ShiftRows d'AES



Utilisation des chiffrements par blocs. Pour l'instant, nous avons vu comment chiffrer un bloc d'une taille donnée avec un algorithme de chiffrement par bloc, or en pratique les messages à chiffrer sont souvent plus longs que cette taille imposée, et rien n'assure que leur longueur en soit un multiple. Pour toutes ces raisons, il est donc nécessaire d'avoir des règles d'utilisation des chiffrement par blocs : remplissage (padding), modes de chiffrement, vecteur d'initialisation.

Pour pouvoir utiliser un chiffrement par blocs, il faut d'abord découper le message à chiffrer en blocs de taille souhaitée. Mais rien n'assure que

le dernier bloc sera plein. Pour cela, il faut donc le compléter. Plusieurs critères peuvent alors influencer le remplissage. Le premier est de garantir la réversibilité du remplissage : si la règle est de rajouter un 1 puis autant de 0 que nécessaire pour remplir le bloc, ceci est facile à enlever lors du déchiffrement, alors que si on ne rajoute que des 0 on ne saura pas lesquels enlever à ce moment car le message peut se finir par des 0. Dans les cas de vérification d'intégrité (fonction de hachage, AES utilisé en mode CBC...), il peut être aussi nécessaire d'ajouter la longueur du message sur un bloc à la fin du message pour éviter la recherche de collision avec des mots plus longs.

Notons que, en cryptographie asymétrique, le remplissage sert souvent à s'assurer que le message à chiffrer respectera bien une liste de règles sans lesquels le chiffrement devient faible. Finalement, dans le cas des chiffrements à flot, le remplissage peut au contraire servir à cacher la longueur du message.

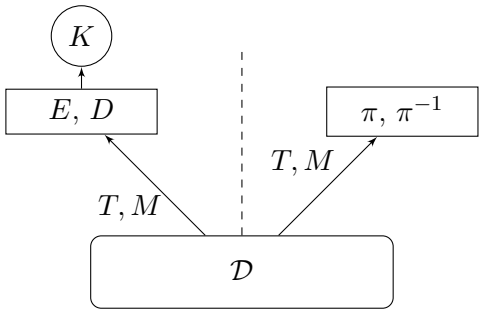
Chiffrements par blocs adaptables

Les chiffrements par blocs tels que décrit ci-dessus ont un intérêt limité dans la mesure où ils ne permettent pas naturellement de tenir compte de l'emplacement de la donnée chiffrée, par exemple du numéro de secteur dans le cas de chiffrement de disque dur, ou des numéros de table, ligne et colonne dans le cas de chiffrement de bases de données. Pour combler ce manque, les modes de chiffrements par blocs adaptables (« tweakable encryption modes ») sont récemment apparus.

Définition. Les chiffrements adaptables, introduits par Liskov, Rivest et Wagner [LRW02], sont particulièrement pertinents dans le contexte de chiffrement de disques durs. En effet, l'aspect adaptable consiste en l'ajout d'un paramètre T appelé « tweak » indépendant de la clef K. Le tweak dépend généralement du numéro de secteur et du numéro du bloc à chiffrer et permet de prévenir le déplacement de données sur le disque par un attaquant. Plus précisément, si k est la taille de la clef, b la taille du bloc et t la taille du tweak, un algorithme de chiffrement adaptable $\widetilde{E}$ et son algorithme de déchiffrement associé $\widetilde{D}$ sont définis par $\widetilde{E} : \{0,1\}^k \times \{0,1\}^b \times \{0,1\}^t \to \{0,1\}^b$ et $\widetilde{D} : \{0,1\}^k \times \{0,1\}^b \times \{0,1\}^t \to \{0,1\}^b$ tels que, pour tout bloc M de taille b, toute clef K de taille k et tout tweak T de taille t, $\widetilde{D}\left(K, T, \widetilde{E}\left(K, T, M\right)\right) = M$.

La notion de sécurité est ici plus forte que dans le cas d'un chiffrement par blocs classique. Cette fois l'attaquant tente de distinguer $(T, M) \mapsto E_K(T, M)$ d'une famille de permutations $(T, M) \mapsto \pi_T(M)$ où les permutations $(\pi_T)_{T \in \{0,1\}^t}$ sont indépendantes et tirées au hasard. Cela est illustré par la figure 1.10 et doit être comparé à la notion d'indistinguabilité classique décrite à la figure 1.3.

FIGURE 1.10 – Indistinguabilité d'un algorithme de chiffrement adaptable



Trivialement, un chiffrement adaptable sûr utilisé avec un tweak constant (T constant) est un chiffrement par bloc sûr. Il en découle que contrairement aux modes opératoires avec IV, que nous verrons en détail plus tard, il ne sera pas nécessaire d'imposer l'utilisation de tweaks uniques. L'inverse est bien évidemment plus compliqué. Pour construire un chiffrement adaptable à partir d'un chiffrement par blocs classique, on pourrait par exemple choisir comme tweak une partie de la clef secrète, mais dans ce cas, le chiffrement adaptable n'est sûr que si le chiffrement par bloc est une fonction super-pseudo-aléatoire. Autrement dit, cela impose que les fonctions E_K soient indépendantes. Ce n'est que rarement le cas en pratique, comme le montrent les attaques à clefs liées sur l'AES [BKN09].

De plus, il doit être peu coûteux de changer de tweak. Autrement dit, calculer $E_K(T, M)$ puis $E_K(T', M')$ ne doit pas être plus coûteux que de calculer $E_K(T, M)$ puis $E_K(T, M')$.

Exemples de chiffrements par blocs adaptables La principale façon de créer un chiffrement par blocs adaptable est de transformer, d'adapter un chiffrement par blocs classique. Le premier chiffrement adaptable présenté fut LRW [LRW02]. Rogaway [Rog04] a proposé XE et XEX qui est une variante de LRW. La table 1.2 montre l'adaptation qu'il faut opérer sur un chiffrement par blocs E classique pour obtenir les chiffrements par blocs adaptables évoqués ci-dessus.

TABLE 1.2 – Chiffrements par blocs adaptables.

LRW [LRW02]	$E_K\left(M \oplus h\left(T\right)\right) \oplus h\left(T\right)$
XE [Rog04]	$E_K\left(M \oplus \Delta\right)$
XEX [Rog04]	$E_K\left(M \oplus \Delta\right) \oplus \Delta$

Pour LRW, l'espace des tweaks $\mathcal{T}$ est le même que celui des clefs, c'est-

à-dire qu'il est de la forme $\{0,1\}^k$. Pour XE et XEX, l'espace des tweaks est $\mathcal{T} = \{0,1\}^k \times \mathcal{I}$ où $\mathcal{I} = [1 \dots 2^{k/2}] \times \mathcal{I}_1 \times \dots \times \mathcal{I}_\ell$ et $\mathcal{I}_i \subseteq \mathbb{Z}$. Un tweak se définit alors comme $T = (N, i_1, \dots, i_\ell)$. Finalement, les éléments α_i appartenant à $\mathbb{F}^*_{2^k}$ sont des paramètres des constructions XE et XEX. À partir de cela, la valeur Δ utilisée ci-dessus vaut : $\Delta = \alpha_1^{i_1} \dots \alpha_\ell^{i_\ell} E_K(N)$.

1.1.3 Les modes de chiffrement

Les modes opératoires classiques

Le mode opératoire définit comment on construit, à partir d'une fonction opérant sur des blocs de taille fixe, une transformation permettant de chiffrer des messages de longueur arbitraire. Dans ce contexte, il est encore question de sécurité de type IND-CPA ou IND-CCA pour les schémas de chiffrement. Plus précisément, il s'agit ici de construire deux algorithmes $\mathcal{E}$ et $\mathcal{D}$ qui prennent en entrée une clef et un message clair, respectivement un message chiffré, et retournent un chiffré, respectivement un clair, comme vu à la section 1.1.1. $\mathcal{E}$ est randomisé ou avec un état interne pour pouvoir atteindre les niveaux de sécurité requis. $\mathcal{D}$ est lui nécessairement déterministe.

Parmi les modes opératoires [NIS01] les plus connus ou les plus utilisés se trouvent ECB (figure 1.11), CBC (figure 1.12) et CTR (figure 1.13). Le premier présenté, ECB, n'est pas IND-CPA car il n'est pas randomisé et sans état interne. [BDJR97] et le chapitre 4 de [BR05] démontrent les résultats suivants. Le mode CBC peut, lui, avoir deux variantes. La première est un CBC randomisé, où l'IV est tiré au hasard. La seconde est un CBC avec état interne. Cet état interne est en fait un compteur. Parmi ces deux versions seule CBC-randomisé atteint un niveau de sécurité IND-CPA. Le mode CTR dispose de ces deux variantes, randomisé et avec état interne ; et elles sont toutes les deux IND-CPA, mais avec des bornes de sécurité différentes.

FIGURE 1.11 – Mode ECB

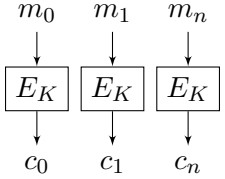


FIGURE 1.12 – Chiffrement et déchiffrement en mode CBC

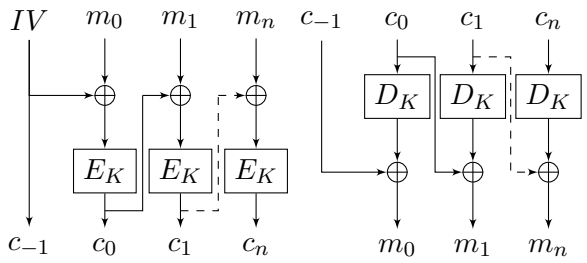
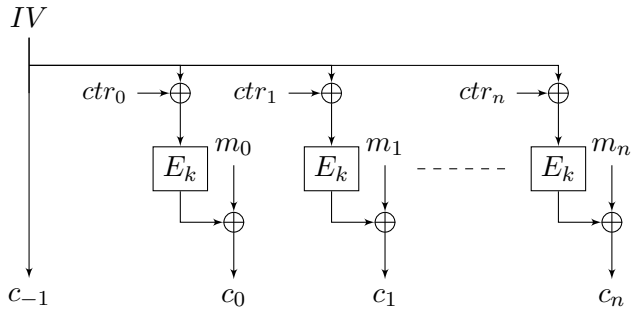


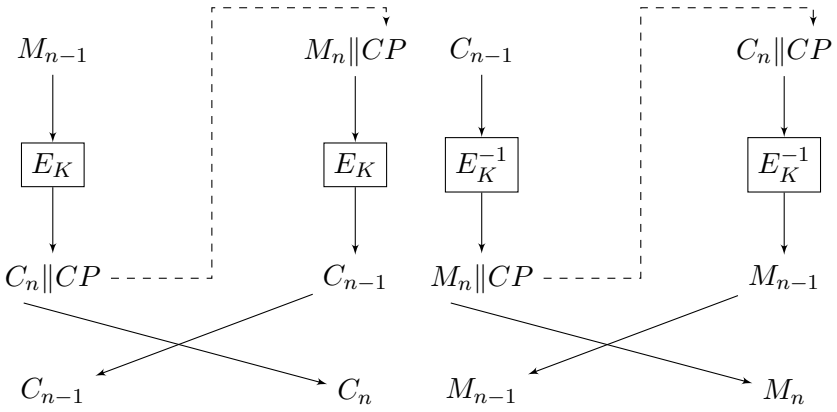
FIGURE 1.13 – Mode CTR



La présence du bloc c_{-1} est justifiée par la nécessité de connaître l'IV pour pouvoir déchiffrer. L'utilisation d'un IV induit donc, a priori, une expansion lors du chiffrement, comme nous l'avons indiqué précédemment.

Pour le mode CBC, il existe une technique appelée « Ciphertext stealing » qui permet de légèrement transformer CBC pour ne plus avoir besoin de remplir le dernier bloc. Cette variante, appelée CBC-CS, consiste a une modification de traitement effectuée lors des deux derniers blocs, comme décrit à la figure 1.14.

FIGURE 1.14 – « Ciphertext stealing »



Le tableau 1.3 permet de lister les avantages et inconvénients de ces modes ; b est la taille du bloc. Dans le cas des modes CTR et CBC, les propriétés sont obtenues lorsque les IV sont choisis aléatoirement.

TABLE 1.3 – Propriétés de certains modes de chiffrement

	Limite d'indistinguabilité	Chiffrement parallélisable	Déchiffrement parallélisable	Précalculs possibles	Propagation des erreurs
ECB	1	oui	oui	non	non
CBC	$2^{b/2}$	non	oui	non	oui
CTR	$2^{b/2}$	oui	oui	oui	non

En pratique, on utilisera donc uniquement les modes CBC et CTR. Le mode ECB ne peut être utilisé que pour chiffrer des messages plus petits qu'un bloc, et même dans ce cas, une clef ne peut généralement être utilisée qu'une unique fois. Le mode CTR est fréquemment utilisé pour transformer un chiffrement par blocs en chiffrement à flot.

Le paramétrage par un IV

Dans la pratique, on souhaite généralement chiffrer des grands blocs indépendamment les uns des autres, comme par exemple dans le cas des disques durs où l'on désire chiffrer chaque secteur du disque séparément. De façon similaire dans le cas d'un protocole de communication, potentiellement téléphonique, on veut chiffrer des trames numérotées les unes après les autres, mais il ne faut pas que le déchiffrement de l'une dépende des autres trames. Cela nécessite un chiffrement parallélisable, par exemple un chiffrement à

flot ou un chiffrement par blocs utilisé avec le mode CBC avec des blocs de petite taille.

On a déjà vu que pour atteindre un niveau de sécurité IND-CPA, et en particulier pour assurer la non conservation de l'égalité, l'algorithme de chiffrement considéré doit soit être paramétré par un aléa, autrement dit randomisé, soit disposer d'un état interne, comme un compteur. Pour garantir cette propriété, on a recours à un IV, aussi appelé vecteur d'initialisation ou valeur initiale, pour paramétrer l'algorithme de chiffrement. Cela a pour effet de modifier la définition d'un schéma de chiffrement donnée initialement dans la section 1.1.1. Cette fois, les algorithmes de chiffrement $\mathcal{E}$ et de déchiffrement $\mathcal{D}$ dépendent d'un paramètre supplémentaire, l'IV. $\mathcal{E}$ devient ainsi déterministe, et l'expansion de message est ainsi évitée. Cela induit aussi un nouveau modèle d'adversaire. Ce nouvel attaquant choisit les messages et les IV, mais il ne peut appeler l'algorithme de chiffrement (respectivement de déchiffrement) avec un même IV qu'une unique fois. On demande donc alors que, quand la clef K est tirée au hasard uniformément, $\mathcal{E}_K$ appliquée à des messages de m bits soit indistinguable d'une fonction aléatoire sur $\{0,1\}^m$.

Seule la clef doit être secrète dans un algorithme de chiffrement d'après les principes de Kerckhoffs [Ker83]. Ce vecteur d'initialisation doit donc être déduit d'une valeur publique. Il doit aussi avoir une utilisation unique, c'est-à-dire qu'un message clair donné ne doit être chiffré qu'une unique fois avec le même IV. Cela assure que le chiffrement sera non déterministe et que deux messages identiques auront des chiffrés différents.

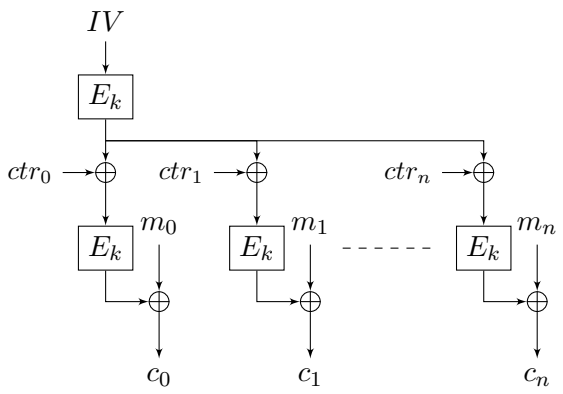
La façon de choisir l'IV varie en fonction des besoins. Il peut provenir d'une source d'aléa, ce peut être un compteur comme dans le mode CTR, l'adresse physique du bloc lors d'un chiffrement de disque dur, un numéro de version, il peut être fixé comme dans certaines fonctions de hachage ou certaines utilisations du mode CBC... Ce choix n'est pas anodin. Par exemple, le mode CBC est sûr quand l'IV est choisi au hasard, mais ne l'est pas quand il correspond à un compteur interne. Si l'IV provient d'une source d'aléa, il faut faire attention au nombre de chiffrements autorisés.

Il est aussi important de garantir l'intégrité de l'IV. En effet, un attaquant pourrait modifier l'IV associé à un chiffré dans les cas où l'IV n'est pas spécifié dans la norme mais un nonce généré spécifiquement. Si une telle altération devrait être remarquée car le message déchiffré sera vraisemblablement illisible, dans la version classique du mode CTR, vue à figure 1.13, il est possible de tronquer le message en modifiant l'IV en $IV \oplus ctr_i$ par exemple si les ctr_i sont consécutifs. Dans le contexte du mode CTR où l'on souhaite produire un chiffrement à flot et donc une suite pseudo-aléatoire, il est aussi attendu que la suite chiffrante générée soit indistinguable d'une suite aléatoire. Or H. Gilbert [Gil03] a montré que la version standard du CTR comme décrit à la figure 1.13 n'a pas non plus cette propriété, mais qu'il est possible d'adapter le mode CTR pour l'obtenir. La figure 1.15 présente un cas particulier du mode proposé dans [Gil03], où les rotations sur

les compteurs sont fixées à l'identité, cas autorisé dans la description de ce mode. La version modifiée du mode CTR est en l'occurrence celle utilisée dans les téléphones portables 3G sous le nom « Milenage ». Ce mode de chiffrement appliqué avec un chiffrement par bloc $(\mathcal{E}_K)_{K\in\mathcal{K}}$ parfait conduit à une fonction $\{0,1\}^n \times \sharp\{IV\} \to \{0,1\}^n$ indistinguable d'une fonction tirée uniformément au hasard dans l'ensemble des fonctions ayant les mêmes paramètres, comme cela est démontré dans le deuxième théorème de [Gil03]. Le mode CTR ne vérifie pas cette propriété. En effet, si on appelle $\mathcal{E}$ sur (IV, M_0, M_1) puis sur $(IV \oplus 1, M_1, M_2)$, alors les blocs d'indice 1 du premier chiffré et 0 du second chiffré sont identiques. La construction Milenage est elle immunisée contre cette attaque. Cette version modifiée permet aussi de vérifier l'intégrité de l'IV car l'attaquant n'a pas accès à $E_k(IV) \oplus ctr_i$.

Par contre, utiliser plusieurs fois le même IV peut avoir de lourdes conséquences. Le meilleur exemple de cela est lorsque que l'on chiffre deux valeurs en mode CTR avec le même IV dans le cas où l'on ne chiffre qu'un bloc de message. Dans ce cas, un attaquant aurait connaissance de $c_1 = m_1 \oplus E_K(IV)$ et de $c_2 = m_2 \oplus E_K(IV)$. De ces deux données, il peut immédiatement déduire $m_1 \oplus m_2 = c_1 \oplus c_2$, ce qui peut être un point de départ intéressant pour une attaque sur le chiffrement utilisé.

FIGURE 1.15 – Mode CTR, proposé par H. Gilbert [Gil03]



De la même façon, le mode CBC avec un IV présente des faiblesses identiques. En effet, deux couples (IV, M) et (IV', M') tels que les premiers blocs des messages, M_0 et M'_0 vérifient $M_0 \oplus M'_0 = \mathrm{IV} \oplus \mathrm{IV}'$ produisent des chiffrés ayant leurs premiers blocs identiques. Pour éviter ce type d'attaque, l'IV du mode CBC doit être imprédictible par un attaquant. On parle alors souvent de « sel » pour distinguer un IV imprédictible mais à usage unique.

Dans ces deux derniers cas, l'IV est bien déduit d'une valeur publique, mais il est modifié en une valeur imprédictible par l'attaquant en utilisant

le chiffrement sous-jacent et la clef. Une façon utilisée en particulier dans le chiffrement de disques pour atteindre ce but est ESSIV [Fru05] (« Encrypted Salt-Sector Initialization Vector »). À partir d'une fonction de hachage cryptographique h, de l'algorithme de chiffrement E utilisé, de la clef K et de la valeur publique dont est dérivé l'IV S, qui est le numéro du secteur dans le cas d'un chiffrement de disques durs, l'IV est calculé comme cela :

$$IV = E_{h(K)}(S).$$

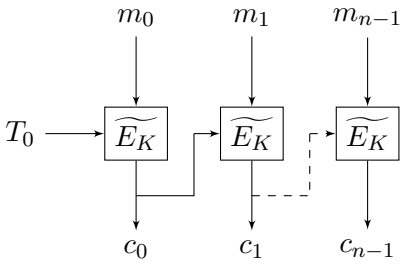
Les modes pour les chiffrements adaptables

La contrainte de ne pas pouvoir utiliser deux fois le même IV peut être non adaptée aux contraintes techniques, comme dans le cas du chiffrement de disque dur où l'IV ne peut simplement se déduire que du numéro de secteur, et où il n'est pas acceptable de n'utiliser qu'une unique fois un secteur donné. On veut alors une propriété plus forte que d'être indistinguable d'une fonction aléatoire dans un modèle où l'adversaire ne peut pas utiliser deux fois le même IV. On veut dans ce cas utiliser les contraintes de sécurité vues à la section 1.1.2. Plus précisément, on veut que, le schéma de chiffrement appliqué à des blocs de m bits soit un chiffrement par bloc adaptable, c'est-à-dire qu'il soit indistinguable d'une famille de permutations de $\{0,1\}^m$ paramétrées par un tweak, comme cela est décrit à la figure 1.10.

Nous avons donc vu à la section 1.1.2 des exemples de chiffrements par blocs adaptables. Pour pouvoir les utiliser, il faut aussi utiliser des modes opératoires. Ceux-ci, à cause du tweak, ne peuvent de fait être identiques aux modes de chiffrements classiques. Nous allons donc voir ici quelques modes permettant l'utilisation des chiffrements par blocs adaptables.

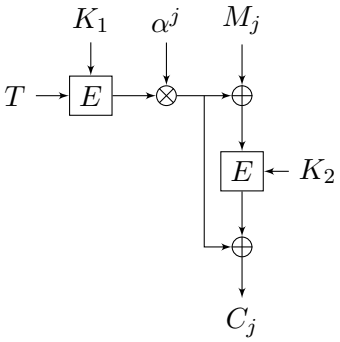
Le mode TBC [LRW02], initialement proposé pour LRW, fonctionne de façon similaire à CBC. En effet, TBC utilise le chiffré du bloc $i-1$ comme tweak du bloc i. Le premier tweak est lui le véritable tweak. Cela est décrit à la figure 1.16.

FIGURE 1.16 – Mode TBC



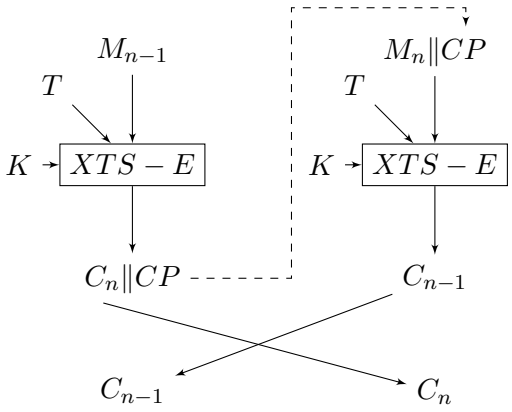
Le mode XTS (« XEX Tweakable Block Cipher with Ciphertext Stealing ») [IEE08, NIS10] est un mode proposé pour être utilisé avec une variante de XEX [Rog04] où deux clefs différentes sont utilisées, sans apport de sécurité[3]. Il permet en particulier de chiffrer des blocs incomplets, et ainsi de ne pas augmenter la taille du message chiffré. Son fonctionnement est décrit aux figures 1.17 et 1.18. La première illustre le chiffrement d'un bloc, et la seconde, celle des deux derniers blocs, dans le cas où le bloc M_n est un bloc partiel, c'est-à-dire qu'il est plus petit que les autres blocs. α est juste une constante, fixée à la valeur de l'élément primitif de $GF\left(2^{128}\right)$, qui est élevée à la puissance j, numéro du bloc, et multipliée à la valeur dans un corps fini. Dans la figure 1.17 on reconnaît à la fois le chiffrement de type XEX, et le mode de type ECB.

FIGURE 1.17 – Mode XTS



[3] http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/XTS_comments-Liskov_Minematsu.pdf

FIGURE 1.18 – Chiffrement des deux derniers blocs avec XTS



Ce mode a malgré tout des inconvénients qui résultent de l'utilisation d'un chiffrement adaptable sur des blocs de b bits dans un mode similaire à ECB. En effet, il est possible pour un adversaire de détecter qu'un bloc de b bits a été modifié à l'intérieur d'un message ce qui peut poser problème quand b est relativement petit. Il est aussi indispensable d'utiliser conjointement un contrôle d'intégrité sinon, un adversaire peut remplacer un bloc de chiffré par sa valeur précédente.

Par ailleurs, il faut distinguer deux types de modes de chiffrement opérant sur des chiffrements adaptables :

- « wide block tweakable encryption » (chiffrement adaptable sur des blocs larges),
- « narrow block tweakable encryption » (chiffrement adaptable sur des blocs étroits).

Le premier cas correspond au modèle où le bloc complet résultant du chiffrement, c'est-à-dire l'intégralité du message chiffré, se comporte comme un chiffrement adaptable. Dit autrement, la notion de sécurité des chiffrements adaptables s'applique dans ce cas au message complet et non plus juste à un bloc de b bits pris indépendamment. Le second cas impose que chaque bloc de b bits pris séparément soit chiffré d'une façon qui vérifie la sécurité d'un chiffrement adaptable, mais n'indique rien sur leur agencement global.

De plus, les modes TBC et XTS décrits ci-dessus tombent dans ce dernier cas. Il s'agit en effet de chiffrements dits « en ligne » [BBKN01], c'est-à-dire que le chiffré des i premiers blocs ne dépend que du clair des i premiers blocs. Cela peut être un inconvénient, par exemple car on peut détecter la modification d'un bloc, mais peut aussi être pratique, car on n'a pas besoin d'attendre d'avoir tous les blocs pour commencer à chiffrer, ou déchiffrer.

On peut alors considérer l'indistinguabilité par rapport à une famille de permutations en ligne, c'est-à-dire une famille de permutations suivant la définition précédente, notion de sécurité présentée dans [BBKN01].

D'autres propositions, comme CMC [HR03], XCB [MF04, MF07] (breveté) et EME [HR04], antérieurement breveté par l'Université de Californie Davis, existent. Elles sont toutes les trois du type « wide block tweakable encryption ». Elles permettent donc de construire une permutation pseudo aléatoire sur des grands blocs à partir d'un chiffrement par bloc ayant des petits blocs. En particulier, cela leur permet de combler un problème de LRW[4], mais en contrepartie, CMC et EME imposent un double chiffrement, ce qui dégrade fortement leurs performances. EME se décompose en cinq étapes. Une valeur L est calculée initialement comme $E_K(0^n)$, où 0^n est le mot de n bits dont tous les bits valent 0. Ensuite, les étapes sont les suivantes :

- XOR de $2^i L$, où la multiplication est effectuée dans le corps fini $\mathbb{F}_{2^b}$, au bloc de clair M_i,
- chiffrement de tous les blocs,
- calcul d'un nouveau masque L' et XOR de $2^{i-1}L'$ avec les blocs 2 à n, le premier étant traité séparément, c'est à cette étape que le tweak est ajouté par un XOR à chaque bloc ainsi généré,
- rechiffrement de tous les blocs,
- XOR de $2^i L$ au bloc i.

On remarque aussi qu'il faut parcourir trois fois tous les blocs de message. CMC a un fonctionnement similaire. EME a aussi deux variantes EME* et EME2. EME* [Hal04] vise à étendre EME pour gérer des messages de taille arbitraire avec des données associées. EME2 [IEE11] est une variante standardisée par l'IEEE. XCB a un fonctionnement différent qui ne requiert qu'un unique chiffrement, mais, en contrepartie, ajoute $2n$ multiplications dans $GF\left(2^{128}\right)$. XCB n'est donc pas significativement plus rapide que EME.

Ces constructions permettent donc de construire un chiffrement par bloc adaptable sur des grands blocs, typiquement 512 octets, à partir d'un chiffrement par bloc classique qui, lui, opère sur des blocs plus courts, par exemple 128 bits.

1.1.4 Protection de l'intégrité

Nous avons vu comment utiliser des outils de cryptographie symétrique pour protéger la confidentialité de données. Nous allons maintenant passer en revue quelques moyens d'en garantir l'intégrité. Les outils de base pour cela sont les fonctions de hachage cryptographiques qui calculent des empreintes

[4] http://grouper.ieee.org/groups/1619/email/msg00962.html

de données de taille quelconque, avec des propriétés que nous verrons plus tard. Elles peuvent être utilisées avec une clef secrète pour aussi prouver l'origine des données, c'est ce qui se passe avec certains codes d'authentification de message (MAC). Finalement, nous étudierons les modes permettant d'assurer confidentialité et intégrité des données.

Les fonctions de hachage cryptographiques

Définition. Une fonction de hachage est une fonction h qui associe au message m une empreinte de taille fixe $h(m)$. Plus généralement, c'est une fonction définie de la manière suivante, pour un n fixé :

$$\begin{array}{rccc} f: & \{0,1\}^* & \to & \{0,1\}^n \\ & m & \mapsto & h(m) \end{array}$$

Propriétés générales. Une fonction de hachage cryptographique doit vérifier les quatre propriétés suivantes pour présenter un intérêt :

- résistance aux préimages[5] : étant donné un haché h_0, il doit être difficile d'inverser h pour trouver un message m vérifiant $h(m) = h_0$;
- résistance aux secondes préimages : étant donné message m_0 et son haché h_0, il doit être difficile de trouver un message m, différent de m_0 tel que l'on ait $h(m) = h(m_0) = h_0$;
- résistance aux collisions : il doit être difficile de trouver deux messages m et m' tels que $h(m) = h(m')$;
- propriété de l'oracle aléatoire : la fonction de hachage doit se comporter comme une fonction choisie aléatoirement dans l'ensemble des fonctions de $\{0,1\}^* \to \{0,1\}^n$; cette propriété sert essentiellement pour les signatures et les chiffrements à clef publique dont la preuve de sécurité suppose que la fonction de hachage employée se comporte comme un oracle aléatoire.

Si l'on note n la taille des hachés de la fonction h, on peut évaluer la complexité des attaques génériques, c'est-à-dire qui s'appliquent à n'importe quelle fonction de $\{0,1\}^*$ dans $\{0,1\}^n$. Dans le cas de la recherche de préimages et de secondes préimages, la meilleure attaque générique a une complexité de 2^n calculs de h. Par contre, l'attaque par paradoxe des anniversaires permet de trouver une collision en $2^{\frac{n}{2}}$ calculs de h.

Dans le cadre des fonctions de hachage, ainsi que celui des chiffrements à flot [EJT07], il existe des attaques à IV choisis. L'idée principale de ces attaques est la suivante. Une fonction de hachage est souvent formée de

[5] Nous parlerons ici de préimages et non d'antécédents, pour rester plus proche de l'appellation anglaise.

plusieurs itérations d'une fonction de compression, qui prend en argument un bloc de message à hacher et soit la sortie de la fonction précédente, soit l'IV pour la première itération. Dans le contexte des fonctions de hachage utilisant une construction de Merkle-Damgård, l'intérêt principal est d'infirmer la preuve de sécurité. En effet, la sécurité de la construction de Merkle-Damgård repose sur le fait que la fonction de compression résiste aux collisions, qu'elles proviennent d'une différence sur la partie du message ou sur l'IV. De façon plus anecdotique, il peut aussi être intéressant de trouver une attaque en choisissant l'IV, car on pourra espérer ensuite produire cet IV à partir d'itérations de fonctions de compression en partant de l'IV imposé dans la définition de la fonction de hachage. On aurait alors une attaque sur la fonction entière.

Construction. Une construction classique de fonction de hachage cryptographique est d'itérer une fonction de compression, avant de procéder à une extraction du résultat. Les fonctions de compression f et de finalisation g peuvent se représenter comme suit :

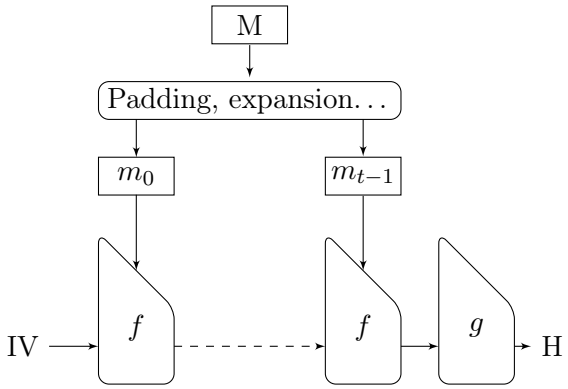
$$\begin{array}{lrl} f: & \{0,1\}^{n'} \times \{0,1\}^{n''} & \rightarrow \{0,1\}^{n'} \\ g: & \{0,1\}^{n'} & \rightarrow \{0,1\}^{n} \end{array}$$

où n, n' et n'' peuvent avoir des valeurs différentes. Une fois le message m rempli et découpé en t blocs de n'' bits, on obtient alors $h(m)$ de la façon suivante :

$$\begin{array}{lcl} h_{-1} & = & IV \\ h_i & = & f\left(h_i, m_i\right), \ \forall i \in \{0, t-1\} \\ h(m) & = & g(h_{t-1}). \end{array}$$

En 1989, Merkle [Mer89] et Damgård [Dam89] ont montré indépendamment qu'avec un IV bien défini, un remplissage approprié et une fonction de compression qui résiste aux collisions, la fonction de hachage construite sur ce modèle est aussi résistante aux collisions.

FIGURE 1.19 – Construction de Merkle-Damgård



De nombreuses fonctions de hachage sont construites sur ce principe. Mais il existe d'autres façons de procéder, en particulier la construction éponge. La construction de Merkle-Damgård elle-même se décline de plusieurs manières suivant la forme de la fonction de compression. Celle-ci peut être, par exemple, fondée sur un chiffrement par blocs E_K suivant différents modes [PGV93], en particulier :

- Davies-Meyer : $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}$,
- Matyas-Meyer-Oseas : $h_i = E_{h_{i-1}}(m_i) \oplus m_i$,
- Miyaguchi-Preneel : $h_i = E_{h_{i-1}}(m_i) \oplus h_{i-1} \oplus m_i$.

Exemples. MD4 [Riv91] et MD5 [Riv92] ont été conçues par Ron Rivest. Elles suivent toutes les deux la construction de Merkle-Damgård. Elles fournissent des empreintes de 128 bits et travaillent sur des blocs de 512 bits. Elles sont toutes les deux obsolètes. MD4 ne résiste plus aux attaques par collisions [Dob96] ni aux attaques par préimages [Leu08], et MD5 est aussi vulnérable aux attaques par collisions [WY05]. Sa résistance aux attaques par préimages n'est quand à elle plus que pratique.

Pour remédier aux attaques sur MD4 et MD5, le NIST, aidé par la NSA, a proposé SHA [FIP93], désormais appelé SHA0. SHA1 [FIP95] a en effet rapidement corrigé une faiblesse de cette fonction. Les empreintes générées par SHA0 et SHA1 font 160 bits. SHA2 [FIP02] est un ensemble de fonctions corrigeant des faiblesses de SHA1 avec des tailles de hachés plus élevées. Il est composé de 4 fonctions, SHA-224, SHA-256, SHA-384 et SHA-512, dont l'empreinte générée fait respectivement 224, 256, 384 et 512 bits. La famille SHA est basée sur les mêmes principes que MD4 et utilise la construction de Merkle-Damgård comme elle. Les fonctions SHA-224 et SHA-256 travaillent

sur une taille de bloc de 512 bits, et les fonctions SHA-384 et SHA-512 travaillent sur 1024 bits.

S'il y a des collisions pratiques sur SHA0 [BCJ+05] et théoriques sur SHA1 [WYY05], il n'y a actuellement aucune attaque par collision sur une fonction SHA2 complète.

À cause des faiblesses de ces fonctions, notamment de la famille SHA dont la conception est relativement proche de celle de MD4, le NIST a lancé en 2007 un concours [NIS] pour définir une fonction SHA3 basée sur d'autres concepts. Le résultat devrait être connu fin 2012.

Les codes d'authentification de message (MAC)

Définition. Un code d'authentification de message (MAC) est une famille de fonctions de hachage paramétrées par une clef secrète. Cela sert en particulier à authentifier l'origine de données. Généralement les MAC sont construits à partir d'autres primitives cryptographiques et les trois principales façons de faire cela utilisent soit des fonctions de hachage cryptographiques, soit du hachage universel, soit un mode propageant les erreurs avec un chiffrement par blocs.

MAC basé sur des fonctions de hachage cryptographiques. Le plus connu est HMAC (Hash-based Message Authentication Code) [BCK96]. La seule primitive requise est une fonction de hachage cryptographique h, par exemple SHA1 ou SHA2. Soit :

h : une fonction de hachage cryptographique,

n : la taille de la sortie de la fonction h,

K : la clef (soit remplie avec des zéros pour atteindre la taille n, soit hachée avec h si elle était trop longue),

M : le message à authentifier,

ipad : l'octet `0x36` répété $\frac{n}{8}$ fois,

opad : l'octet `0x5C` répété $\frac{n}{8}$ fois.

La RFC 2104 [HK97] définit alors HMAC comme suit :

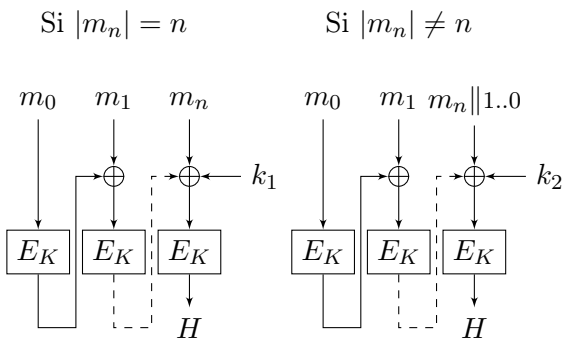
$$H\left(K \oplus \text{opad} \parallel H\left(K \oplus \text{ipad} \parallel M\right)\right).$$

MAC basé sur du hachage universel. Plusieurs MAC utilisent du hachage universel, comme par exemple UMAC [BHK+99] et VMAC [Kro06]. L'idée est ici de choisir une fonction de hachage au sein d'une famille de fonctions, le choix se déduisant de la clef.

MAC basé sur du chiffrement par blocs. Finalement, on peut utiliser un chiffrement par blocs pour générer un MAC. On a vu précédemment que certains modes opératoires, comme CBC, propageaient les erreurs lors du chiffrement. Ils peuvent donc être utilisés pour vérifier l'intégrité de données. La version la plus simple est CBC-MAC ; elle consiste à chiffrer le message rempli avec un 1 puis des 0 et utilise 0 comme IV. Si cette façon est sûre pour générer des MAC pour des messages de taille fixe, elle ne l'est pas pour des messages de taille variable, même en ajoutant la longueur du message dans le dernier bloc [BKR94, PvO95].

Il existe plusieurs variantes permettant de pallier ce problème : XCBC, OMAC, PMAC. OMAC est le nom générique pour OMAC1 et OMAC2 qui sont deux MAC basés sur un chiffrement par blocs différant très légèrement proposés par T. Iwata et K. Kurosawa. La version retenue par le NIST est OMAC1 sous le nom CMAC. Sa description est faite à la figure 1.20, où k_1 et k_2 sont deux sous-clefs distinctes déduites de la clef principale K. La version standardisée par le NIST est AES-CMAC [NIS05], où l'algorithme de chiffrement est AES.

FIGURE 1.20 – CMAC (appelé aussi OMAC1)



1.1.5 Le chiffrement authentifié

Définition.

Nous avons vu comment chiffrer des messages avec des outils de chiffrement symétriques, ainsi que les façons d'en garantir l'authenticité. Les modes de chiffrement authentifié consistent à combiner ces deux approches pour à la fois protéger la confidentialité d'un message et assurer son intégrité. La façon naturelle d'obtenir cela est de combiner un chiffrement et un MAC, mais il existe des modes dédiés qui permettent de meilleures performances et garantissent que la combinaison est bien sûre.

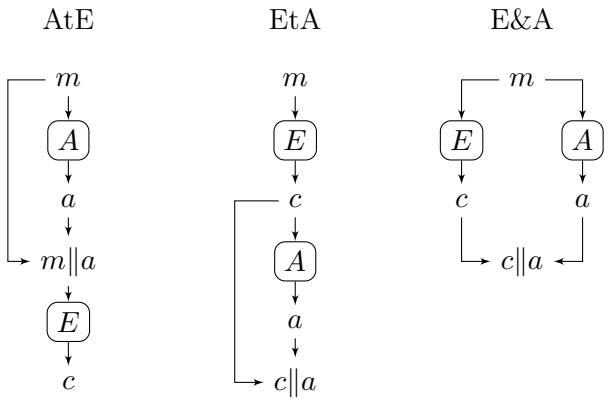
Les différentes possibilités.

Combiner un chiffrement et un MAC pour obtenir un chiffrement authentifié n'est pas du tout anodin. Pour choisir une telle combinaison, il faut à la fois choisir les outils de façon pertinente, mais aussi l'ordre dans lequel ils seront agencés. Il y a trois ordres possibles :

- authentifier puis chiffrer (AtE) :
  $a = A(m), c = E(m, a)$, on transmet c ;
- chiffrer puis authentifier (EtA) :
  $c = E(m), a = A(c)$, on transmet $c\|a$;
- chiffrer et authentifier (E&A) :
  $c = E(m), a = A(m)$, on transmet $c\|a$;

où m est le message clair, E est une fonction de chiffrement symétrique et A un MAC. La figure 1.21 illustre ces trois possibilités. Ces trois méthodes sont couramment utilisées, par exemple SSL (et TLS) utilise AtE, IPSec se base sur EtA et SSH suit E&A.

FIGURE 1.21 – L'ordre du chiffrement et de l'authentification



Ces trois façons ne sont pourtant pas équivalentes. Si on suppose que le chiffrement symétrique E n'est pas vulnérable aux attaques à clairs choisis et que le MAC A est résistant aux attaques à messages choisis, H. Krawczyk [Kra01] a montré les résultats suivants. Tout d'abord chiffrer puis authentifier (EtA) est sûr. Ensuite, authentifier puis chiffrer (AtE) est généralement non sûr, même si la combinaison mise en œuvre dans SSL n'est a priori pas mise en défaut. Finalement, chiffrer et authentifier (E&A) est aussi généralement vulnérable. Par contre, si on ajoute comme contrainte sur le chiffrement l'emploi du mode CBC ou d'un chiffrement à flot, E&A devient sûr.

Chiffrements authentifiés à une passe.

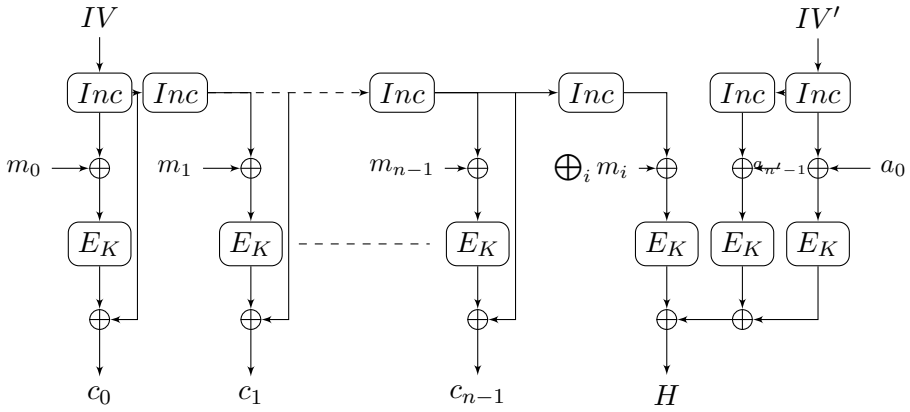
Une façon de procéder, utilisée dans de vieilles versions de Kerberos, est le mode PCBC (Propagating CBC ou Plaintext CBC). Il fonctionne de la façon suivante. Si $M = M_1, \ldots, M_n$, le i^{e} bloc de chiffré est défini par $C_i = E_K(M_i \oplus G_{i-1})$ où $G_0 = IV$ et pour i plus grand que 1, $G_i = g(M_i, C_i)$. PCBC était utilisé avec le XOR comme fonction g dans la quatrième version de Kerberos, mais n'est plus utilisé depuis par ce mode n'est pas sûr, bien qu'étant très efficace car un seul chiffrement est requis par bloc. En effet, la permutation de deux blocs consécutifs n'est pas décelable [Koh89]. Dans la section 9.6.5 du chapitre 9 du *Handbook of applied cryptography* [MvOV97], il est suggéré d'utiliser $g(a, b) = a + b \mod 2^m$, où m est la taille du bloc. Il est aussi conseillé d'ajouter une constante, par exemple l'IV, une valeur aléatoire ou M_1, à la suite du dernier bloc et d'utiliser le dernier bloc ainsi chiffré comme un MAC.

Offset Codebook Mode (OCB) [RBB03, Rog04, KR11][6] est un mode basé sur ECB proposé par P. Rogaway qui s'inspire en partie d'IAPM de C. Jutla. Ce dernier a aussi proposé IACBC, basé sur CBC, comme XCBC de V. Gligor et P. Donescu. Trois versions d'OCB existent, la seconde permettant la vérification de l'authenticité de données associées non chiffrées, la troisième améliorant les performances. Ce que nous appellerons simplement OCB dans la suite de ce manuscrit correspond à OCB3. Ce mode ne requiert qu'une seule passe, c'est-à-dire qu'un unique chiffrement est requis pour chaque bloc de message. Ce mode est considéré comme sûr et aucune version n'est sujette à des vulnérabilités connues.

Une fois le message M rempli et découpé en blocs $m_0, \ldots, m_{n-1}$ et les données associées A remplies et découpées en blocs $a_0, \ldots, a_{n'-1}$, OCB retourne les 96 premiers bits de H et le chiffré obtenu comme décrit à la figure 1.22.

[6] http://www.cs.ucdavis.edu/~rogaway/ocb/

FIGURE 1.22 – OCB



En 2010, G. Bertoni, J. Daemen, M. Peeters et G. Van Assche [BDPA10] ont proposé un mode de chiffrement authentifié en une unique passe basé sur les modes éponges. Cela permet d'avoir un mode à une passe non breveté. Par contre, ce mode est encore trop neuf pour avoir été pris en compte par le NIST.

Chiffrements authentifiés à deux passes.

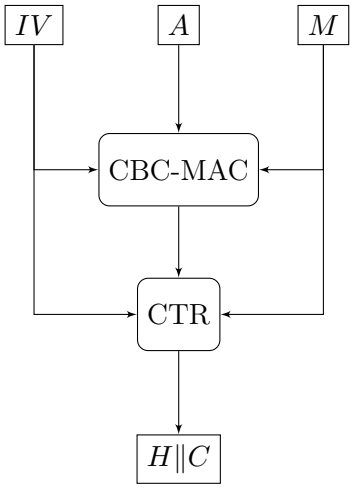
Le seul problème du mode OCB est qu'il est breveté. C'est pourquoi d'autres modes, à deux passes cette fois, ont été proposés ultérieurement. Ces modes gèrent aussi des données associées non chiffrées, ce que ne faisait pas la première version d'OCB. En effet, la gestion de données associées non chiffrées est extrêmement importante dans notre contexte puisque l'on pourrait souhaiter authentifier certaines valeurs publiques, comme par exemple les numéros de table, ligne et cellule ou de version, sans vouloir les chiffrer. Le mode proposé par le NIST est CCM-AES. GCM-AES est aussi normalisé pour des besoins particuliers. D'autres modes comme EAX et CWC existent aussi.

Encrypt-then-MAC [BN00, ISO09] est le mode le plus simple. Il ne gère en contre partie pas de données associées. Comme nous l'avons vu, ce mode est réputé sûr si le chiffrement et le MAC le sont [Kra01]. La seule contrainte est d'utiliser deux clefs indépendantes pour le chiffrement et le MAC.

Ainsi, Counter with CBC-MAC (CCM) [NIS04] est un mode AtE qui combine le mode CTR pour le chiffrement et CBC-MAC pour le MAC, comme cela est décrit à la figure 1.23. Deux chiffrements sont donc nécessaires par bloc de message. Par contre une unique clef est utilisée, ceci est possible car l'IV utilisé pour l'authentification ne collisionne pas avec les compteurs utilisés pour le chiffrement. En effet, le compteur utilisé dans le

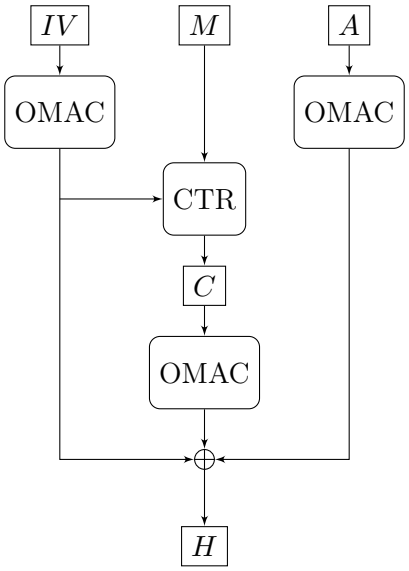
mode CTR initialisé par $(flag\|IV\|0\ldots0)$ où le premier octet donne notamment le nombre d'octets nécessaires pour coder la longueur en octets du clair M. L'origine du développement de ce mode se trouve dans l'existence de brevets pour OCB. La définition de CCM par le NIST assure que si l'on utilise des IV différents, l'IV utilisé pour le mode CBC et les compteurs utilisés pour le mode CTR seront tous distincts deux à deux, ce qui était la condition voulue.

FIGURE 1.23 – CCM



Le mode EAX [BRW04] (A Conventional Authenticated-Encryption Mode) est similaire à CCM, mais apporte deux propriétés intéressantes en plus. La première est que EAX peut traiter des données « on-line », en particulier sans en connaître la longueur par avance. La seconde est que EAX ne traite qu'une seule fois des données associées qui restent identiques au sein d'une même session. Cela est principalement dû à l'utilisation d'OMAC (décrit à la figure 1.20 à la page 37) en remplacement de CBC-MAC. EAX est décrit à la figure 1.24 où IV est le vecteur d'initialisation, M le message, A les données associées. EAX retourne $C\|H$.

FIGURE 1.24 – EAX



Le mode EAX' [MBPB11] est une mise à jour du mode EAX actuellement en cours d'étude pour une éventuelle normalisation par le NIST. Ce mode vise principalement à apporter quelques optimisations à EAX, sans en altérer la sécurité.

Galois/Counter Mode (GCM) [MV04, NIS07] a la particularité d'utiliser un autre MAC que CBC-MAC ou OMAC : le mode d'authentification de Galois. Celui-ci se calcule plus aisément qu'un CBC-MAC, et en particulier se parallélise mieux avec le chiffrement en mode compteur qu'un CBC-MAC. Cela permet d'avoir de meilleures performances. Par contre, plusieurs problèmes notables ont été soulevés concernant ce mode. Il a des clefs faibles, les MAC courts doivent être évités et les IV doivent vraiment être uniques. Les deux derniers points sont cruciaux dans l'utilisation de GCM.

CWC [KVW04] (Carter Wegman authentication with Counter encryption) peut être vu comme une variante de GCM utilisant le MAC de Carter Wegman. Ce mode a les mêmes avantages que GCM, mais sans ses vulnérabilités.

Le tableau 1.4, extrait de [KR11], liste les différents modes de chiffrements authentifiés basés sur des chiffrements par blocs existants.

TABLE 1.4 – Chiffrements authentifiés basés sur du chiffrement par blocs

Nom	Ref	Date	Nb	Description	Standard	B
EtM	[BN00]	2000	2	Encrypt-then-MAC	ISO 19772	Non
RPC	[KY00b]	2000	1	Ajout de compteurs et de constantes dans les blocs, puis ECB		
IAPM	[Jut01]	2001	1	Mode intégré qui a fait école, aussi IACBC	Oui	
XCBC	[GD01]	2001	1	Concurrent des travaux de Jutla, aussi XECB	Oui	
OCB1	[RBB03]	2001	1	Design optimisé, similaire à IAPM		Oui
TAE	[LRW02]	2002	1	Refonte d'OCB1 utilisant un chiffrement par bloc adaptable		
CCM	[NIS04]	2002	2	Chiffrement mode compteur + CBC MAC	NIST 800-38C	Non
CWC	[KVW04]	2004	2	Chiffrement mode CTR + CWC MAC basé sur GF($2^{127}-1$)		Non
GCM	[MV04]	2004	2	Chiffrement mode CTR + CWC MAC basé sur GF(2^{128})	NIST 800-38D	Non
EAX	[BRW04]	2004	2	Chiffrement mode compteur + CMAC, un CCM propre	ISO 19772	Non
OCB2	[Rog04]	2004	1	OCB1 optimisé et avec des données associées	ISO 19772	Oui
CCFB	[Luc05]	2005	1	Similaire à RPC, mais avec du chaînage		
CHM	[Iwa06]	2006	2	Sécurité au delà de la limite du paradoxe des anniversaires		
SIV	[RS06]	2006	2	Déterministe, résiste aux mauvaises utilisations	RFC 5297	Non
CIP	[Iwa08]	2008	2	Sécurité au delà de la limite du paradoxe des anniversaires		
HBS	[IY09b]	2009	2	Déterministe, une clef unique		
BTM	[IY09a]	2009	2	Déterministe, une clef unique		
OCB3	[KR11]	2010	1	Raffinement des versions précédentes d'OCB		Oui

Légende : Ref : référence
Nb : nombre de passes
B : breveté

1.2 Bases de données

1.2.1 Qu'est-ce qu'une base de données ?

Un système de gestion de base de données regroupe des informations, des programmes permettant d'y accéder, un environnement qui se veut agréable et efficace à utiliser.

Le but d'une base de données est d'agréger des données, par exemple, le nom des membres d'une association avec leurs coordonnées, les transactions effectuées par une banque, les horaires d'une compagnie ferroviaire, les informations fiscales d'une ville, d'un pays.

Si l'on traite de telles données à la main, sans l'aide de base de données, on risque d'être confronté aux problèmes suivants :

- données redondantes,
- données inconsistantes,
- données difficiles d'accès,
- formats non uniformes (fichier texte, pdf. . .),
- intégrité et cohérence difficiles à maintenir,
- non atomicité des mises à jour (si le disque est plein, une mise à jour peut s'arrêter au milieu),
- non gestion de la concurrence, ce qui peut aussi générer des incohérences,
- difficulté de gérer les droits des utilisateurs.

L'intérêt d'utiliser des bases de données est qu'elles permettent de pallier ces différents problèmes.

Les différents niveaux d'abstraction

Les bases de données proposent principalement trois niveaux d'abstraction.

Niveau physique. Le niveau physique est le niveau le plus bas, c'est lui qui décrit comment les données sont stockées. Ce niveau n'est réellement connu que par les développeurs du moteur de la base de données. Les autres utilisateurs ou développeurs d'applications utilisant la base de donnée n'ont pas à s'en préoccuper.

Niveau logique. C'est au niveau logique que l'on aperçoit les données ainsi que les relations dans la base. C'est le niveau où évoluent les développeurs d'applications utilisant la base de données.

Niveau visible. C'est le niveau de vision qu'ont les utilisateurs des applications utilisant la base de données. Ils ne connaissent ni les détails des types de données, ni toutes leurs relations. Il est aussi possible que certaines informations leur aient été cachées.

Les différents modèles

Modèle relationnel. Il y a de nombreux modèles de base de données. Un des plus utilisés est le modèle relationnel. Dire qu'une base suit ce modèle signifie qu'elle est structurée selon les principes de l'algèbre relationnelle. L'idée principale [Cod70] est de stocker les données sous forme de relations, ce que nous définirons plus tard. Les systèmes utilisant ce modèle sont appelés systèmes de gestion de base de données relationnels (SGBDR). Il en existe de nombreux exemples tels MySQL, SQLite, PostgreSQL et Oracle Database.

Modèle objet. Un autre modèle est le modèle objet, qui peut se combiner au précédent, comme dans le cas de PostgreSQL et Oracle Database. L'information est ici représentée par des objets, selon le même principe que la notion d'objet dans les langages orientés objets.

Modèle entité-association. Le modèle entité-association (« entity-relationship model ») concerne des descriptions de haut niveau de modèles conceptuels de données. Il décrit graphiquement les données sous forme de diagrammes contenant des entités et des associations. L'Unified Modeling Language (UML) en est un exemple. Il est structuré autour des notions d'entité et de relation. Une entité est une chose identifiable de façon unique, une relation décrit l'interaction entre deux entités. De plus chaque entité ou relation peut avoir des attributs l'identifiant.

Autres modèles. Il existe d'autres modèles, comme le modèle réseau, le modèle hiérarchique et le modèle semi-structuré qui ne comportent pas de séparation entre les données et le schéma.

Quels langages pour accéder aux données ?

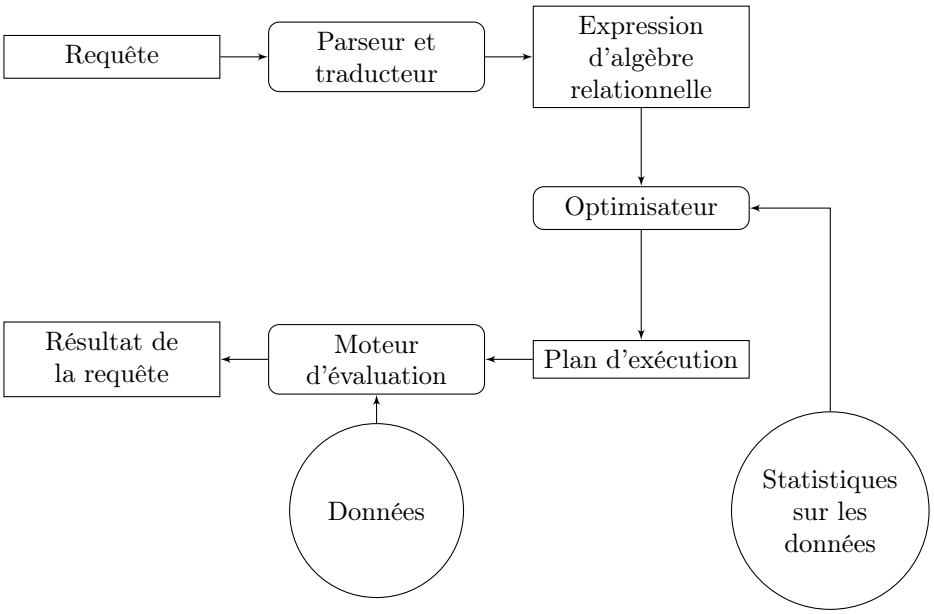
Un langage de manipulation de données consiste en l'ensemble des commandes permettant de manipuler une base de données. Il existe deux classes de tels langages. Dans les langages procéduraux, l'utilisateur précise à la fois les données requises et la façon dont elles doivent être manipulées, alors que dans les langages déclaratifs, l'utilisateur ne spécifie que les données requises,

mais rien ne concernant leur manipulation. Le langage le plus utilisé est SQL (Structured Query Language), qui est un langage déclaratif.

Gestion des requêtes

Pour effectuer une requête, il y a trois étapes principales : parser et traduire la requête, l'optimiser et finalement l'évaluer. L'évaluation de la requête est une étape critique car en fonction de la façon dont elle est faite, le coût de la requête peut varier énormément. Le processus d'exécution d'une requête est décrit à la figure 1.25.

FIGURE 1.25 – Processus d'exécution d'une requête



Structure générale

De nombreuses classes d'utilisateurs interagissent avec la base de données :

- programmeurs applicatifs, qui interagissent avec le système via le langage de manipulation de données,
- utilisateurs avancés, qui font des requêtes dans un langage de base de données,
- utilisateurs spécialisés, qui écrivent des applications non traditionnelles,
- utilisateurs naïfs, qui utilisent les programmes évoqués ci-dessus.

1.2.2 Le modèle relationnel

Définitions

Formellement, étant donnés des ensembles S_0, S_1, ..., S_{n-1}, une relation se définit comme un sous-ensemble du produit cartésien $S_0 \times S_1 \times \ldots \times S_{n-1}$, c'est-à-dire un ensemble de n-uplets $s_0, s_1, \ldots, s_{n-1}$ où, pour chaque $i \in \{0 \ldots n-1\}$, $s_i \in S_i$. Par défaut, les n-uplets à l'intérieur d'une relation ne sont pas ordonnés. Une base de données relationnelle est alors constituée de relations. Dans une telle base, une table est un ensemble de données organisées dans un tableau où les lignes représentent la relation et où les colonnes correspondent aux catégories, c'est-à-dire que les données correspondant à S_0 seront toutes dans la même colonne. La différence entre relation et table est assez ténue. Tout d'abord une table est composée de lignes, alors qu'une relation est composée de n-uplets. Mais la principale variation est qu'une table est ordonnée contrairement à une relation.

Par exemple, si on dispose des trois ensembles suivants :

- `lastname` = {Jacob, Villedieu, Dupont, Dupond},
- `firstname` = {Stéphane, Alice, Bob},
- `city` = {Paris, Lyon, Marseille, Chambéry},

`relation` = {(Jacob, Stéphane, Paris), (Jacob, Stéphane, Chambéry), (Dupond, Alice, Marseille)} est une relation sur `lastname` × `firstname` × `city`. On a l'habitude de représenter cela sous la forme décrite dans le tableau 1.5.

TABLE 1.5 – Exemple de relations

`lastname`	`firstname`	`city`
Jacob	Stéphane	Paris
Dupond	Alice	Marseille
Jacob	Stéphane	Chambéry

Les clefs

Définition générale. Une super-clef est un sous ensemble de $S_0 \times S_1 \times \ldots \times S_{n-1}$ qui permet d'identifier de façon unique chaque n-uplet d'une table donnée. Dans l'exemple précédent, {`lastname`, `firstname`, `city`} est une super-clef. Si on ajoute comme contrainte sur la table qu'il y a au maximum une ville par personne, {`lastname`, `firstname`} est alors aussi une super-clef. Parmi ces clefs, celles qui vont nous intéresser sont celles qui sont minimales, comme {`lastname`, `firstname`} avec l'hypothèse précédente. En l'absence de super-clef acceptable pour une table, il est habituel d'ajouter un attribut identifiant les n-uplets, généralement appelé `id`. De plus, il est préférable de

choisir des clefs invariantes au cours du temps, par exemple le nom d'une personne change moins souvent que son adresse.

Clef primaire. La clef primaire d'une table est une clef minimale choisie comme expliqué précédemment. Elle doit impérativement être unique.

Clef étrangère. Une clef étrangère permet de mettre en relation plusieurs tables d'une base de données. En effet, il est fréquent que plusieurs tables se réfèrent à un même élément défini dans une autre table. Indiquer alors qu'un attribut d'une table correspond à la clef primaire d'une autre en la déclarant comme clef étrangère permet de maintenir une meilleure cohérence dans la base. Par exemple, si, comme dans la figure 1.6, nous avons une table des utilisateurs avec leurs noms et prénoms, et une table de leurs adresses (principales, de vacances...), l'attribut `id` de la table des adresses sera une clef étrangère.

`lastname`	`firstname`	`city_id`
Jacob	Stéphane	3
Dupond	Alice	2
Jacob	Stéphane	1

`id`	`city`
1	Chambéry
2	Marseille
3	Paris

TABLE 1.6 – Exemple de clefs étrangères

Même si cela nous concerne peu ici, il est intéressant de savoir que les opérations sur les tables (sélection, insertion...) se font dans une algèbre, appelée algèbre relationnelle.

1.2.3 Index

Concepts de base.

Le but d'un index est simple : accélérer l'accès aux données de la table. Un index est constitué d'entrées de la forme {clef de recherche, pointeur}. Généralement, l'index est beaucoup plus petit que la table indexée. Il y a de nombreuses façons de faire un index. La méthode la plus naïve est de ne garder que l'attribut à indexer et les pointeurs vers les n-uplets associés. L'absence des autres champs de la table rend le parcours et donc la recherche plus rapide. Néanmoins, d'autres optimisations sont habituellement utilisées pour obtenir un meilleur gain. Les deux principales sont d'ordonner l'index ou de regrouper les clefs par paquets en utilisant une fonction de hachage (non nécessairement cryptographique ici).

Plusieurs paramètres permettent d'évaluer l'efficacité d'un index. Il peut ou non permettre des requêtes par intervalle. Les performances peuvent aussi varier d'un index à un autre, que ce soit pour les temps d'accès, d'insertion ou de suppression. La place prise par l'index peut aussi rentrer en compte.

Dans un index ordonné, comme son nom l'indique, les entrées sont ordonnées en fonction de la valeur de la clef de recherche. Cela peut permettre d'effectuer des requêtes en temps logarithmique en le nombre de valeurs prises par la clef de recherche, et également des requêtes par intervalle pour cette relation d'ordre.

L'index peut être dense ou creux. Un index dense contiendra toutes les clefs de recherche de la table indexée, alors qu'un index creux n'en contiendra que certaines. Un index creux n'a d'intérêt que si la table est séquentiellement ordonnée selon les clefs de recherche et bien qu'ayant une taille plus petite, il sera généralement plus lent qu'un index dense. Il est évidemment possible d'avoir un compromis, comme par exemple cumuler les deux, en ayant un premier index creux qui renvoie vers des index denses, ce qui peut être intéressant en particulier si l'index ne tient pas en mémoire, car on n'aura alors qu'à charger un des index denses.

Un index primaire, contrairement à un index secondaire, est un index dans lequel chaque attribut considéré apparaît au plus une fois. L'index secondaire est intéressant lorsque, par exemple, on veut obtenir la liste des habitants d'une ville dans un annuaire. Au lieu de pointer directement vers le n-uplet correspondant dans la table, la partie primaire d'un index secondaire pointe vers un ensemble de pointeurs, dirigés eux vers les n-uplets correspondant.

Index basés sur un arbre

Les index évoqués pour l'instant ont été essentiellement supposés séquentiels. S'ils ne sont pas ordonnés, il faut en parcourir la liste dans l'ordre pour trouver l'élément cherché. Dans le cas contraire, une recherche dichotomique est nécessaire, mais dans ce cas c'est la mise à jour de l'index qui devient coûteuse. Donc, même si ces exemples sont parfois utiles, il peut être avantageux d'utiliser d'autres sortes d'index dont le coût de recherche et de réorganisation sont moins élevés. En particulier, les deux inconvénients principaux des index séquentiels sont que leur performance se dégrade au fur et à mesure que leur taille augmente, et qu'il faut périodiquement les réorganiser. On préfère donc souvent utiliser des index construits sous forme d'arbre.

Il y a principalement deux types d'arbres utilisés pour construire des index : les arbres B et B+ [Com79]. Ce sont des généralisations des arbres binaires de recherche, qui, eux, ont un nombre de fils limité à deux [Knu98, CLRS09].

Arbres B+. Un arbre B+ est un arbre ayant les propriétés suivantes :

- toutes les feuilles ont la même profondeur,
- tous les nœuds internes ont entre $\lceil \frac{n}{2} \rceil$ et n fils,
- une feuille a entre $\lceil \frac{n-1}{2} \rceil$ et $n-1$ valeurs,
- si la racine n'est pas une feuille, la racine a au moins 2 fils,
- sinon, si la racine est une feuille (et donc le seul nœud), elle a entre 0 et $n-1$ valeurs,
- le dernier élément de chaque feuille pointe vers la feuille suivante,
- les nœuds sont ordonnés.

Les clefs de recherche sont ordonnées dans les feuilles de l'arbre. Les arbres B+ ne sont généralement pas profonds car pour stocker N valeurs, la hauteur de l'arbre est bornée par $\left\lceil \log_{\lceil \frac{n}{2} \rceil} (N) \right\rceil$. Par exemple, la figure 1.26 représente des données stockées dans un arbre B+.

Les arbres B+ ont de nombreux avantages. Il n'y a pas besoin de réorganiser l'index globalement. Toutes les mises à jour se font localement. Les arbres B+ ne requièrent pas l'allocation d'un gros bloc mémoire et peuvent être stockés en morceaux séparés. Finalement, les recherches y sont très efficaces.

Ici, l'insertion et la suppression d'une valeur sont un peu plus complexes que dans le cas des index séquentiels car il faut maintenir les propriétés de l'arbre. De même un arbre B+ induit un surcoût en taille. Mais ces inconvénients sont mineurs et n'empêchent pas leur très forte utilisation.

Les requêtes se font naturellement en descendant dans l'arbre jusqu'aux feuilles. En général, un nœud occupe un bloc du disque, soit 4 kilo-octets ; les entrées de l'index font typiquement environ 40 octets, d'où une valeur de 100 pour n. Cela veut dire que pour un index avec un million de clefs de recherche, au maximum 4 nœuds sont accédés lors d'une recherche.

Arbres B. Les arbres B se différencient des arbres B+ par le fait que les clefs de recherche ne peuvent apparaître qu'une unique fois, alors que les clefs des nœuds internes réapparaissaient systématiquement dans les feuilles pour les arbres B+. Donc les nœuds internes doivent pointer vers un ensemble de pointeurs ou directement vers un n-uplet dans la table. La figure 1.27 représente les mêmes données que celles de la figure 1.26, mais cette fois stockées dans un arbre B.

Les arbres B ont quelques avantages sur les arbres B+. Le nombre de nœuds utilisés peut être plus faible qu'avec un arbre B+. Il est de plus possible d'atteindre la clef cherchée sans avoir eu besoin de descendre jusqu'aux feuilles de l'arbre.

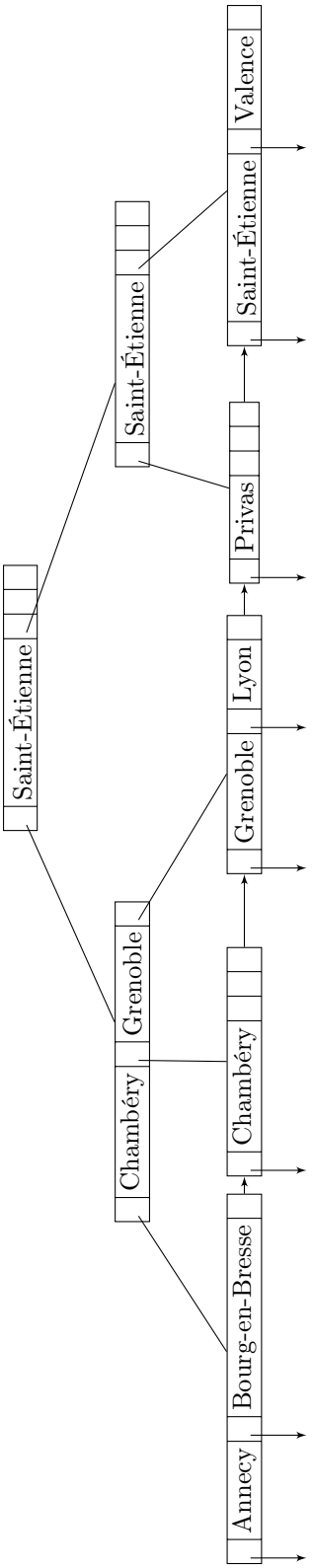


FIGURE 1.26 – Arbre B+ (avec $n = 3$)

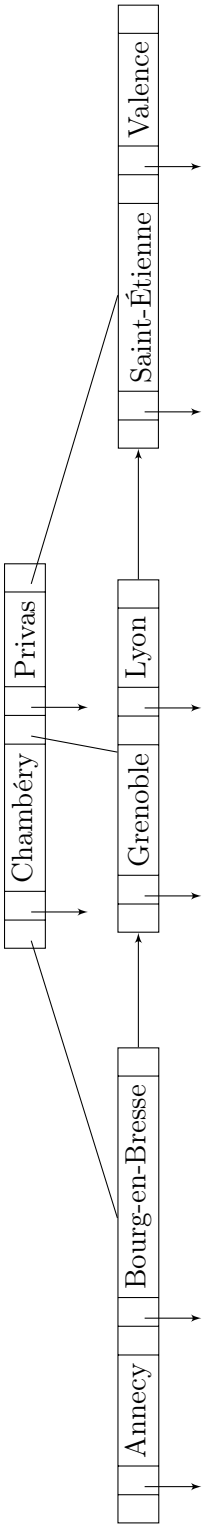


FIGURE 1.27 – Arbre B (avec $n = 3$)

Mais ce phénomène est très rare et entraîne une augmentation de la taille des nœuds internes. Cela complexifie aussi les opérations d'insertion et de suppression. Les arbres B+ sont donc généralement préférés aux arbres B car les inconvénients de ce dernier sont prépondérants sur ses avantages.

Hachage

Une autre façon de construire des index est d'utiliser des tables de hachage.

Hachage statique. Les outils dont nous aurons besoin pour cela sont les suivants. Tout d'abord, un paquet[7] est une unité de stockage contenant une ou plusieurs entrées. Il est généralement stocké sur un bloc de disque. L'indice du paquet contenant une clef se déduit directement de cette clef en lui appliquant une fonction de hachage h. Cette fonction de hachage est une fonction allant de l'espace de toutes les clefs de recherche vers les adresses des paquets. Il s'agit d'une fonction de hachage au sens classique en informatique : elle doit distribuer les valeurs de façon uniforme dans l'espace d'arrivée, mais aucune des autres propriétés des fonctions de hachage cryptographiques n'est pertinente ici. Au sein d'un même paquet, les données sont stockées séquentiellement.

Cette façon de gérer les index est principalement adaptée aux index secondaires. Un exemple d'index basé sur du hachage est présenté à la figure 1.28.

Si l'on choisit bien la fonction h, on peut diminuer la probabilité d'avoir un dépassement de paquet, mais jamais l'éliminer. Il faut donc se prémunir contre ce risque en prévoyant un mécanisme adapté le cas échéant. La meilleure façon de faire cela est de chaîner les paquets liés.

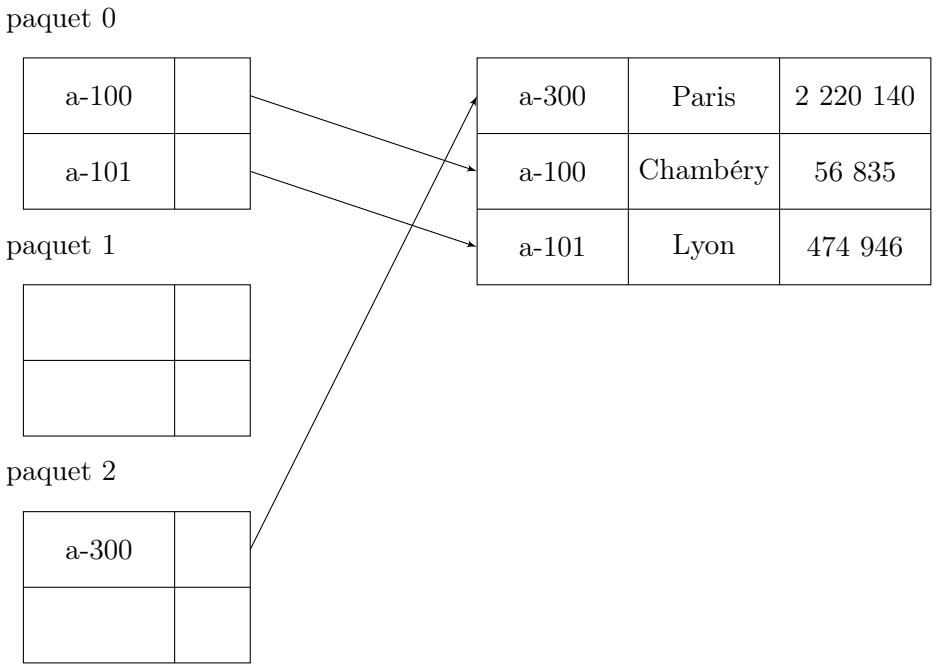
Le hachage statique est assez mal adapté aux bases de données car celles-ci évoluent et l'espace des paquets fluctue continuellement. Une réorganisation périodique permet de contourner ce problème, mais cela est trop coûteux en règle générale. Il est préférable d'autoriser la modification du nombre de paquets dynamiquement.

Hachage dynamique. Le hachage dynamique est très adapté aux bases de données où le nombre de paquets peut varier. La solution est d'autoriser la fonction de hachage à se modifier dynamiquement. Étant donnée une fonction de hachage dont l'espace image est $\{0 \dots 2^{32} - 1\}$, par exemple, on fixe le nombre de paquets à 2^i en commençant par $i = 0$. i évolue ensuite en fonction des besoins. Trouver le paquet correspondant à une valeur se fait en calculant le haché de cette valeur et en regardant ses i premiers bits.

Lors d'une insertion, s'il n'y a pas assez de place, on peut couper le paquet plein, et pour cela augmenter i. De même, si le nombre de paquets vides est grand, on peut diminuer i.

[7] Nous traduirons ici le terme anglais « bucket » par paquet.

FIGURE 1.28 – Index basé sur du hachage



La performance du hachage dynamique ne diminue pas quand la taille des tables augmente, de plus le surcoût en taille est minimal.

Par contre, il induit un niveau en plus de redirection, la table des adresses des paquets peut croître énormément, et modifier cette table est très coûteux.

Index bitmap

Les index bitmap sont principalement utilisés pour des requêtes mettant en jeu des clefs multiples pour des attributs prenant peu de valeurs distinctes. Un bitmap est simplement un tableau de bits contenant la présence ou non d'un attribut pour les différentes entrées. Plus précisément, un bitmap indique la présence ou l'absence d'une valeur dans chaque ligne d'une colonne donnée.

Si l'on considère la table 1.7 ci-dessous, nous obtenons les bitmap suivants :

- pour le champ `lastname` : Jacob : 101 ; Dupond : 010,
- pour le champ `firstname` : Stéphane : 101 ; Alice : 010,
- pour le champ `city` : Paris : 100 ; Marseille : 010 ; Chambéry : 001,
- pour le champ `gender` : m : 101 ; f : 010.

TABLE 1.7 – Table de données

`lastname`	`firstname`	`city`	`gender`
Jacob	Stéphane	Paris	m
Dupond	Alice	Marseille	f
Jacob	Stéphane	Chambéry	m

Filtre de Bloom [Blo70]

Un filtre de Bloom est une structure qui teste l'appartenance à un ensemble S de façon efficace. Le résultat de ce test n'est pas un résultat certain mais il vérifie les deux contraintes suivantes :

- pour toute recherche sur un élément appartenant à l'ensemble S, la réponse sera `true` ;
- pour toute recherche sur un élément n'appartenant pas à l'ensemble S, la réponse sera `false` avec une probabilité donnée.

Il peut donc y avoir des faux positifs, mais les faux négatifs sont écartés.

L'idée principale est la suivante. On dispose d'un tableau de taille m et de k fonctions de hachage h_i (pas nécessairement cryptographiques) qui ont pour valeurs possibles de sortie l'ensemble $\{1, \ldots, m\}$. Pour chaque élément s de l'ensemble S, la case $h_i(s)$ est mise à 1. Pour chercher si un élément x est dans l'ensemble S, on cherche alors si les cases $h_i(x)$ du tableau valent toutes 1.

Il est alors possible de généraliser les filtres de Bloom pour obtenir des structures similaires aux index. Par exemple, on peut concaténer plusieurs filtres obtenus pour des sous-ensembles de S. Une recherche sur ces filtres permet de restreindre le nombre de sous-ensembles où l'élément désiré pourrait se trouver.

Conclusion sur les index

Comparaison. Pour indexer un unique attribut, les deux principales possibilités sont donc les index ordonnés sous forme d'arbre et les tables de hachage. Le choix entre ces deux possibilités vient des spécificités de ce qui est indexé : les données sont-elles souvent mises à jour ? Faut-il privilégier le temps moyen d'accès ou le pire cas ? Souhaite-t-on faire des recherches sur la valeur des clefs ou sur des intervalles ?

En pratique, les principales bases de données (PostgreSQL, Oracle, SQL-Server, MySQL) ne supportent pas les tables de hachage, ou les déconseillent à cause de piètres performances.

Les index en SQL. Créer un index en SQL se fait aisément via la commande suivante :
`CREATE INDEX <index_name> ON <relation_name> (attribute_list)`
La suppression est également simple :
`DROP INDEX <index_name>`

Tout est ensuite géré par le moteur de base de données et les utilisateurs n'ont plus à s'en préoccuper.

Chapitre 2

Protection des bases de données

Pour les raisons évoquées dans l'introduction, les applications gérant des données qui sont stockées et manipulées à distance se généralisent. Cette externalisation ne doit pourtant pas se faire à n'importe quel risque et il faut donc protéger ces données. Cela doit se faire de deux façons. La première est d'assurer techniquement leur confidentialité et intégrité contre d'éventuels attaquants. Et il faut aussi encadrer juridiquement ces externalisations pour pouvoir se retourner contre un attaquant si les mesures techniques n'ont pas suffi, et aussi pour expliciter ce que l'hébergeur a le droit d'en faire. Cela est d'autant plus important pour des données sensibles, comme par exemple des données médicales, où il y a un danger encouru si elles tombaient entre de mauvaises mains. Une fois ces protections passées en revue, nous étudierons l'influence du niveau de centralisation sur la sécurité des données de santé.

2.1 Protection technique des bases de données

2.1.1 Les acteurs en présence

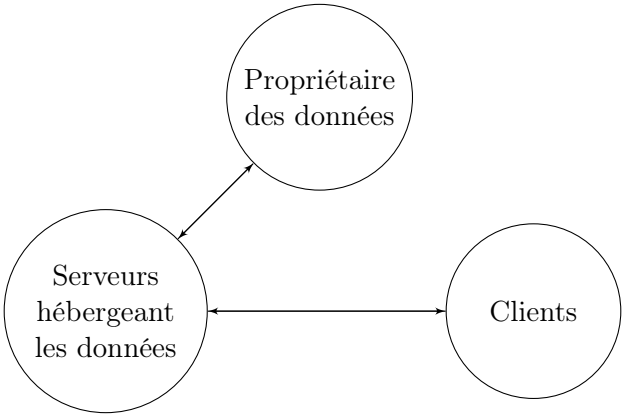
Pour se défendre contre un risque, la première étape est de l'identifier correctement. Cela implique, en particulier, de cerner les acteurs en jeu, de comprendre les motivations des attaquants et d'estimer leurs moyens.

Qui sont ces acteurs ?

Les acteurs peuvent être vus sous plusieurs angles. D'abord comme utilisateurs du système d'information, c'est-à-dire par rapport aux données et aux applications les manipulant. Ensuite, en fonction de leur rôle social. Dans le cas de l'hébergement de systèmes de santé, les deux acteurs principaux seront le patient et le médecin, mais ce ne sont pas les seuls.

Les acteurs d'un point de vue du système d'information. Dans un premier temps, nous allons décrire les acteurs d'un point de vue du système d'information. En particulier, nous allons détailler les différents accès et droits qu'ils possèdent. Le modèle DAS (« Data As a Service ») est constitué des entités suivantes. La figure 2.1 résume les interactions entre ces acteurs.

FIGURE 2.1 – Modèle « Data As a Service »



Le propriétaire des données produit et possède, a priori, les données. Ses capacités de calcul et de stockage sont, a priori, faibles comparées à celles du serveur de l'hébergeur. Ses accès aux données ne se font donc qu'à distance, et il n'a pas d'accès physique aux serveurs les hébergeant. Ses droits d'accès sont en fait limités par ce que le serveur lui laisse faire.

Les clients peuvent, ou non, être les propriétaires des données. Par exemple, dans une banque, la banque possédera les données et les personnes ayant des comptes dans cette banque seront les clients. Ici, les accès sont limités à des accès à distance et par les droits définis sur les serveurs, qui seront moins importants encore pour les non-propriétaires.

Les données sont externalisées, et donc stockées sur les serveurs d'un hébergeur de données tiers. Ces serveurs ont de grosses capacités de stockage et de calcul, que ce soit pour créer, stocker, accéder, ou manipuler des données. L'hébergeur dispose de plusieurs types d'employés dont les accès aux données diffèrent :

- l'employé de l'hébergeur : il n'a a priori aucun accès aux données hébergées,
- le technicien de l'hébergeur : il ne dispose d'aucun accès à distance, mais d'un accès physique lui permettant de réparer les machines en cas de besoin ; il a aussi a priori un accès de type « user » sur tous les serveurs,

- l'administrateur de la base de données : c'est lui qui gère les installations, réorganisations, migrations, mises à jour, sauvegardes ; il dispose de tous les droits imaginables sur la base, mais a priori pas ceux d'administrateur des machines à proprement parler,

- l'administrateur du serveur : lui a, a priori, tous les accès et droits sur les serveurs.

Ceci est bien évidemment très schématique et varie grandement d'une situation à l'autre. Mais cela permet de comprendre les différents types d'accès aux données, et donc les vulnérabilités qui y sont liées.

La place des acteurs de santé dans ce système. Dans le cadre des données de santé, il convient alors de déterminer la place des acteurs du système de santé vis-à-vis du système d'information :

- le patient : c'est lui le propriétaire des données ;

- le médecin : il est client des données, mais peut les éditer, en particulier, c'est lui qui a la charge de mettre à jour les données de ses patients ;

- les auxiliaires de médecine : eux aussi sont des clients des données ; en fonction de leurs rôles précis, ils peuvent aussi être des éditeurs des données, par exemple dans les hôpitaux, les personnels chargés de prendre quotidiennement des mesures (température. . .) pour suivre l'évolution des patients ;

- l'hébergeur : dans le cadre du DMP, ou du dossier pharmaceutique, il joue le rôle décrit précédemment ; dans des systèmes plus distribués, comme nous le verrons plus tard, l'hébergeur des données peut aussi être le patient, ou des hôpitaux.

Quelles sont les motivations des attaquants ?

Les motivations des attaquants peuvent être nombreuses et variées. Nous nous limiterons ici aux systèmes de santé. Malgré cela, nous ne pourrons qu'esquisser les idées principales, mais cela suffira à montrer qu'il faut se protéger de multiples façons. Les attaques peuvent essentiellement porter sur trois aspects : la confidentialité des données, leur intégrité et leur accessibilité.

Confidentialité : récupération de données. La première cible d'une attaque que l'on peut imaginer est sûrement une personne précise. Il est en effet probable qu'il puisse y avoir des intérêts à obtenir les données d'une personne. Les motifs peuvent être très variés. Un paparazzi sera motivé pour

avoir des détails concernant une célébrité qu'il suit. Pour un homme politique, un entrepreneur, un sportif ou autre, l'état de santé de ses adversaires est une information primordiale pour mieux appréhender leurs faiblesses.

Dans certains cas, ce sera plutôt des données d'un groupe de personnes indépendamment de leur identité qui seront visées. Une assurance a tout intérêt à connaître le passif d'une personne avant de l'accepter comme client, et va donc chercher à disposer d'une grande base de données. Un simple accès à des données statistiques ou aléatoires peut aussi être intéressant. Cela peut permettre à une assurance de mieux adapter ses tarifs, ou de faire des études épidémiologiques. Plus largement, une motivation peut aussi être de discréditer la sécurité du système d'information.

Intégrité : effacement ou modification de données. Prendre connaissance de données n'est pas la seule attaque envisageable. Falsifier ou supprimer des données peut aussi avoir un intérêt dans de nombreux cas.

Cela peut avoir des motivations multiples, mais principalement, ce sera pour cacher des erreurs. En effet, un médecin ou un pharmacien s'étant trompé de prescription, de médicaments peut vouloir le cacher. De même, un groupe pharmaceutique se rendant compte qu'un de ses médicaments est en fait dangereux pour la santé aura tout intérêt à essayer de dissimuler cela, soit en supprimant les données liées aux maladies consécutives à l'utilisation du médicament incriminé, soit en changeant la liste des personnes l'ayant pris.

Mais effacer des données choisies aléatoirement peut aussi se justifier de multiples façons. Par exemple, pour supprimer un sur-ensemble des données que l'on souhaite voir disparaître. Mais, cela peut aussi être fait dans l'unique but de nuire au système, ou de prouver qu'il est mal sécurisé, comme cela devient de plus en plus à la mode.

Fonctionnement du service. Finalement, même sans toucher directement aux données, il est possible de nuire au système informatisé de santé avec diverses motivations. D'une simple volonté d'exhiber une faiblesse du système à la possibilité d'éteindre le système à un moment opportun. Cela consiste généralement en un DoS, « Denial of Service », qui rend inutilisables les serveurs de l'hébergeur.

Quels sont les moyens des attaquants ?

Les moyens des attaquants peuvent être multiples car ils peuvent se situer à n'importe quel niveau des acteurs vus précédemment. De plus, un groupe d'attaquants n'est évidemment pas à mettre de côté. Leurs moyens peuvent alors raisonnablement excéder ceux de l'hébergeur.

Accès physique. Si l'attaquant a un accès physique aux données, cela lui donne un avantage indéniable car il pourra alors plus aisément contrecarrer les contrôles d'accès.

Accès distant. Même si un tel accès peut être suffisant pour mener des attaques, il requiert une mise en place plus compliquée, et cela nécessite aussi généralement de devoir passer un plus grand nombre de barrières.

2.1.2 Le contrôle d'accès

Le contrôle d'accès est le premier rempart contre les intrusions non autorisées. Il inclut l'authentification, l'autorisation et l'audit (traçabilité). Son but est de contrôler l'accès de sujets à un objet. Les sujets et les objets peuvent être tous les deux des entités logicielles ou matérielles, ou des humains.

Identification et authentification. L'identification permet de connaître l'identité présumée d'un sujet, par exemple son nom. L'authentification consiste à vérifier que cette identité correspond bien à ce sujet. Pour cela, il faut avoir préalablement créé et validé l'identité. L'authentification s'appuie généralement sur certains des points suivants :

- quelque chose connue du sujet, ce peut être par exemple un mot de passe,
- quelque chose possédée par le sujet, comme par exemple un jeton d'authentification ou une carte d'accès,
- une caractéristique du sujet lui-même, comme une empreinte, son iris, son ADN.

Autorisation. Cela détermine les droits du sujet sur le système. Par exemple, sur un système de type UNIX, les droits sur les fichiers sont la lecture, l'écriture, et l'exécution.

Audit, traçabilité. Les accès et actions des sujets sur le système doivent être répertoriés dans des journaux (« log ») qui doivent être correctement maintenus et suivis. Ils ont le double but de détecter des usages non autorisés, et de permettre de recréer des incidents pour mieux les comprendre. De plus, ils peuvent servir de preuve devant un juge.

Différents modèles de contrôle d'accès

Il existe deux grandes catégories de contrôle d'accès, les modèles discrétionnaires et les modèles non-discrétionnaires (MAC, RBAC) [Gol06].

Contrôle d'accès discrétionnaire (« Discretionary Access Control », DAC). Dans ce modèle, le propriétaire d'un fichier décide qui peut y accéder, et avec quels droits. Tout repose donc sur ce propriétaire.

Contrôle d'accès obligatoire (« Mandatory Access Control », MAC). Dans ce modèle, les accès sont déterminés par le système, et non pas par le propriétaire. L'idée ici est que les fichiers ont des niveaux d'accès minimum associés, et les sujets ayant les niveaux d'accès suffisant peuvent y accéder, ce droit ne dépend que des autorisations accordées à un sujet et nullement de son identité. Ici les niveaux sont hiérarchiques et suivent donc un ordre. Ce modèle est basé sur le modèle de sécurité Bell-LaPadula.

Contrôle d'accès à base de rôles (« Role-Based Access Control », RBAC). Ce modèle est aussi non-discrétionnaire, donc c'est encore le système qui décide ici des accès. Contrairement au cas du MAC, les droits sont ici associés à des rôles, ce qui apporte une grande flexibilité.

Limites du contrôle d'accès

L'importance des contrôles d'accès est évidente, mais il ne faut pas oublier qu'ils ne peuvent, à eux seuls, garantir la totalité de la sécurité d'un système. Pour nous convaincre, nous allons rapidement passer en revue quelques limitations.

Protection inefficace en présence de bug. La première faille potentielle dans un système de contrôle d'accès est simplement un bug dans le système gérant ces accès. Ce risque ne peut jamais être écarté comme en témoignent les « root exploit » sous Linux.

Protection inefficace contre le vol. Si les contrôles d'accès permettent la limitation des accès au système par le chemin numérique normal, ils n'empêchent pas le vol d'un disque dur par exemple. De plus, avec l'aide de systèmes sur CD ou clef USB, un accès physique à une machine garantit systématiquement un accès administrateur sur cette machine, même si cela peut engendrer des alertes.

Protection inefficace contre la destruction. Finalement, comme dans le point précédent, un accès physique peut permettre de détruire des disques. Certains types d'attaques de type « déni de service » peuvent aussi endommager des systèmes, sans y avoir accès.

On voit donc bien qu'il faut assortir ces contrôles d'accès d'autres types de protection pour obtenir un système réellement sécurisé.

2.1.3 Nécessité d'une protection cryptographique

Comme nous venons de le voir, le contrôle d'accès ne peut pas suffire pour protéger une base de données, en particulier si elle contient des données sensibles comme des données de santé. Il faut donc que la défense soit composée de plusieurs niveaux. Autrement dit, une défense en profondeur semble nécessaire pour assurer une bonne protection des données. Nous allons commencer par mieux définir ce concept, puis nous verrons ce que la cryptographie peut apporter en plus dans ce contexte.

Défense en profondeur

Le terme de défense en profondeur est emprunté au vocabulaire militaire, où cette technique consiste à limiter un risque en utilisant une grande variété de mesures de protection. La DCSSI (Direction centrale de la sécurité des systèmes d'information, maintenant ANSSI, Agence nationale de la sécurité des systèmes d'information) a réalisé un mémento [DCS04] en 2004 sur ce concept pour étayer la nécessité de s'en inspirer pour protéger des systèmes d'information. Le point de départ de cette étude est le constat suivant :

> « En matière de sécurité, dans le domaine des systèmes d'information comme ailleurs, le plus dangereux est bien souvent de se reposer, consciemment ou non, sur une fausse assurance. Une démarche saine serait de gérer l'incertitude, de maintenir une inquiétude raisonnée et d'entretenir une véritable vigilance. »

Le mémento [DCS04] situe l'origine de ce concept, dans le domaine militaire, à l'époque de Vauban. Les principes de la défense en profondeur ont évolué depuis cette époque et sont maintenant les suivants :

- le renseignement, première ligne de défense,
- une défense coordonnée et ordonnée,
- la perte d'une ligne doit affaiblir l'attaque et permettre de renforcer les autres,
- une ligne de défense doit être complète,
- défendre n'exclut pas d'attaquer.

Ces principes ont depuis été étendus plus généralement à d'autres domaines industriels.

Une des conséquences de l'accident de « Three Miles Island » du 29 mars 1979 a été l'apparition de l'utilisation de la défense en profondeur dans le domaine industriel, et en particulier dans le cadre de la sûreté nucléaire, où elle est constituée de trois barrières successives et indépendantes, ainsi que

de trois lignes de défense de natures différentes : pertinence de la conception, détection des défauts et des incidents, et limitations des conséquences.

De cela, le mémento [DCS04] déduit qu'il faut au minimum les points suivants pour que l'on puisse parler de défense en profondeur :

- il faut plusieurs lignes de défenses indépendantes, autonomes et complètes,
- ces lignes doivent coopérer,
- la perte d'une ligne doit renforcer la défense ; la défense est dynamique.

Application à la sécurité des systèmes d'information. Les principes généraux de la défense en profondeur sont énumérés dans le tableau 2.1, issu du mémento [DCS04].

TABLE 2.1 – Principes généraux de la défense en profondeur [DCS04]

Propriété	**Implications**
Globalité	La défense doit donc englober : • les aspects organisationnels, • les aspects techniques, • la mise en œuvre.
Coordination	Les moyens mis en œuvre doivent donc agir : • grâce à une capacité d'alerte et de diffusion, • à la suite d'une corrélation des incidents.
Dynamisme	Le SI doit disposer d'une politique de sécurité identifiant : • une capacité de réaction, • une planification des actions, • une échelle de gravité.
Suffisance	Chaque moyen de protection doit bénéficier : • d'une protection propre, • d'un moyen de détection, • de procédures de réaction.
Complétude	• Les biens sont protégés en fonction de leur criticité. • La défense est constituée d'au moins trois lignes. • Le retour d'expérience est formalisé.
Démonstration	• La défense est qualifiée. • Il existe une stratégie d'homologation. • L'homologation adhère au cycle de vie du système d'information.

Il découle de ce qui précède que protéger des données sensibles externalisées avec un simple contrôle d'accès n'est pas acceptable. D'où la nécessité

d'assurer la confidentialité et l'intégrité des données par d'autres moyens indépendants.

Confidentialité et intégrité

Pour la confidentialité, comme pour l'intégrité, il faut considérer à la fois les attaques à distance, et les attaques où l'attaquant aura un accès physique à la machine.

Protéger contre les accès à distance. La première mesure contre les accès à distance est celle vue précédemment : le contrôle d'accès. Ensuite, chiffrer (correctement) les données, ainsi qu'assurer leur intégrité est le minimum à faire, que ce soit pour le stockage ou pendant la transmission. Malgré cela, il se peut que l'on doive laisser des données en clair, ou au moins révéler quelques informations sur les données stockées, comme par exemple leur répartition. Normalement, ces informations ne seront visibles que de l'administrateur de la base de données.

Protéger contre les accès physiques. Si les protections précédentes sont nécessaires, il ne faut pas négliger ce qu'un accès physique apporte en plus comme possibilités à un attaquant. Tout d'abord, il pourra connaître tout ce qui n'est pas chiffré sur le disque : journaux, méta-données... Il est donc nécessaire d'utiliser un chiffrement de disque dur pour se prémunir contre la fuite de ces données. De plus, cela n'est pas non plus suffisant si l'attaquant peut avoir plusieurs accès aux ordinateurs dans le temps. Il pourra alors monter des attaques de type *Evil Maid Attack* [Rut09], qui sont plus difficiles à contrecarrer, mais dont la protection a pour principe d'utiliser soit un TPM, soit une clef USB pour vérifier l'intégrité des entêtes des disques.

2.1.4 Autres problèmes ouverts

De nombreux problèmes liés à la gestion des données de santé sont en fait ouverts et aucune solution existante pour les résoudre n'est totalement acceptable. Nous verrons plus tard ceux liés à la cryptographie qui sont le sujet de cette thèse. Nous nous concentrerons ici sur les autres problèmes ouverts, qui sont souvent des problèmes annexes qui ne s'appliquent pas à tous les cas d'externalisation de base de données, mais qui sont primordiaux pour les données sensibles, comme celles de santé. Ces deux principaux problèmes sont la destruction de données et l'anonymisation.

Destruction de données et droit à l'oubli

Lors de l'externalisation d'une base de données, il est important d'être capable de détruire des données en cas de besoin.

Mise à jour du matériel. Lors d'une mise à jour de matériel, il faut détruire correctement les données des vieux disques avant de les jeter ou de les recycler. Cela est assez aisé, car on peut détruire les entêtes du disque contenant la clef en récrivant suffisamment de fois dessus si le disque était préalablement chiffré ; sinon, il faut le faire sur le disque entier.

Changement d'hébergeur et droit à l'oubli. L'article L.1111-8 du Code de la santé publique, créé par la loi du 4 mars 2002, stipule que « lorsqu'il est mis fin à l'hébergement, l'hébergeur restitue les données qui lui ont été confiées, sans garder de copie, au professionnel, à l'établissement ou à la personne concernée ayant contracté avec lui ». Dans ce cas, il faut supprimer toutes les données concernant un utilisateur de façon irréversible, sans toucher aux autres. Actuellement, il restera toujours des traces des informations liées à cet utilisateur dans les méta-données de la base de données, et il n'est possible de les enlever complètement avec aucun système de base de données couramment utilisé. De plus, ces informations sont dupliquées dans certains hôpitaux, chez des médecins, et il peut y avoir des sauvegardes. D'où l'impossibilité d'implémenter de façon satisfaisante le droit à l'oubli. L'état de l'art de ce problème a été approfondi dans le rapport [ABC+10].

Anonymisation d'une base de données

L'anonymisation d'une base de données avant de la transmettre à un tiers est un souhait récurrent.

Sans possibilité de désanonymisation. Dans certains cas, on ne souhaite pas pouvoir « désanonymiser ». Par exemple, lorsque la Sécurité sociale transmet des données à des assurances pour qu'elles puissent calculer des statistiques, il n'est pas souhaitable qu'une désanonymisation soit possible. Supprimer les identifiants (nom, prénom, numéro de sécurité sociale…) n'est pas suffisant. En effet, des éléments à faible pouvoir d'identification lorsqu'utilisés seuls (code postal, âge…) peuvent se transformer en quasi-identifiants s'ils sont utilisés ensemble. Cela a été mis en évidence par Sweeney [Swe02] en 2002. Il s'en est suivit de nombreux travaux sur les différents niveaux d'anonymisation (k-anonymité, ℓ-diversité…), mais le résultat reste identique. Si l'anonymisation est complètement irréversible, les données anonymisées sont en pratique inutilisables ; il ne reste grosso modo plus que leur moyenne.

Avec possibilité de désanonymisation. Dans le cas de la recherche épidémiologique, il est nécessaire de pouvoir désanonymiser les données intéressantes pour pouvoir prendre contact avec des patients dont les cas sont dignes d'être étudiés de façon approfondie. Une solution serait d'ajouter un identifiant unique chiffré aux données anonymisées comme précédemment, mais l'absence de solutions dans le cas précédent rend cela inutilisable. Ceci est

donc un problème pour lequel nous ne disposons d'aucune solution concrète satisfaisante.

2.2 Exemples de gestion de données de santé

Comme nous l'avons déjà vu en introduction, le Code de la santé publique autorise, depuis la loi « Kouchner » du 4 mars 2002, l'hébergement de données de santé, mais précise que cet hébergement ne peut avoir lieu qu'avec le consentement exprès de la personne concernée dans un contrat d'hébergement :

> « Les professionnels de santé ou les établissements de santé ou la personne concernée peuvent déposer des données de santé à caractère personnel, recueillies ou produites à l'occasion des activités de prévention, de diagnostic ou de soins, auprès de personnes physiques ou morales agréées à cet effet. Cet hébergement de données, quel qu'en soit le support, papier ou informatique, ne peut avoir lieu qu'avec le consentement exprès de la personne concernée.[1] »

2.2.1 Exemples de systèmes de bases de données de santé

Le dossier médical personnel (DMP)

Le dossier médical personnel[2] est un projet lancé à l'initiative du ministère français de la santé via la loi du 13 août 2004 relative à l'assurance maladie[3], codifié dans l'article L.161-36-1 du Code de la sécurité sociale [L04b]. Il est encadré dans les articles L.1111-14 à L.1111-22 du Code de la santé publique. Le but principal premier était d'informatiser les données de santé des Français et ainsi de faire économiser de l'argent à la Sécurité sociale. Le projet n'a actuellement toujours pas complètement abouti, mais il a été relancé ces dernières années et l'hébergement du dossier a été attribué aux groupes Atos Origin et La Poste, au travers leurs filiales respectives SANTEOS et Extelia.

Le dossier pharmaceutique (DP)

Le dossier pharmaceutique (DP)[4] concerne les titulaires de la carte vitale. Il est encadré par l'article L.1111-23 du Code de la santé publique créé par la loi du 30 janvier 2007[5]. Il contient des données de santé car il a l'historique de tous les médicaments dispensés, avec ou sans prescription, pendant

[1] Article L.1111-8, alinéa 1 du Code de la santé publique

[2] http://www.dmp.gouv.fr/web/dmp/

[3] Loi n° 2004-810 du 13 août 2004 relative à l'assurance maladie

[4] http://www.ordre.pharmacien.fr/DP/index1.htm

[5] Article L.1111-23 du Code de la santé publique, créé par la loi n° 2007-127 du 30 janvier 2007

quatre mois. Ce sont les pharmaciens qui le remplissent lors de l'achat d'un médicament. Cela est fait dans plusieurs buts :

- lutte contre l'iatrogénie,
- lutte contre les redondances de traitements,
- amélioration du retrait des lots de médicaments,
- suivi thérapeutique,
- à plus longue échéance, réception immédiate des alertes sanitaires.

Les données sont centralisées chez l'hébergeur de données SANTEOS qui a été sélectionné par le Conseil de l'ordre national des pharmaciens.

Le dossier médico-social partagé (DMSP)

Le dossier médico-social partagé[6] vise à pallier les principaux défauts des dossiers papiers utilisés précédemment par les Centres Locaux d'Information et de Coordination Gérontologique (CLIC) du département des Yvelines. L'objectif est de corriger les inconvénients suivants :

- l'absence de confidentialité, car le dossier est consultable par tous les intervenants à domicile (médecins, infirmières, assistantes sociales, aides à domicile), ainsi que par la famille,
- la difficulté de partager des informations lors de consultations ou de soins hors du domicile du patient,
- la duplication de données non automatisée vers d'autres applications,
- l'absence de lien avec le dossier médical personnel.

La solution retenue fut donc des clefs USB sécurisées de type PlugDB[7] sur lesquelles sont stockées les données des patients. De plus SANTEOS, héberge une partie des données[8]. En fait, les données sont de quatre types possibles :

- régulières : présentes sur le serveur et sur la clef USB, accessibles via un contrôle d'accès sur le serveur,
- secrètes : présentes uniquement sur la clef USB,
- secrètes durables : présentes sur la clef USB et répliquées sous forme chiffrée sur le serveur, la clef USB étant nécessaire pour le déchiffrement,

[6] http://www-smis.inria.fr/~DMSP/dmsp_presentation.php
[7] http://www-smis.inria.fr/~DMSP/plugdb_presentation.php
[8] http://www-smis.inria.fr/~DMSP/dmsp_details.php?lvl=1_3

- restreintes : comme les données secrètes durables, mais déchiffrables par les membres d'un ensemble de personnes prédéfini.

2.2.2 Les moyens d'accès à ces bases de données

L'accès à ces systèmes de bases de données de santé est donné principalement par deux cartes : la carte vitale détenue par les patients et la carte professionnelle.

La carte vitale

Depuis 1998, les bénéficiaires de l'assurance maladie en France peuvent bénéficier de la carte vitale, conçue par le groupement d'intérêt économique SESAM-Vitale. Elle permet d'automatiser la gestion des remboursement de soins médicaux. Depuis 2007, il y a une mise à jour vers la carte vitale 2, qui apporte en particulier par rapport à la première version la clef d'entrée pour son dossier médical personnel, s'il apparaît un jour[9]. Cette clef d'entrée est constituée d'une clef privée permettant l'accès au DMP du porteur de la carte. L'arrêté du 14 mars 2007 relatif aux spécifications physiques et logiques de la carte d'assurance maladie et aux données contenues dans cette carte[10] définit les données stockées dans la carte vitale 2. Seules des données administratives y figurent, en particulier pour identifier le propriétaire de la carte et préciser ses droits vis-à-vis du système de santé.

La carte de professionnel de santé (CPS)

La carte de professionnel de santé (CPS) est une carte à puce électronique individuelle protégée par un code confidentiel qui permet d'assurer l'accès aux données médicales pour les professionnels de santé[11]. Elle est distribuée par l'ASIP Santé. Depuis 2010, la carte en vigueur est la carte CPS3. Elle contient de nombreuses informations[12], dont des données d'identification du porteur (numéro d'identification, noms patronymique et d'exercice du professionnel de santé, profession), des données relatives à l'activité, et des éléments techniques nécessaires aux fonctions de confiance (certificats de signature et d'authentification X509). Cette carte sert en particulier à chiffrer et signer les échanges de données relatives au remboursement d'actes médicaux[13].

[9] http://www.sesam-vitale.fr/divers/vitale2/carte_vitale2.asp

[10] http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000461627

[11] http://esante.gouv.fr/espace-cps/guide/la-cps-carte-d-identite-electronique-des-professionnels-de-sante

[12] http://esante.gouv.fr/espace-cps/guide/les-donnees-de-votre-carte

[13] Article L.161-33, créé par l'ordonnance 96-345 1996-04-24 article 8 II, IV JORF 25 avril 1996, http://www.legifrance.gouv.fr/affichCodeArticle.do?idArticle=LEGIARTI000006741272

2.3 Protection juridique des bases de données externalisées

Comme nous l'avons dit au début de ce chapitre, en plus d'une protection technique des bases de données, il faut une protection juridique. Le droit entourant cela peut être divisé en deux parties, la première générale aux technologies de l'information et de la communication (TIC), la seconde spécifique aux données de santé, qui, par leur caractère sensible, doivent être encadrées plus spécifiquement. L'annexe A donne quelques notions de base du droit des TIC qui s'applique évidemment aux données de santé.

2.3.1 Le droit spécifique aux données de santé

Nous allons ici voir les spécificités du droit des données de santé, ainsi que la façon dont le droit s'applique dans le cas du DMP. Pour cela, nous nous appuierons sur le rapport [Zor10] du projet DEMOTIS.

Les principaux textes de loi concernant les données de santé sont les suivants :

- la loi n° 2002-303 du 4 mars 2002 relative aux droits des malades et à la qualité du système de santé[14], dite loi Kouchner,
- la loi n° 2004-810 du 13 août 2004 relative à l'assurance maladie[15],
- la loi n° 2005-370 du 22 avril 2005 relative aux droits des malades et à la fin de vie[16],
- le décret n° 2006-6 du 4 janvier 2006 relatif à l'hébergement de données de santé à caractère personnel et modifiant le Code de la santé publique[17],
- la loi n° 2007-127 du 30 janvier 2007 ratifiant l'ordonnance n° 2005-1040 du 26 août 2005 relative à l'organisation de certaines professions de santé et à la répression de l'usurpation de titres et de l'exercice illégal de ces professions et modifiant le code de la santé publique[18],
- la loi n° 2007-199 du 14 février 2007 relatif à la carte d'assurance maladie et modifiant le Code de la sécurité sociale[19],

[14] JORF n° 54 du 5 mars 2002, page 4118, texte n° 1
[15] JORF n° 190 du 17 août 2004, page 14598, texte n° 2
[16] JORF n° 95 du 23 avril 2005, page 7089, texte n° 1
[17] JORF n° 4 du 5 janvier 2006, page 174, texte n° 14
[18] JORF n° 27 du 1 février 2007, page 1937, texte n° 1
[19] JORF n° 39 du 15 février 2007, page 2799, texte n° 26

- le décret n° 2007-960 du 15 mai 2007 relatif à la confidentialité des informations médicales conservées sur support informatique ou transmises par voie électronique et modifiant le Code de la santé publique[20],
- la loi n° 2009-879 du 21 juillet 2009 portant réforme de l'hôpital et relative aux patients, à la santé et aux territoires[21],
- l'ordonnance n° 2010-177 du 23 février 2010 de coordination avec la loi n° 2009-879 du 21 juillet 2009 portant réforme de l'hôpital et relative aux patients, à la santé et aux territoires[22],
- le décret n° 2010-1229 du 19 octobre 2010 relatif à la télémédecine[23],

Ces textes modifient principalement le Code de la santé publique et le Code de la sécurité sociale.

De plus, les données de santé nominatives sont naturellement des données à caractère personnel, et leur traitement est encadré par la loi informatique et liberté du 6 janvier 1978.

Le responsable du traitement.

La loi informatique et libertés du 6 janvier 1978 identifie un acteur ayant des responsabilités particulières : le responsable du traitement des données à caractère personnel, c'est-à-dire la personne ou l'autorité publique qui détermine les finalités et les moyens du traitement des données. La notion de traitement automatisé, au sens de la loi informatique et libertés inclut toutes les opérations sur les données notamment la collecte ou la conservation.

Le difficulté ici est de savoir qui est le responsable du traitement. Dans le cadre du Cahier des Clauses Techniques Particulières (CCTP) concernant le DMP [San09], il est stipulé que le responsable du traitement dans le cadre du DMP sera l'ASIP santé[24], qui endosse toutes les responsabilités associées à ce rôle. En particulier, c'est le responsable du traitement qui doit assurer la sécurité et la confidentialité des données.

L'hébergeur.

Le rôle et les responsabilités de l'hébergeur de données de santé est défini par l'article L.1111-8 du Code de la santé publique.

Dans le cadre du DMP, l'hébergeur est choisi par l'ASIP santé via le marché public intitulé « hébergement national des dossiers médicaux per-

[20] JORF n° 113 du 16 mai 2007, page 9362, texte n° 210
[21] JORF n° 0167 du 22 juillet 2009, page 12184, texte n° 1
[22] JORF n° 0047 du 25 février 2010, page 3585, texte n° 45
[23] JORF n° 0245 du 21 octobre 2010, texte n° 13
[24] Agence nationale des systèmes d'information partagés de santé

sonnels institués par l'article L.1111-14 du Code de la santé publique[25] ». Par ailleurs, le Code de la santé publique précise que cet hébergement ne peut avoir lieu qu'avec le consentement exprès de la personne concernée dans un contrat d'hébergement[26].

2.3.2 Les responsabilités des acteurs dans le cadre du DMP

L'accès au DMP par le professionnel de santé est subordonné, d'après l'article L.1111-15 du Code de la santé publique, à l'autorisation que donne le patient d'accéder à son dossier. En pratique, les autorisations sont définies dans une matrice d'habilitations[27] qui définit les types de documents auquel le professionnel a accès en fonction de son rôle (médecin généraliste, gynécologue, dentiste...). La loi interdit par contre au médecin du travail, ainsi qu'au médecin conseil d'assurance d'accéder au DMP (article L.1111-18 du Code de la santé publique).

Le CCTP pour l'appel d'offre pour l'hébergeur encadre l'accès classique au DMP[28] :

> « L'accès d'un professionnel de santé au DMP d'un patient (en alimentation ou en consultation) n'est possible que :
>
> 1. s'il est identifié et authentifié [...] ;
> 2. s'il dispose de l'autorisation d'accès donnée par le patient ;
> 3. s'il présente l'INS[29] du patient. »

L'autorisation est étendue à l'équipe de soin, en particulier dans le cas d'une hospitalisation, qui est définie dans le CCTP comme cela : « L'équipe de soins est constituée par l'ensemble des professionnels de santé intervenant autour d'un patient à l'occasion d'un séjour. » En effet, l'article L.1110-4 du Code de la santé publique stipule que « lorsque la personne est prise en charge par une équipe de soins dans un établissement de santé, les informations la concernant sont réputées confiées par le malade à l'ensemble de l'équipe. ». La mise en place de cela est pourtant difficile à faire cohabiter avec la matrice évoquée précédemment.

Le patient a le droit de masquer des données, c'est-à-dire de rendre inaccessibles certaines informations de son DMP. Ce droit lui est accordé par l'article 40 de la loi informatique et libertés du 6 janvier 1978, et est rappelé dans l'article L.1110-4 du Code de la santé publique. Néanmoins, le CCTP

[25] Référence 09-219810, BOAMP n° 198B, annonce n° 573 ; référence JOUE 2009/S 198-285000

[26] Article L.1111-8, alinéa 1 du Code de la santé publique

[27] http://dmp.gouv.fr/c/document_library/get_file?uuid=12a90ca5-9232-424e-8d9e-3438e15158f9&groupId=67702

[28] Page 48 de [San09]

[29] INS : identifiant national de santé

précise qu'un « document masqué reste accessible uniquement aux acteurs suivants, avec une marque précisant le masquage :

- au patient,
- à l'auteur du document,
- au médecin traitant,
- au DIM [Département d'information médicale] du GIP [Groupement d'intérêt public] en charge du DMP et au médecin du titulaire requis par le DIM, sur demande du patient lors de la communication par exemple du contenu de son DMP ou des traces fonctionnelles. »

Ce droit est prévu dans le CCTP, mais il suscite de nombreuses craintes, comme l'explique le rapport parlementaire de P.-L. Fagniez [Fag07]. En particulier, les deux principales récriminations sont :

- l'altération de la relation de confiance entre le patient et le professionnel de santé,
- la détérioration des soins pour le patient induite par une connaissance seulement parcellaire de son dossier par le professionnel de santé.

Il s'agit toutefois de la situation actuelle, en l'absence du DMP, dans la mesure où un patient peut, intentionnellement ou non, dissimuler des informations à un professionnel de santé.

Concernant les droits en écriture, l'article L.1111-15 du Code de santé publique indique que le professionnel de santé est, lorsqu'il prend en charge un patient, la personne tenue d'alimenter le DMP de ce patient.

La propriété intellectuelle.

Les bases de données bénéficient d'une double protection juridique. Leur contenu est protégé par l'article L.341-1 du Code de la propriété industrielle et leur contenant, c'est-à-dire leur structure, par le droit d'auteur. De cette façon, le producteur de la base est protégé contre toute extraction substantielle de celle-ci et la structure contre toute reproduction non autorisée. Dans le cas du DMP, la structure de la base est bien entendu conçue par l'ASIP santé. De même, étant donné l'investissement effectué par l'ASIP santé dans le DMP, celui-ci peut légitimement être considéré comme produit par l'ASIP santé.

2.3.3 Les acteurs de santé par rapport au droit des TIC

S'ils ne sont pas les seuls, les deux principaux acteurs de santé sont évidemment le patient et le médecin. Nous allons voir quelles sont leurs places dans les systèmes d'information de santé.

Le patient.

En fonction du système utilisé, le patient peut avoir plusieurs rôles : celui de personne concernée et celui de l'hébergeur.

Dans le cadre du dossier médical personnel (DMP), le patient est le propriétaire de données hébergées par un tiers, l'hébergeur de santé. Il n'est donc que la personne concernée dans ce mode fonctionnement.

Dans le projet de Dossier Médico-social Partagé (DMSP)[30,31], les données des patients sont stockées sur une clef USB qui leur appartient. Donc en plus de la qualité de propriétaire des données, les patients deviennent aussi hébergeurs de données.

Le professionnel de santé.

Le professionnel de santé peut lui aussi avoir plusieurs rôles : simple utilisateur, mais aussi hébergeur.

Son premier rôle est évidemment de consulter et d'alimenter les données de ses patients. Il est donc un utilisateur des données, mais pas leur propriétaire. S'il garde une copie des dossiers de ses patients, il peut devenir hébergeur. De même, actuellement, les hôpitaux ont fréquemment des dossiers médicaux dans leur système d'information propre, ils sont donc aussi des hébergeurs de données de santé.

Le fait qu'un même acteur puisse avoir plusieurs casquettes complexifie l'appréciation des textes de loi sur les données de santé. Cela a aussi pour effet d'offrir potentiellement plus de failles dans le système d'information de santé pour d'éventuels attaquants.

2.4 Où stocker les données de santé ?

Nous avons vu qu'il existe déjà plusieurs exemples de données de santé informatisées et que leur stockage variait d'un exemple à l'autre. Nous allons voir l'influence du type d'hébergement sur les menaces qui pèsent sur ces systèmes de santé, sur les mesures de protection à mettre en œuvre, ainsi que sur les responsabilités juridiques induites.

Dans tous les cas envisageables, les professionnels de santé ont le droit de lire et d'éditer certaines informations de leurs patients ou clients. Il en découle qu'un accès à un ordinateur d'un professionnel de santé ayant les droits pour accéder aux données des personnes visées suffit dans une certaine mesure à voler, voire parfois modifier ou supprimer, les données visées par

[30] http://www-smis.inria.fr/~DMSP/dmsp_presentation.php
[31] http://www-smis.inria.fr/~DMSP/plugdb_presentation.php

l'attaque. Ce risque, s'il ne doit pas être minimisé ni négligé, ne sera pas considéré ici, car nous nous concentrerons sur la sécurité des données chez l'hébergeur, et plus particulièrement celles stockées dans les bases de données de l'hébergeur.

2.4.1 De façon centralisée

Nous avons vu deux exemples de systèmes de santé où les données sont centralisées chez un hébergeur. Dans le cadre du dossier médical personnel, l'hébergement est assuré par le binôme formé par SANTEOS et Extelia, alors que pour le dossier pharmaceutique, il est fait par SANTEOS.

Les rôles vis-à-vis du droit des différents acteurs

Ici, nous avons le cas le plus classique où tous les acteurs sont bien différenciés. Le Cahier des Clauses Techniques Particulières [San09] concernant le DMP, le cahier des charges émis par le Conseil de l'ordre national des pharmaciens[32], ainsi que les lois précédemment étudiées définissent précisément les rôles. L'hébergeur est bien identifié, les responsables du traitement sont respectivement l'ASIP santé et le Conseil de l'ordre national des pharmaciens, le propriétaire des données est le patient, et les professionnels de santé sont des clients des données qui ont aussi pour charge de remplir le dossier. Ils endossent donc simplement les responsabilités prévues par les lois que nous avons passées en revue antérieurement.

Que craindre ?

Ce système a certains avantages indéniables. En particulier, les dossiers sont accessibles en permanence et de partout, ce qui est particulièrement intéressant pour mieux réagir et adapter les traitements en cas d'accident. Par exemple, la loi (article L.1111-17 du Code de la santé publique) prévoit explicitement que le médecin régulateur des appels d'aide médicale urgente puisse accéder au dossier médical des patients. De plus, les dossiers ne peuvent pas être perdus par un patient, et sont sauvegardés régulièrement par l'hébergeur.

Par contre, les dossiers sont exposés car accessibles en ligne. Le vol, l'altération ou la suppression de données ont la même difficulté pour un grand ensemble de patients que pour un seul ; plus précisément dans les deux cas, il faut s'attaquer à l'hébergeur.

[32] http://www.ordre.pharmacien.fr/pdf/appel-offre2-DP-cahier-des-charges.pdf

Comment se protéger

Évidemment, la première étape de la protection est le contrôle d'accès, après authentification des utilisateurs. Ensuite, un chiffrement des disques durs semble une étape nécessaire pour prévenir la divulgation d'informations en cas de vol. Pour prévenir l'altération de ces données, il faut aussi assurer l'intégrité des disques durs. Cela ne protège que contre des attaquants n'ayant pas d'accès aux ordinateurs de l'hébergeur. Pour se protéger de ces derniers, qui peuvent être des employés de l'hébergeur ou des attaquants ayant réussi à mettre en défaut le contrôle d'accès, il faut en plus de cela chiffrer les données hébergées, c'est-à-dire celles contenues dans la base de données, et assurer leur intégrité. Ce contrôle cryptographique peut au choix être effectué par l'administrateur de la base de données, ou par le propriétaire des données, ce qui a pour effet de protéger ces données même contre cet administrateur. En particulier, aucune information en clair ne doit donc être présente chez l'hébergeur.

De plus, il serait souhaitable que la compromission d'une clef ne compromette pas la sécurité de tous les dossiers hébergés. Cela implique que les dossiers des différents patients doivent être chiffrés avec des clefs différentes.

2.4.2 De façon décentralisée

Le stockage de données de santé de façon décentralisée est une alternative désormais envisagée par le législateur. En effet, l'article 30 de la loi n° 2011-940 du 10 août 2011 modifiant certaines dispositions de la loi n° 2009-879 du 21 juillet 2009 portant réforme de l'hôpital et relative aux patients, à la santé et aux territoires[33], dite « loi Fourcade », approuvé par le Conseil constitutionnel le 4 août 2011, introduit l'article L.1111-20 du Code de la santé qui institue la possibilité d'expérimenter jusqu'au 31 décembre 2013 « un dossier médical sur support portable ».

Nous avons déjà mentionné un exemple de ce type. C'est le dossier médico-social partagé où les données sont hébergées sur des clefs USB sécurisées par les patients. Il y a de plus, une sauvegarde hébergée par SANTEOS pour certaines données. Nous ne nous intéresserons ici qu'aux données décentralisées complètement, c'est-à-dire les données uniquement présentes sur la clef USB et non sauvegardées, même sous forme chiffrée sur le serveur.

Les rôles vis-à-vis du droit des différents acteurs

Dans le cas du DMSP, le patient est le propriétaire des données, mais le responsable du traitement est le conseil général des Yvelines, et c'est à ce titre qu'il lui incombe de faire les déclarations à la CNIL. Les acteurs de santé

[33] JORF n° 0185 du 11 août 2011, page 13754, texte n° 2

sont toujours des clients des données qui ont aussi pour charge de remplir le dossier.

Que craindre ?

Le système de santé est ici très différent. La première crainte semble raisonnablement l'absence de sauvegarde. Il suffit donc de voler une clef USB pour supprimer tout l'historique des données secrètes du patient. Une sauvegarde, comme celle effectuée pour les données secrètes durables semble être souhaitable. De même, le vol d'une clef USB et la possession d'un lecteur de carte suffit pour obtenir son contenu, sauf si son contenu est chiffré ou s'il y a un contrôle d'accès avec mot de passe.

Par contre, dans ce système, le vol, l'altération ou la suppression de données de plusieurs patients sont rendus difficiles par leur éparpillement. De même, il est impossible, sauf à récupérer toutes les clefs USB de tous les patients, de pouvoir, par exemple, effacer les traces de la prise d'un médicament.

Comment se protéger

Ici, la protection doit couvrir la confidentialité et l'intégrité des données présentes sur la clef USB. De plus, une sauvegarde, par exemple sous forme chiffrée, permettrait aux données d'être pérennes.

2.4.3 De façon mi-centralisée

Entre ces deux modèles, il est possible d'avoir un stockage centralisé localement. C'est par exemple le cas de Nadis[34], le logiciel de référence pour la prise en charge des patients infectés par le VIH ou une hépatite. Ici le serveur de données se situe au niveau d'un service de médecine, d'un hôpital, du réseau médical d'une ville. Il existe aussi des moyens de faire communiquer entre eux plusieurs serveurs Nadis.

Les rôles vis-à-vis du droit des différents acteurs

Ici, le patient redevient simple propriétaire des données. Les acteurs de santé deviennent eux hébergeurs et responsables des traitements, en plus de rester clients des données et d'avoir la charge de remplir le dossier.

Que craindre ?

Le système mélange les deux précédents. Les dossiers sont a priori accessibles aisément en cas d'urgence, et correctement sauvegardés. Par contre, ils sont accessibles en ligne, au moins au sein de l'hôpital ou de l'officine concernée,

[34] http://www.nadis.fr/

ce qui présente un risque évident. Les craintes sont donc similaires à celles du dossier centralisé.

Par contre, l'existence de plusieurs dossiers dans divers locaux rend une attaque contre un ensemble de dossiers plus dure à mettre en œuvre, sauf si le critère définissant l'ensemble est un critère géographique ou un critère médical, comme par exemple une pathologie soignée dans un hôpital ou un institut spécialisé qui concentrerait tous les dossiers ciblés.

Comment se protéger

Une protection similaire à celle évoquée dans le cas des serveurs centralisés est à mettre en place.

Chapitre 3

Confidentialité et intégrité des bases de données

Nous avons vu dans le chapitre précédent qu'il était primordial de garantir la confidentialité et l'intégrité des données de santé. Nous allons ici détailler les niveaux où la protection doit être implémentée, puis nous passerons en revue les solutions existantes. Finalement, nous détaillerons les exigences du chiffrement des bases de données, auxquelles aucune solution technique actuelle ne répond de manière suffisante.

3.1 Les différents niveaux de chiffrement des bases de données

La protection de la confidentialité et de l'intégrité se fait évidemment grâce à du chiffrement. La première question qui se pose alors est celle de l'endroit où le chiffrement et le déchiffrement doivent avoir lieu : au niveau des fichiers, des partitions et des disques durs ou au niveau de la base de données ou finalement au niveau applicatif ? Le chapitre de « Encyclopedia of Cryptography and Security » consacré au chiffrement des bases de données [BG10] différencie bien ces trois possibilités et leurs implications.

La figure 3.1, issue de [BG10], résume les trois stratégies décrites ci-après. Les tableaux 3.1 et 3.2 précisent eux qui a accès à la clef et aux données en clair, en fonction des niveaux de chiffrement.

FIGURE 3.1 – Les niveaux de chiffrement possibles [BG10]

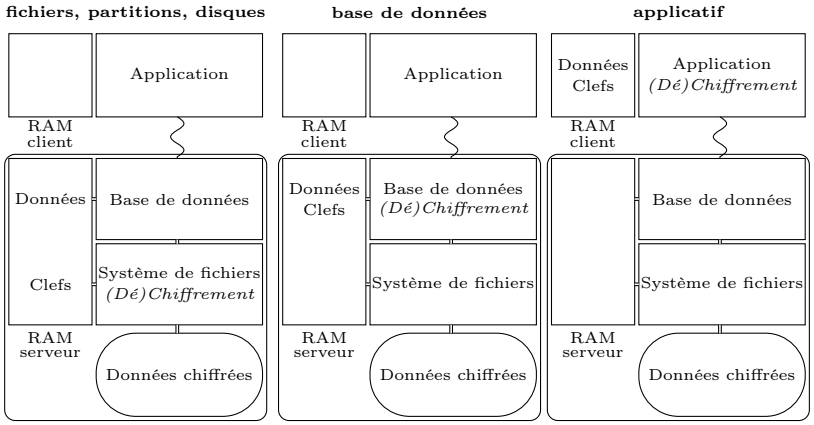


TABLE 3.1 – Qui a accès à la clef de déchiffrement ?

Niveau du Chiffrement	Serveur			Poste client
	adminis-trateur	administrateur de la BD	autre utilisateur	utilisateur
Disque dur	Oui	Non	Non	Non
BD	Oui	Oui	Non	Non
Applicatif	Non	Non	Non	Oui

TABLE 3.2 – Qui peut accéder aux données en clair ?

Niveau du Chiffrement	Serveur			Poste client
	adminis-trateur	administrateur de la BD	autre utilisateur	utilisateur
Disque dur	Oui	Oui	Oui	Oui
BD	Oui	Oui	Non	Oui
Applicatif	Non	Non	Non	Oui

3.1.1 Au niveau des fichiers, partitions, disques durs

Chiffrer et déchiffrer au niveau du stockage, c'est-à-dire du système de fichiers, des partitions ou des disques durs, signifie protéger les données statiques. Il y a principalement, deux façons de faire cela. La première est de le faire en dessous du système de gestion de fichiers, le chiffrement étant

alors directement géré lors de l'écriture des données sur le disque ; c'est ce qui est couramment appelé Full Disk Encryption. Dans ce cas, chaque bit stocké sur le disque est un bit de chiffré. Ceci peut se faire au niveau du disque, ou de façon plus souple au niveau d'une partition du disque dur. L'autre solution est d'utiliser un système de gestion de fichiers chiffré. Cela apporte une plus grande souplesse dans la gestion des accès aux données, et des clefs de chiffrement. Dans ce cas, les méta-données ne sont généralement pas chiffrées.

Cela est nécessaire pour protéger les données sensibles du disque, dont les bases de données, contre toute attaque impliquant un accès physique au disque, en particulier contre le vol. De plus, ce type de chiffrement est transparent et n'affecte pas les logiciels faisant appel aux bases de données, ni les bases de données elles-mêmes.

Mais ce type de chiffrement n'est pas suffisant dans le cadre des bases de données pour plusieurs raisons. La première est que ni le disque, ni le système de fichiers n'a connaissance de la structure de la base de données, ni a fortiori des droits associés à ses composants. Il n'est en particulier pas possible de limiter finement les accès aux données de la base en n'utilisant que ce type de chiffrement. Une autre limitation est que tout est chiffré, et qu'il n'est pas possible de ne chiffrer qu'une partie de la base. Finalement, si l'on choisit un système de gestion de fichiers chiffré, il faut faire attention à ce qu'on lui fait chiffrer. En particulier, il ne faut pas laisser des méta-données de la base de données, ou certains de ses fichiers temporaires en clair. De plus, un tel chiffrement ne permet pas d'utiliser des clefs différentes pour les différents patients.

Pour ce type de chiffrement, une clef utilisateur est demandée au montage du disque dur. La clef maître est généralement stockée sur le disque, chiffrée avec les clefs utilisateurs. Il faut donc au minimum qu'un administrateur connaisse une clef utilisateur, ou qu'il y ait un serveur dédié à la gestion des clefs. Mais de toutes façons, la clef maître sera stockée en mémoire vive (RAM) pendant l'utilisation du disque ou de la partition, et sera donc accessible aux administrateurs du serveur. De même, des données en clair peuvent se trouver en mémoire, et donc accessibles aux administrateurs. Par contre, la clef ne sera pas ainsi divulguée aux autres personnes ayant accès au disque.

3.1.2 Au niveau de la base de données

Le chiffrement et le déchiffrement au niveau de la base de données signifient que les données arrivent en clair dans le moteur de la base, et en ressortent de la même façon. C'est le moteur de base de données qui chiffre en stockant les données.

Cela a quelques avantages, en particulier cela permet de prendre en compte la structure de la base et le chiffrement peut ainsi être lié aux droits des utilisateurs, à la sensibilité des données. La granularité du chiffrement

peut aussi être adaptée : il peut avoir lieu au niveau des tables, des lignes, des colonnes, des champs, des index... De même, on peut alors choisir quelles méta-données doivent être chiffrées. Il est aussi possible de chiffrer en fonction de conditions logiques, par exemple, seulement les montants remboursés supérieurs à une valeur donnée. De plus, ce type de chiffrement n'affecte en rien les applications construites sur les bases de données.

Mais ce fonctionnement a au moins une grande limitation. Il faut adapter, voire créer un nouveau système de gestion de bases de données pour y inclure cette gestion du chiffrement. Ce nouveau système aura des performances détériorées par rapport à sa version standard ne supportant pas de chiffrement. Tout d'abord parce que le chiffrement prend du temps et qu'il faut généralement allouer de la place pour stocker les tags visant à assurer l'intégrité, mais aussi parce qu'il n'est pas aisé de créer des index sur des données chiffrées. Cette question particulière sera discutée ultérieurement. Néanmoins, nous verrons plus loin que, contrairement au niveau précédent, ce niveau permet une protection de l'intégrité plus aisée à mettre en place.

Finalement, ce niveau de chiffrement permet d'avoir une gestion des clefs plus fine que précédemment. Il est en effet possible, du moins théoriquement, d'avoir plusieurs clefs distinctes par base de données, et a fortiori des clefs qui diffèrent d'une base hébergée à l'autre, au contraire d'une clef étant unique pour le disque dans le cas précédent.

3.1.3 Au niveau applicatif

Le principe du chiffrement et du déchiffrement au niveau applicatif est de déléguer le chiffrement et le déchiffrement aux applications utilisant la base de données. Les données sont donc envoyées et récupérées chiffrées par les applications. La base de données et le serveur ne voient, ne manipulent et ne stockent donc que des données chiffrées.

Cela présente de nombreux avantages. Les clefs de chiffrement des données ne vont jamais sur le serveur, et donc même les administrateurs des serveurs n'ont aucun accès aux données en clair. De plus, en termes de gestion des clefs et de la granularité, on a ici la gestion la plus fine possible, car le chiffrement peut s'adapter à la logique de l'application.

Malgré ces nombreux points positifs, ce type de chiffrement possède quelques inconvénients. Il faut tout d'abord adapter les applications, ce qui n'est a priori pas un problème dans le cas du DMP, celles-ci n'existant pas encore. Les principales limitations sont en fait liées à la gestion des données chiffrées par le système de gestion de bases de données. En effet, créer un index sur des données chiffrées impose des contraintes, par exemple en terme de communications entre le système de gestion de base de données et le niveau applicatif qui effectue le chiffrement et le déchiffrement. De plus certaines fonctionnalités des bases de données ne pourront pas forcément être aussi efficaces que sur des données en clair, sauf à réduire la sécurité du

chiffrement.

D'un point de vue cryptographique, les chiffrements au niveau de la base de données et au niveau applicatif sont similaires. Leur différence réside dans le fait que chaque opération de chiffrement et de déchiffrement se fait localement sur le serveur dans le premier cas, alors qu'elle nécessite une communication, dans le second cas, ce qui est souvent rédhibitoire.

Dans l'absolu, il paraît donc conseillé d'utiliser une combinaison des premier et troisième niveaux de chiffrement pour protéger au mieux les données. Tout d'abord, un chiffrement au niveau applicatif permet de ne donner en clair à l'hébergeur que des données non sensibles, les méta-données de toutes les données, chiffrées ou non, et potentiellement quelques informations qui seraient divulguées si l'on utilise des chiffrements adaptés permettant de meilleurs performances au niveau base de données. Par ailleurs, ces informations ne seraient visibles que par les administrateurs du serveur, voire les administrateurs de la base de données. Si le chiffrement ne peut pas être mis en œuvre au niveau applicatif, il peut néanmoins être remplacé par un chiffrement au niveau de la base de données. Ensuite, un chiffrement de type Full Disk Encryption permet de protéger l'ensemble des données, méta-données incluses, d'un vol ou de toute autre corruption issue d'un accès physique aux disques. En particulier, même les informations connues des administrateurs ne pourront être obtenues par un vol d'un disque par un non administrateur.

3.2 Solutions existantes

Avant de se pencher sur l'état de l'art de la recherche sur le chiffrement de bases de données, nous allons d'abord étudier les solutions commerciales existantes pour protéger un serveur de bases de données, du point de vue cryptographique. La première étape de la protection est effectuée par des cryptoprocesseurs. Ensuite vient le chiffrement des disques durs, et, finalement, celui des bases de données.

3.2.1 Cryptoprocesseurs et protection des clefs

Trusted Platform Module (TPM)

Lors du démarrage d'un serveur, il faut tout d'abord s'assurer que le matériel est bien le bon, et que les logiciels sont aussi bien ceux que l'on veut lancer. Autrement dit, au démarrage d'un serveur, il est intéressant de pouvoir certifier que le matériel et les données logicielles des disques durs de ce serveur n'ont pas été altérés. C'est le but du Trusted Platform Module (TPM), qui décrit aussi bien la spécification du cryptoprocesseur chargé des

certifications, que de ses implémentations. La spécification du TPM a été élaborée par le Trusted Computing Group (TCG)[1] qui est un consortium regroupant de très nombreuses entreprises[2] dans le but d'élaborer des éléments informatiques sécurisés. Le but principal du TPM est d'assurer l'intégrité de l'ensemble de la chaîne de démarrage (BIOS, MBR, Boot Loader, hyperviseur, noyau, initrd, modules, systèmes de fichiers, fichiers de configuration, applications).

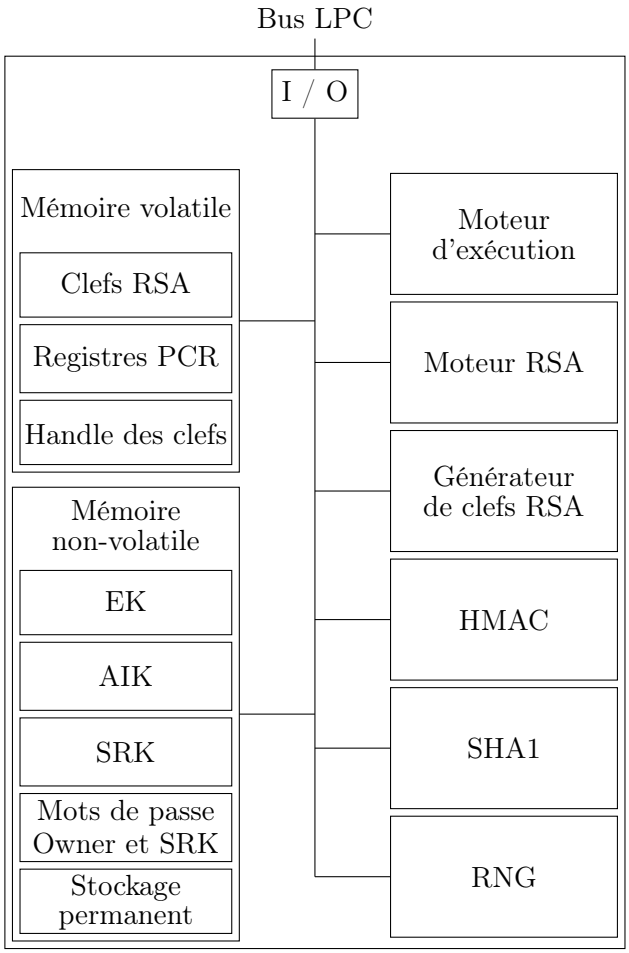
Le TPM est une puce matérielle cryptographique. Dans le but d'attester de l'intégrité de l'architecture matérielle et logicielle d'un ordinateur, elle est composée de fonctions cryptographiques de base (générateur d'aléa, générateurs de clefs, fonction de hachage, algorithme de chiffrement et déchiffrement asymétrique, générateur de MAC) et de stockages sécurisés. Comme décrit dans [GGR10], dont nous avons repris la figure 3.2, ses composants sont les suivants :

- mémoire non-volatile : elle contient les données permanentes suivantes :
  - clef d'approbation (EK, « endorsement key ») : c'est la paire de clefs RSA générée par le constructeur du TPM,
  - clef racine de stockage (SRK, « storage root key ») : c'est une paire de clefs RSA qui joue le rôle de clef racine dans un système de hiérarchie des clefs classique,
  - secret d'authentification du propriétaire (OAD, « owner authorization data ») : c'est une valeur de 160 bits qui constitue un mot de passe commun entre le composant et le propriétaire du TPM,
  - module d'attestation des clefs d'identités (AIKs, « module attestation identity keys ») : c'est le module (optionnel) de stockage des clefs AIK, paires de clefs utilisées lors du protocole d'attestation de la plateforme,
- mémoire volatile : elle contient les données temporaires,
  - registres PCR (PCR, « module platform configuration register ») : ce sont des registres de 160 bits permettant de stocker l'état d'une plateforme ; le TGC en impose un minimum de 16, les 8 premiers réservés pour l'usage interne du TPM, les autres dédiés aux applications (Boot Loader...),
  - toutes les clefs (RSA...) non stockées dans la mémoire non-volatile ;
- générateur d'aléa (RNG) physique,
- module SHA1 ,

[1] http://www.trustedcomputinggroup.org/
[2] http://www.trustedcomputinggroup.org/about_tcg/tcg_members

- module HMAC,
- génération des clefs RSA : module permettant la génération de clefs RSA jusqu'à 2048 bits,
- module RSA : module de chiffrement et de signature RSA.

FIGURE 3.2 – Composition d'une puce TPM [GGR10]



Par exemple, le TPM d'IBM[3] contient les éléments suivants :

[3] http://domino.research.ibm.com/comm/research_projects.nsf/pages/gsal.TCG.html

TABLE 3.3 – Contenu du TPM d'IBM

Unités fonctionnelles	**Mémoire non volatile**	**Mémoire volatile**
Générateur de nombres aléatoires	Clef d'approbation (2048 bits)	10 emplacements pour clefs RSA
Fonction de hachage cryptographique	Clef racine de stockage (2048 bits)	16 PCRs
HMAC	Secret d'authentification du propriétaire (160 bits)	Gestion des clefs
Générateur de clefs RSA	Gestion des sessions d'authentification	
Chiffrement/ déchiffrement RSA		

L'exposé à la conférence SSTIC 2010 consacré au TPM [GGR10] détaillait aussi les services offerts nativement par le TPM :

- unicité de la plate-forme, via la clef unique EK et la certification de la partie publique de cette clef par une autorité de certification associant le TPM à un certificat unique,
- stockage sécurisé : le TPM intègre un mécanisme de protection de la confidentialité des données qui repose sur une hiérarchie de clefs dont la racine est la clef SRK,
- scellement de données : c'est un mécanisme de stockage sécurisé basé sur l'état de la plate-forme ; le descellement d'une donnée ne peut se faire que si l'état des registres PCR au moment du scellement et du descellement sont identiques,
- opérations cryptographiques (SHA1, génération de clefs, chiffrement et signature RSA, génération d'aléa),
- extension des registres PCR : ce mécanisme, qui permet d'agréger des mesures réalisées par des logiciels tiers dans les registres PCR du TPM, sert à l'attestation du poste client. La mise à jour d'un registre PCR est simple :

$$\mathrm{PCR}_{t+1}[i] = \mathrm{SHA1}\left(\mathrm{PCR}_t[i] \parallel \mathrm{SHA1}\left(\mathrm{DATA}\right)\right),$$

  où i désigne l'index du registre concerné par l'extension.

En particulier, un TPM sert à assurer l'intégrité de l'ensemble de la chaîne de démarrage, ce qui se fait via les chaînes de confiance CRTM (Core Root Trusted Measurement) et DRTM (Dynamic Root of Trust Measurement). Le

premier lie les éléments critiques du démarrage de la machine précédemment cités. Si l'on note $\{\mathcal{E}_i\}_{0\leq i\leq n-1}$ ces différents éléments, lorsque c'est au tour de l'élément $\mathcal{E}_i$ de se lancer, il effectue une mesure de l'élément $\mathcal{E}_i$, c'est-à-dire le calcul d'un haché SHA1 de cet élément, et il l'écrit dans un registre PCR, via l'extension vue précédemment. Cette valeur est ensuite comparée à celle stockée lors du scellement. Si ces deux valeurs sont identiques, le processus continue. Le DTRM a pour but de créer une chaîne de confiance indépendante de la plateforme matérielle et elle peut être initiée lorsque le système d'exploitation est déjà lancé. Son fonctionnement est similaire à celui du CRTM.

Si le TPM permet de s'assurer que l'ordinateur est bien dans l'état voulu à la fin de son démarrage, cela n'empêche malheureusement pas des attaques de type « cold boot attack » [HSH+08] si la RAM a une rémanence trop grande, car alors il sera possible de récupérer les clefs présentes en RAM. La seule solution contre cela est d'avoir de la RAM très performante ayant le moins de rémanence possible. La protection des systèmes chiffrés contre des attaques mémoires est un sujet d'actualité [Dor11]. De plus, la clef racine est générée par le fabriquant du TPM et c'est lui qui l'y insère. Cela signifie qu'il dispose donc de la clef, ce qui est une limitation. Finalement, cette dernière étant stockée dans le TPM, il n'est donc pas inconcevable qu'il soit possible pour un attaquant d'y accéder même si cela demande des moyens colossaux.

Gestion des clefs

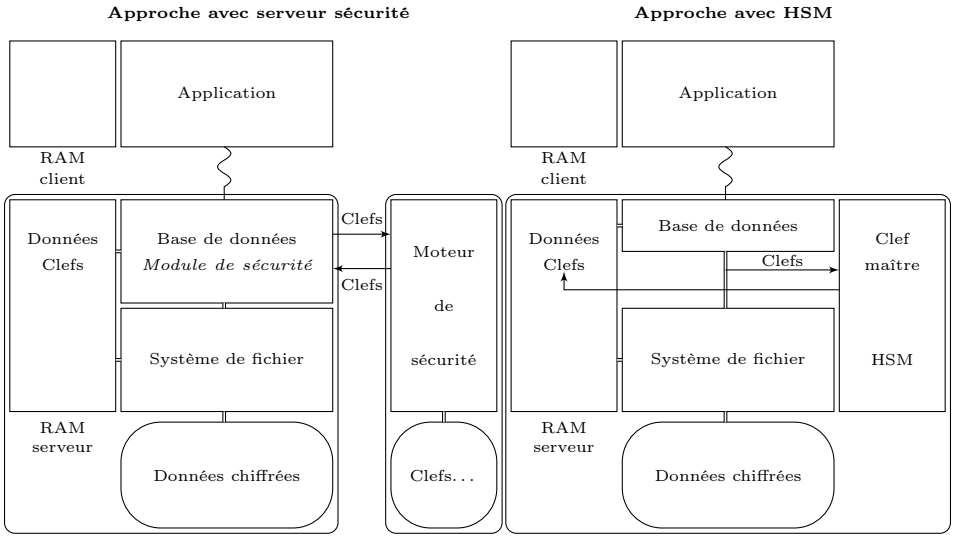
Le chiffrement repose sur des clefs qui permettent de chiffrer, déchiffrer, calculer des MAC... Connaître la clef (privée dans le cas d'un chiffrement asymétrique) est donc le but ultime de tout attaquant. Tout système cryptographique est donc limité par la protection de ses clefs. Leur stockage et les restrictions permettant d'y accéder sont alors un problème fondamental, en particulier, ne doivent pouvoir y avoir accès que les personnes dûment autorisées.

Dans certains cas, la solution adéquate est l'utilisation de mots de passe par l'utilisateur. Mais cela n'est pas toujours suffisant. Par exemple, dans le cas d'un chiffrement au niveau de la base de données, il faut déchiffrer les données au niveau du serveur sans intervention humaine et il faut donc stocker les clefs quelque part dans un endroit accessible par le serveur. De même, si l'on chiffre les disques durs, mais on souhaite que les serveurs puissent démarrer sans intervention humaine, il faut aussi stocker la clef quelque part. Les deux cas sont similaires, donc nous nous intéresserons principalement au premier, légèrement plus complexe, et plus à propos.

Comme expliqué dans [BG10] et [ABC+10], la façon la simple de procéder est de stocker toutes les clefs dont nous aurons besoin dans un fichier, ou une base de données dédiée, et de chiffrer cela avec une clef maître, stockée aussi quelque part sur le serveur. Le problème de cette solution est que tout

administrateur du serveur, voire des bases de données, peut utiliser cette clef maître à sa guise et modifier, lire, supprimer les données sans aucune restriction, et sans laisser de traces. Il existe principalement deux façons de remédier à ce problème, détaillées à la figure 3.3, mais dans les deux cas, les clefs de chiffrement et des données en clair apparaîtront dans la RAM du serveur, ce qui laisse des possibilités aux attaquants, mais complique néanmoins grandement leur tâche.

FIGURE 3.3 – Gestion des clefs de chiffrement [BG10]



La première approche consiste à utiliser un serveur dédié à la sécurité, serveur physiquement distinct de celui contenant les bases de données. Ce type de serveur est généralement appelé serveur de sécurité. Ce serveur gère alors les utilisateurs, les rôles, les droits d'accès, les règles et clefs de chiffrement. Le serveur de base de données contient lui un module de sécurité qui authentifie les utilisateurs, chiffre, déchiffre, en faisant appel au serveur de sécurité. Ce modèle permet de bien séparer les rôles au sein de l'hébergeur, et en particulier de réduire les possibilités d'accès aux clefs. Cela renforce donc la sécurité du système.

L'autre approche utilise un HSM (« hardware security module »), un cryptoprocesseur comme le TPM, mais qui s'en différencie par sa position et par ses fonctionnalités. En effet, un HSM est généralement branché sur un port PCI, contrairement à un TPM souvent situé dans la carte mère. De même, leurs API diffèrent grandement, et en particulier un HSM implémente généralement une seule fonctionnalité. Ce module matériel, situé dans le serveur de base de données, va contenir la clef maître, et les clefs chiffrées

seront sur le serveur, comme dans la façon la plus simple de procéder. Ici, la base de données va faire déchiffrer les clefs dynamiquement par le HSM, qui les effacera de la mémoire dès leur utilisation terminée. Cette solution offre les mêmes limitations que le TPM. Les HSM sont assez répandus et de nombreux modules de cette sorte sont certifiés par le NIST[4].

3.2.2 Chiffrement des disques durs

Une fois la chaîne de démarrage sécurisée, comme nous l'avons vu précédemment, une méthode peut être de chiffrer les disques durs. Pour cela, il existe principalement deux solutions : chiffrer le disque en dessous du système de gestion de fichiers, ou utiliser un système de gestion de fichiers chiffré. Si ces deux solutions sont maintenant bien matures, la protection de l'intégrité des disques durs ne l'est pas, même s'il commence à y avoir des prémices d'applications gérant cela.

Full Disk Encryption

Chiffrer totalement un disque dur peut se faire soit en matériel soit en logiciel. Du point de vue matériel, cela peut se faire soit directement avec un disque dur contenant les modules de chiffrement, soit via des chipsets servant d'interface. Le principal avantage de cette façon de procéder est que le chiffrement est complètement transparent pour le système d'exploitation, et de très bonnes performances en découlent. Par contre, cela génère un coût d'achat ou de migration de matériel que le logiciel n'induit pas. De même, le chiffrement matériel n'a pas la flexibilité du chiffrement logiciel où l'on peut aisément ne chiffrer que les partitions voulues, et même redimensionner les partitions (chiffrées ou non) en cas de besoin.

Du point de vue logiciel, il existe de nombreuses solutions[5], disponibles sur plusieurs systèmes d'exploitation, et sous diverses licences. Les plus couramment utilisés sont `dm-crypt` sur Linux, TrueCrypt et PGPDisk sur Microsoft Windows et Apple MacOS. Les fonctionnalités varient d'un logiciel à l'autre, mais les plus fréquentes sont les suivantes. Certains logiciels offrent une connexion possible avec un TPM, comme par exemple PGPDisk, ce qui permet de lier la chaîne de démarrage avec le chiffrement du disque. Il existe aussi des gestionnaires de clefs, tel le module de `dm-crypt` LUKS (Linux Unified Key Setup), qui permettent l'utilisation de plusieurs clefs de chiffrement pour une unique partition. Cela permet de gérer plusieurs utilisateurs et de révoquer des clefs sans avoir à rechiffrer toute la partition avec une clef différente. Les clefs peuvent être données soit sous forme de mot de passe, soit prises dans un fichier, soit issues d'une carte à puce.

[4] http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm
[5] http://en.wikipedia.org/wiki/Comparison_of_disk_encryption_software

Initialement, certains logiciels chiffraient en utilisant un IV ne dépendant que de la position du bloc à chiffrer, et comme nous l'avons vu dans la section 1.1.3, cela pose problème. Par exemple, Cryptoloop a cette propriété, ce qui a mené à des attaques permettant de détecter la présence de fichiers tatoués (« watermarked ») sur des disques chiffrés [Saa04]. Mais des techniques ont été mises en place pour remédier à ces problèmes pour les successeurs de Cryptoloop, comme `dm-crypt` pour Linux qui utilise l'API cryptographique du noyau Linux. À défaut d'assurer des IV différents lors d'une mise à jour d'un secteur, la solution d'ESSIV permet d'utiliser des IV inconnus des attaquants : ils sont issus du numéro du secteur et d'un haché de la clef, comme nous l'avons vu à la section 1.1.3. Le support de ESSIV [Fru05] (« Encrypt Salt-Sector intialization Vector ») est apparu dans le noyau Linux 2.6.10. La version 2.6.20 a ensuite vu l'ajout de LRW, remplacé par XTS depuis la version 2.6.24.

Malgré ce large choix disponible, peu de logiciels sont entièrement satisfaisants, que le problème vienne de leur pauvre qualité de programmation, de mauvais paramètres par défaut [GN06]...

Chiffrement au niveau du système de fichiers

Il est aussi possible de chiffrer au niveau du système de fichiers. Cela donne une plus grande flexibilité, mais laisse plus de données et méta-données en clair. eCryptfs[6] pour Linux, FileVault pour Apple MacOS et Encrypting File System (EFS)[7] pour Microsoft Windows fonctionnent de cette façon. Tous ces logiciels sont placés au-dessus du système de fichiers normal (ext4, HFS Plus, NTFS...) et ne s'y substituent nullement. De cela découle l'impossibilité de chiffrer une partition de SWAP avec un tel logiciel. Il est important de noter qu'a priori de tels logiciels ne chiffrent pas les noms de fichiers, même lorsque les fichiers sont chiffrés. Il est par contre, tout à fait possible de combiner cette approche avec celle du Full Disk Encryption.

Le choix entre un chiffrement de type Full Disk Encryption et un chiffrement au niveau du système de fichiers se fait en fonction des besoins exprimés[8], et dans notre cas, la nécessité d'une confidentialité totale nous impose le choix du Full Disk Encryption.

Protection de l'intégrité

Nous venons de passer en revue les moyens existant pour chiffrer les disques durs. Si nous n'avons pas parlé de protection de l'intégrité, c'est que cela n'est actuellement pas proposé par les systèmes vus précédemment. Le problème

[6] http://ecryptfs.sourceforge.net/
[7] http://msdn.microsoft.com/en-us/library/aa364223%28VS.85%29.aspx
[8] http://ecryptfs.sourceforge.net/ecryptfs-faq.html#compare

est en effet ardu : en plus de gérer les données chiffrées, il faut calculer les MACs et surtout leur allouer de la place et les stocker.

Dans le cas des Full Disk Encryption, le fonctionnement du chiffrement qui prend un bloc de données, le chiffre et le stocke sur le disque de façon transparente, ne permet pas de stocker un tag à côté du bloc chiffré. En effet, la transparence d'une telle méthode a pour implication l'impossibilité de décaler les blocs lors du stockage. De plus cela imposerait aux tags d'occuper un bloc entier de disque, soit généralement 512 octets, ce qui excède largement la taille d'un code d'authentification de message, entre 96 et 160 bits. Une solution consisterait à stocker les tags ensemble sur une partie du disque préallouée dans ce but ; mais cela est aussi impossible. Premièrement, le disque devrait faire de grands parcours lors de l'enregistrement d'un bloc ce qui induirait des performances catastrophiques et réduirait indéniablement la durée de vie du disque. Deuxièmement, les logiciels, comme par exemple `dm-crypt`, voient les blocs de façon anonyme, c'est-à-dire qu'ils prennent un bloc de clair et retournent un bloc de chiffré, sans avoir plus d'information sur ce bloc. En particulier, ils ne connaissent pas l'emplacement du bloc. S'ils calculaient le tag correspondant à un bloc i, ils ne pourraient donc pas le stocker comme étant le tag correspondant à ce bloc i, n'ayant pas accès à ce numéro i.

Pour avoir un contrôle de l'intégrité avec un chiffrement de type Full Disk Encryption, il y a principalement trois possibilités. La première est de vérifier globalement le disque en calculant un MAC du disque entier au démarrage et de le comparer à une valeur sur une clef USB, ou une carte à puce, mais cela ralentit trop le démarrage, ne donne aucune information pendant l'utilisation du disque, et manque grandement de précision. Ensuite, il ne serait pas inenvisageable de vérifier l'intégrité au niveau du système de fichiers. Certains systèmes, tels ZFS[9] ou Btrfs[10], calculent déjà des sommes de contrôle (« checksum ») basées, respectivement, sur SHA-2 et CRC-32C ; ces calculs de hachés se font incrémentalement sur l'arbre qui constitue le système de fichiers. Si cela n'est actuellement pas disponible, utiliser un code d'authentification de message à la place de tels hachés est envisageable et conviendrait à nos besoins. Finalement, un « Device Mapper », cadre générique qui gère les associations de blocs de disques, ayant pour but de veiller à la protection de l'intégrité des disques est en cours de développement[11]. L'utilisation de ce « Device Mapper », `dm-verity`, conjointement à celle de `dm-crypt` permettra à terme de bénéficier d'un contrôle d'intégrité et de confidentialité sur les disques durs.

Pour les chiffrements au niveau du système de fichiers, il faudrait soit y implémenter un système de vérification similaire à ceux de ZFS et Btrfs, soit

[9] http://hub.opensolaris.org/bin/view/Community+Group+zfs/WebHome

[10] https://btrfs.wiki.kernel.org/index.php/Main_Page

[11] http://git.chromium.org/gitweb/?p=chromiumos/third_party/kernel.git;a=blob;f=Documentation/device-mapper/dm-verity.txt

y introduire des chiffrements authentifiés, tels ceux que nous avons vus au chapitre 1.

3.2.3 Chiffrement des bases de données

Nous allons plus particulièrement nous intéresser aux bases de données relationnelles, car c'est ce type de base qui a le plus de chance de convenir pour stocker des données de santé, mais aussi car ce sont les bases de données les plus répandues dans la pratique, ce qui renforce la probabilité que ce type de base de données soit utilisé pour stocker des données nous intéressant. Il en existe un nombre assez considérable[12], mais nous nous concentrerons sur un panel des exemples les plus connus et utilisés : Oracle Database, Microsoft SQL Server, Microsoft Office Access, PostgreSQL, MySQL, SQLite. Cette liste permet aussi de mélanger des bases avec des licences différentes, ainsi que des utilisations qui varient grandement. Par exemple, les trois premières sont propriétaires, alors que les autres sont open-source ; SQLite est une base de données embarquée très petite, Microsoft Office Access n'a pas réellement vocation à être une base de données à part entière, mais d'offrir une intégration agréable pour ses utilisateurs dans Microsoft Office, alors que les quatre autres (Oracle Database, Microsoft SQL Server, PostgreSQL, MySQL) sont plus complètes.

Oracle Database. Oracle Database fournit de nombreux outils de sécurité[13], en particulier concernant les contrôles d'accès, mais aussi le chiffrement. C'est évidemment ce dernier point qui nous intéresse particulièrement. La première fonctionnalité de chiffrement offerte est de pouvoir effectuer des sauvegardes de l'ensemble de la base de données en la chiffrant sur le serveur et en l'envoyant ensuite dans les « nuages »[14].

Si cela est intéressant, c'est assez loin des besoins que nous avons, mais Oracle Database propose aussi du chiffrement dans l'utilisation de la base de donnée elle-même[15,16]. Pour se faire, il faut, lors du démarrage de la base de données, que son administrateur rentre le mot de passe maître. On voit donc que l'on a affaire à un chiffrement au niveau de la base de données. Deux choix sont alors proposés : soit chiffrer au niveau des tables, soit au niveau des colonnes. Dans le premier cas, cela n'implique pas de perte de fonctionnalité car la table est chiffrée avec ses index, et, une fois déchiffrée, il est possible d'y effectuer des recherches utilisant ces index ; mais il faut déchiffrer l'intégralité

[12] http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems

[13] http://www.oracle.com/us/products/database/security/index.html

[14] http://www.oracle.com/us/products/database/secure-backup-066578.html

[15] http://www.oracle.com/us/products/database/options/advanced-security/index.html

[16] http://www.oracle.com/us/solutions/sap/wp-ora4sap-tde11g-303816.pdf

de la table à chaque requête la concernant. Dans le second cas, Oracle donne encore deux choix. Le premier consiste à chiffrer chaque élément de la colonne sans utiliser d'IV. De cela découle un chiffrement déterministe de la colonne, qui, logiquement n'autorise alors que des index permettant une recherche par égalité. La deuxième option est d'utiliser un IV, mais alors il n'est plus possible de créer d'index sur la table, ce qui induit un temps de recherche grandement augmenté.

Ces outils peuvent être intégrés dans des solutions d'authentification telles PKI, Kerberos, RADIUS, HSM[17]. Oracle montre aussi son intérêt pour la cryptographie par son intégration rapide des instructions Intel pour l'AES[18].

Microsoft SQL Server. Microsoft SQL Server propose aussi de nombreuses solutions de sécurité pour protéger ses bases de données[19], parmi lesquelles on trouve des solutions de chiffrement[20]. Celles-ci diffèrent légèrement de celles d'Oracle Database. En effet, si le chiffrement par table est toujours possible et effectué au niveau de la base de données, avec une clef connue de l'administrateur, et si cela est toujours interfaçable avec un HSM, il n'est pas possible ici de chiffrer une colonne donnée. Par contre, Microsoft SQL Server offre la possibilité de signer des bouts de codes stockés et exécutés sur la base de données.

Microsoft Office Access. Microsoft Office Access propose moins d'options de sécurité[21], et en particulier le chiffrement se réduit ici au chiffrement global de la base de données uniquement possible en utilisant un mot de passe.

PostgreSQL. PostgreSQL offre aussi des options de sécurité, et en particulier de chiffrement[22]. Principalement, les options proposées sont les suivantes :

- chiffrement de la partition contenant la base de données, ce qui revient à du Full Disk Encryption,
- chiffrement de certains champs spécifiques via un module dédié[23], la partie la plus intéressante étant certainement l'utilisation de PGP qui

[17] http://www.oracle.com/us/corporate/press/018383
[18] http://www.oracle.com/us/corporate/press/173758
[19] http://www.microsoft.com/sqlserver/en/us/solutions-technologies/mission-critical-operations/security-and-compliance.aspx
[20] http://www.microsoft.com/sqlserver/2008/en/us/wp-sql-2008-security.aspx
[21] http://office.microsoft.com/en-us/access-help/introduction-to-access-2010-security-HA010341741.aspx
[22] http://www.postgresql.org/docs/9.1/static/encryption-options.html
[23] http://www.postgresql.org/docs/9.1/static/pgcrypto.html

propose des fonctions plus sûres que celle du module pgcrypto ; ici les possibilités de chiffrer avec ou sans IV, comme dans le cas d'Oracle Database, sont présentes,

- chiffrement côté client, si l'administrateur n'est pas de confiance, ce qui est transparent pour la base de données.

MySQL. MySQL est très pauvre en options de sécurité, et encore plus au niveau des options de chiffrement[24]. Il est uniquement possible de chiffrer des colonnes, sans IV, avec un mot de passe stocké ici directement dans le code SQL, et non au niveau du serveur.

SQLite. SQLite dispose lui de différentes extensions comprenant des outils de chiffrement, en particulier « The SQLite Encryption Extension (SEE) »[25] et « The SQLite Compressed and Encrypted Read-Only Database (CEROD) Extension »[26]. Si cette dernière se contente de compresser et chiffrer la base pour l'exporter sur support du type CDROM, la première vise à chiffrer les fichiers contenant la base lors de son utilisation. Cela est donc encore un chiffrement au niveau de la base de données sans possibilité de régler finement la granularité du chiffrement.

Notre but n'est pas de comparer ces différents moteurs de bases de données, mais de voir les meilleurs outils de chiffrement existant dans le monde des bases de données actuellement disponibles. Dans le cas où le chiffrement a lieu au niveau des bases de données, les outils proposés sont les suivants, la clef étant dans le meilleur des cas chiffrée à l'aide d'un mot de passe connu de l'administrateur de la base :

- chiffrement de l'ensemble des fichiers de la base de données,
- chiffrement d'une table et de ses méta-données associées (index...), son déchiffrement complet étant nécessaire pour son utilisation,
- chiffrement d'une colonne sans utilisation IV, avec possibilité de créer un index pour effectuer des recherches par égalité,
- chiffrement d'une colonne avec utilisation d'un IV, sans possibilité d'y associer un index, cela rendant nécessaire le déchiffrement complet de la colonne pour y effectuer une recherche.

Il est aussi évidemment possible de chiffrer les données au niveau applicatif, mais dans ce cas, la base de données gère la chaîne de caractères chiffrée comme une chaîne de caractères quelconque. De plus, il n'existe aucune solution pour protéger l'intégrité de ces bases de données.

[24] http://dev.mysql.com/doc/refman/5.6/en/encryption-functions.html
[25] http://www.hwaci.com/sw/sqlite/see.html
[26] http://www.hwaci.com/sw/sqlite/cerod.html

3.3 Confidentialité et intégrité des bases de données

Après avoir déterminé nos besoins en terme de sécurité et étudié les solutions techniques qui sont mises en place dans les principaux systèmes de gestion de base de données, nous allons définir dans cette section les desiderata en terme de chiffrement qui permettraient de définir une base de données dont la confidentialité et l'intégrité serait idéalement protégées.

3.3.1 Exigences

Nous passons en revue dans un premier temps les nécessités pour le chiffrement. A priori, nous n'imposons pas l'utilisation d'un unique chiffrement. Pour les chiffrements assurant la confidentialité, nous noterons E^b le chiffrement chiffrant les données dans la table et E^i le chiffrement dédié à l'index. De même, les algorithmes de chiffrements utilisés pour assurer l'intégrité des données seront notés A^b et A^i, respectivement pour celles des données dans les tables et celles dans les index. Nous utilisons A car nous utiliserons généralement des codes d'authentification de message (MAC), mais a priori cela est laissé libre. De plus, si nous avons besoin d'autres algorithmes de chiffrement, nous indiquerons le moment venu les notations utilisées, mais la convention E pour la confidentialité et A pour l'intégrité sera respectée.

Chiffrement des données des tables

La première contrainte qui, bien qu'évidente, doit être précisée, est que le chiffré d'un champ doit avoir une taille aussi proche que possible de celle du champ clair. Idéalement, le chiffrement utilisé ici doit, en plus d'être robuste, ne pas révéler les égalités entre les champs égaux. Si les mises à jour sont permises, il est aussi nécessaire que lorsqu'un champ est mis à jour, la connaissance des deux versions du champ n'apporte pas d'information. Idéalement, de cela découle la nécessité d'utiliser un IV par n-uplet pour le chiffrement, et que cet IV ne soit pas constant au cours du temps.

Les besoins pour E^b sont qu'il soit donc un chiffrement non déterministe dont les IV sont pertinemment choisis pour que pendant la durée de vie de la base de données, jamais deux fois le même IV soit utilisé avec la même clef, ou bien que E^b soit un chiffrement adaptable.

Chiffrement de l'index

Si l'on fait un index, les données qu'il contient doivent bien entendu être chiffrées, sinon il contiendrait la liste des valeurs en clair prises par le champ sur lequel est créé l'index. La recherche par intervalle serait une fonctionnalité intéressante si elle peut être obtenue sans nuire à la sûreté du chiffrement E^i.

Il y a principalement deux façons de construire un index dont les éléments sont chiffrés :

- construire l'index sur les données chiffrées avec E^i,
- construire l'index sur les données en clair et chiffrer ses éléments avec E^i a posteriori.

Construire l'index sur les données chiffrées. Dans ce cas, le chiffrement utilisé E^i doit être déterministe si l'on veut au moins être capable de faire des recherches par égalité. On remarquera qu'il découle de cela que E^b et E^i ne pourront être identiques, au moins pour leur gestion de l'IV, le premier étant déterministe tandis que le second ne l'est pas ; ce point étant relativement peu respecté dans l'état de l'art actuel du chiffrement des bases de données comme nous le verrons au chapitre 4. Nous appellerons ce type d'index un *index sécurisé*.

Construire l'index sur le clair, puis le chiffrer. Cette façon de procéder impose un chiffrement au niveau de la base de données, celle-ci ayant de nombreuses opérations de chiffrement et de déchiffrement à effectuer pour mettre à jour l'index. Par contre, rien n'impose a priori à E^i d'être déterministe. Nous appellerons ce type d'index un *index chiffré*.

Garantir l'intégrité

Finalement, il faut aussi garantir l'intégrité des tables et des index, à moindre coût. La solution la plus naïve, qui a quand même l'intérêt d'assurer une bonne sécurité, est de calculer un MAC par champ, ligne, colonne ou table dans la table et de le stocker dans une table spéciale. Ce MAC serait mis à jour à chaque modification du secteur concerné et vérifié à chaque accès. De même pour l'index, un MAC peut être généré par élément, branche si l'index est un arbre, paquet si c'est une table de hachage, voire globalement. Les mises à jour et les vérifications associées sont analogues à celles pour la table.

Spécificités liées au stockage

Certaines spécificités liées au stockage peuvent avoir des effets sur les chiffrements utilisés. En particulier, si le stockage est séquentiel, cela facilite grandement les choses. En effet, dans ce cas, on n'écrit sur le disque dur les mises à jour que périodiquement par séquences ; il est donc aisé de connaître le numéro du dernier enregistrement. Dans le cas d'un chiffrement au niveau de la base de données, ce numéro peut servir à constituer un IV évoluant au fil du temps et retrouvable aisément.

Dans ce cas, si la table est stockée par ligne et si l'on dispose d'index sur lesquels les recherches sont autorisées, le chiffrement peut être effectué pour tout le n-uplet constituant la ligne d'un seul coup, ce qui, en cas d'utilisation d'un chiffrement par bloc, économisera du temps. De même, si la table est stockée par colonne, on peut chiffrer la colonne d'un seul coup.

3.3.2 Accélération des requêtes

Chiffrer de façon sûre une base de données n'est pas suffisant, il faut encore pouvoir l'interroger de façon efficace. Il faudrait donc qu'il soit toujours possible de construire un index, et si possible de maintenir les recherches par intervalle, celles par égalité étant le strict minimum.

Chapitre 4

Chiffrement des bases de données

Dans ce chapitre, nous allons aborder à proprement parler le chiffrement des bases de données. Dans un premier temps, nous allons voir comment chiffrer de façon sûre une table d'une base de données, chose souvent laissée de côté dans la littérature. Ensuite, nous étudierons les solutions pour accélérer les requêtes, en nous basant sur l'état de l'art dans le domaine de chiffrement de base de données, sujet plus actif dans la communauté de base de données que dans celle de la cryptographie. La première étape est d'expliquer comment chiffrer correctement un index construit sur la table en clair. Ensuite, il est intéressant de se pencher sur les chiffrements à fonctionnalités particulières proposés principalement dans la communauté de bases de données. Finalement, nous verrons comment accélérer les requêtes sans pour autant utiliser un chiffrement de ce type.

4.1 Chiffrement de la table

Dans cette section, nous étudierons comment chiffrer la table. Pour l'instant, nous ne considérons pas encore les contraintes vis-à-vis des requêtes qui devront pouvoir être exécutées de façon efficace. Cela sera l'objet de la section suivante. Ici, nous nous concentrerons donc sur la sécurité et les performances du chiffrement de la table, en utilisant principalement les outils présentés au chapitre 1. Il est important de noter que dans cette section les chiffrements proposés peuvent être mis en place, soit au niveau de la base de données, soit au niveau applicatif, c'est-à-dire les deux derniers niveaux détaillés à la section 3.1.

4.1.1 Comment chiffrer la table ?

Avant de chiffrer une table d'une base de donnée, la première question est de savoir à quelle granularité le chiffrement doit être effectué. Autrement dit, il faut choisir quel est le plus petit élément chiffré indépendamment. Les principales possibilités sont les suivantes :

- la table entière,
- une ligne,
- une colonne,
- une cellule.

Généralement, pour accéder à, modifier ou supprimer une partie du plus petit élément chiffré indépendamment, il faut le déchiffrer et/ou le chiffrer en entier. Néanmoins certaines opérations peuvent être plus efficaces. Par exemple, si l'on chiffre une table en utilisant le mode opératoire CBC, et que l'on connaît la disposition des lignes et colonnes dans la table, et donc quels blocs de la table nous souhaitons déchiffrer, il est alors possible de ne déchiffrer que les blocs désirés sans avoir à déchiffrer intégralement la table. De façon analogue, chiffrer une colonne avec le mode CTR dans le cas où tous les champs ont la même taille et que l'on cherche à accéder au i^{e} élément, offre la même possibilité. Par contre, dans le premier cas la modification d'un élément nécessitera un rechiffrement de toutes les valeurs suivantes. Ce n'est pas le cas dans le second, mais nous avons vu à la section 1.1.3 qu'il n'est pas souhaitable d'utiliser deux fois la même suite chiffrante pour chiffrer deux valeurs distinctes.

Ces différentes granularités ont toutes leurs avantages et inconvénients. Par exemple, si l'on chiffre la table comme un unique élément, toute requête nécessitera un déchiffrement total de la table. Cela peut être anodin dans le cas d'une recherche sur tous les éléments de cette table, mais implique un fort surcoût si l'on cherche à lire une ligne, voire une cellule unique. Cette solution a par contre des avantages : si l'on utilise un chiffrement nécessitant une initialisation, par exemple le pré-chiffrement d'un IV, celle-ci a besoin d'être effectuée une unique fois pour toute la table ; et, en général, peu voire pas de place sera perdue à cause du remplissage.

À l'autre extrémité, un chiffrement au niveau des cellules permet de ne déchiffrer que la cellule désirée. Il permet aussi de ne pas chiffrer toutes les cellules, mais seulement celles le nécessitant. Dans les bases de données, il est fréquent d'avoir des cellules de petite taille, voire même réduites à un unique bit si elle représente un choix de type « Oui/Non ». Si l'on utilise du remplissage pour une telle case, cela va entraîner une grande perte de place. Si l'on utilise un chiffrement à flot, l'initialiser pour chaque cellule différente prendra du temps.

On voit que le choix de la granularité a un impact sérieux sur les performances. Il convient donc de définir quelles seront les contraintes qui seront imposées à la base de donnée pour pouvoir choisir la granularité au mieux. Une configuration classique est celle où la table est chiffrée, et où il n'existe pas d'index permettant d'accélérer les recherches. Dans ce cas, il y a principalement deux possibilités. La première consiste à avoir une table composée uniquement d'une colonne avec des identifiants non chiffrés et d'une colonne contenant des données à chiffrer. Si, en plus de cela, les modifications et les ajouts dans la table sont rares, il peut être intéressant de chiffrer la colonne en une seule fois, car toute recherche nécessitera son déchiffrement total et qu'en cas de recherche fructueuse, l'identifiant associé est aisément récupérable puisqu'il est en clair sur la même ligne de la colonne composée des identifiants. Si la table contient au contraire plusieurs colonnes dont on veut chiffrer le contenu, et que l'on souhaite récupérer la ligne entière et non uniquement l'identifiant, il devient alors intéressant de chiffrer au niveau des cellules. Une deuxième configuration classique est celle où l'on dispose d'un index pour accélérer les requêtes. Dans ce cas, l'index nous donnera directement la ligne désirée, et il peut être adéquat de chiffrer alors par ligne.

En dehors des problèmes liés au remplissage, chiffrer avec une granularité très fine peut parfois s'avérer dangereux. En effet, un attaquant peut alors retirer de l'information d'une table chiffrée. Plus précisément, si l'attaquant a accès à la table chiffrée à deux instants t et t', il est capable de voir si un élément chiffré a changé entre temps. Cela peut être grandement dommageable dans certains cas. Par exemple, dans un cas similaire à celui du DMP, considérons une table stockant si les patients ont le VIH, et supposons que cette table contient deux colonnes : « identifiant » et « a le VIH », la première colonne étant en clair et la seconde chiffrée au niveau des cellules. Alors un changement de valeur dans la seconde colonne ne peut être fait que dans une seule direction ; cela signifie que le patient est maintenant atteint par le VIH. Dans ce contexte, il sera nécessaire de ne pas chiffrer au niveau des cellules, mais des colonnes. De façon similaire, si le tweak ou l'IV utilisé ne dépend que de l'élément chiffré (table, colonne, ligne ou élément), sans tenir compte des mises à jour via un compteur ou un timestamp, et que le chiffrement utilisé n'est pas de type « wide block tweakable encryption », comme vu à la section 1.1.3, il sera généralement possible de détecter quelle partie du chiffré a changé. Par exemple, dans le cas précédent, même avec un chiffrement au niveau de la colonne, si le mode opératoire choisi est le mode CBC, un attaquant pourra voir quel est le premier bloc à avoir changé, et ainsi obtenir une information assez précise.

Une fois la granularité choisie, il reste à déterminer deux choses : l'algorithme de chiffrement, et le mode à utiliser. Le choix de l'algorithme ne porte pas réellement à discussion. Vu qu'il est nécessaire d'avoir quelque chose de robuste, il faut soit prendre AES (décrit à la page 19), voire un des autres

finalistes du concours AES (MARS, RC6, Serpent, Twofish[1]), si l'on veut un chiffrement par bloc, soit, si l'on désire un chiffrement à flot, un chiffrement issu du portfolio eSTREAM [ECR05]. Le critère important pour ce choix est d'utiliser un chiffrement sûr et déjà grandement éprouvé en pratique, potentiellement dans d'autres contextes.

Le choix du mode est plus complexe et dépend du but poursuivi :

- chiffrer les données,
- chiffrer et assurer l'intégrité des données.

Nous allons étudier ces deux cas dans les deux sections suivantes.

4.1.2 Chiffrement simple de la table

Pour chiffrer les données de la table, sans nécessairement en assurer l'intégrité, deux possibilités s'offrent à nous :

- utiliser un mode de chiffrement standard,
- utiliser un mode de chiffrement adaptable.

Chiffrement de la table avec un mode de chiffrement standard

Cela peut être obtenu soit avec un chiffrement à flot, soit avec un chiffrement par blocs, comme vu à la section 1.1.2. Le contexte est ici fortement similaire à celui de la section 3.2.2. En particulier, il faut aussi faire attention au choix de l'IV, comme expliqué à la section 1.1.3, car les attaques, comme celle décrite dans [Saa04] s'appliquent aussi ici. Il faut donc aussi utiliser une technique de renforcement de l'IV, telle ESSIV introduite par [Fru05] et décrite à la section 1.1.3. Pour rappel, l'IV se déduit alors d'une valeur publique S, de la façon suivante : $IV = E_{h(K)}(S)$, où h est une fonction de hachage cryptographique, E l'algorithme de chiffrement utilisé et K la clef. Pour que l'IV soit unique pour chaque élément utilisé, la valeur publique S doit vérifier cette propriété. Voici donc une façon de la calculer en fonction de la granularité :

- chiffrement au niveau de la table : $S = n_{\text{base}} \| n_{\text{table}}$,
- chiffrement au niveau des colonnes : $S = n_{\text{base}} \| n_{\text{table}} \| n_{\text{colonne}}$,
- chiffrement au niveau des lignes : $S = n_{\text{base}} \| n_{\text{table}} \| n_{\text{ligne}}$,
- chiffrement au niveau des cellules : $S = n_{\text{base}} \| n_{\text{table}} \| n_{\text{ligne}} \| n_{\text{cellule}}$,

[1] http://csrc.nist.gov/archive/aes/round2/r2report.pdf

où chaque numéro est écrit sur un nombre de bits constant pour prévenir toute ambiguïté.

Malheureusement, la remarque finale de la section 1.1.3 selon laquelle après la mise à jour d'un élément, un attaquant disposerait du XOR des valeurs claires anciennes et actuelles est pertinente ici si le mode utilisé est le mode CTR ou si on utilise un chiffrement à flot. Cela est aussi vrai pour le premier bloc en cas d'utilisation du mode CBC. Il convient donc d'éviter cette solution dans le cas où un IV va être utilisé plusieurs fois.

Chiffrement de la table avec un mode de chiffrement adaptable

Les chiffrements adaptables et leurs modes de chiffrement associés, vus aux sections 1.1.2 et 1.1.3, permettent, comme dans le cas des chiffrements de disques durs, de pallier ce problème. La version actuellement considérée comme la plus sûre est AES-XTS normalisée par l'IEEE et le NIST [IEE08, NIS10]. Nous l'avons déjà décrite précédemment à la section 1.1.3.

De même, il peut aussi être intéressant d'utiliser dans ce cas des solutions plus coûteuses, comme EME, vu aussi à la section 1.1.3, pour lesquelles la modification d'un bit de clair entraînerait une modification du chiffré de façon indépendante de la position du bit modifié. Cela correspond à la notion de « Wide block tweakable encryption » définie à la section D.3 de [IEE08], qui apporte plus de sécurité, mais en contrepartie, détériore les performances.

Quelle solution privilégier ?

Pour choisir entre ces deux possibilités, nous pouvons distinguer deux principales façons de gérer une base de données. La première, et la plus classique, est d'autoriser la réécriture d'un même champ plusieurs fois, c'est-à-dire qu'un champ situé dans une cellule donnée pourra voir sa valeur évoluer au fil du temps. L'autre solution, décrite par exemple dans [YPM09] et [Guo11], est d'avoir une base de donnée séquentielle où les réécritures sont interdites et où les mises à jour se font par création d'une nouvelle table avec un nouveau numéro de version lorsqu'un nombre critique de mises à jour a été atteint. Nous distinguerons dorénavant ces deux possibilités sous les noms de cas « classique » et « séquentiel ».

Pour le cas classique, le chiffrement avec un mode opératoire standard n'est envisageable que si l'IV dépend aussi d'un numéro de version ou d'un timestamp, ce qui nécessite dans tous les cas une expansion de la taille du chiffré, car cet IV, ou du moins ce numéro de version ou ce timestamp doit être stocké avec le chiffré, et ceci n'est donc souvent pas acceptable. Par contre, la solution avec un chiffrement adaptable présente moins d'inconvénients, même si le même tweak est utilisé plusieurs fois ; la seule information pouvant alors être déduite est le fait qu'une valeur chiffrée ait été mise à jour.

Dans le cas séquentiel, le chiffrement avec un mode opératoire standard ne pose plus de problème, l'IV dépendant alors naturellement du numéro de la version. Un chiffrement avec un mode adaptable peut aussi être utilisé dans ce cas, mais ne sera pas privilégié, étant généralement plus coûteux.

4.1.3 Chiffrement et protection de l'intégrité de la table

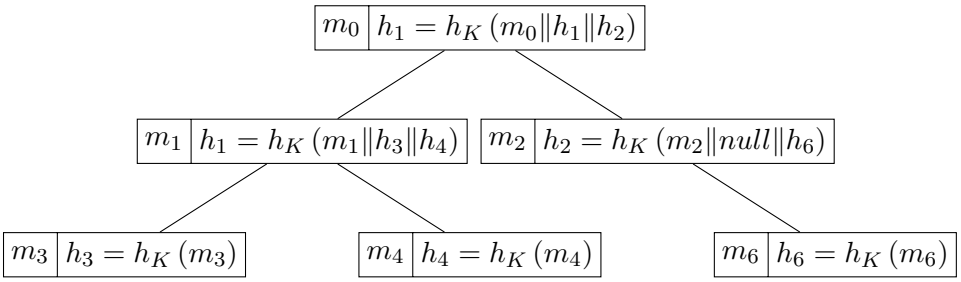
Comme nous l'avons évoqué précédemment, chiffrer une base de donnée n'est pas suffisant pour des données sensibles. Il est aussi nécessaire d'assurer son intégrité. Nous allons voir dans un premier temps les solutions les plus classiques pour assurer l'intégrité, essentiellement les modes de chiffrement authentifiés vus à la section 1.1.5, et en particulier le mode CCM [NIS04], qui est le principal mode normalisé par le NIST, et donc l'un des plus étudiés. Cette solution est celle préconisée dans [Küh06]. Puis nous détaillerons une méthode plus récente appelée AREA [ETS+06, ECL+07, Fru05] (« Added Redundancy Explicit Authentication »), qui se veut moins coûteuse qu'un véritable mode de chiffrement authentifié.

Protection classique de l'intégrité des bases de données

La façon classique de protéger l'intégrité de données que l'on souhaite chiffrer, et a fortiori d'une base de donnée, est l'utilisation de signatures ou de MACs, comme vu à la section 1.1.4, ou celle d'un chiffrement authentifié, comme présenté à la section 1.1.5.

H. Hacigümüs, B. R. Iyer et S. Mehrotra [HIM03] proposent deux moyens simples d'assurer l'intégrité des bases de données. Tout d'abord, il est possible d'assurer l'intégrité des données au niveau des cellules. La bonne façon de procéder ici est d'utiliser des MACs et non le chiffré de la valeur concaténée à son haché comme proposé dans cet article. La seconde solution consiste à effectuer des signatures incrémentales au niveau de la table. Cela est particulièrement adapté aux index en arbre où la structure même de l'index permet l'utilisation de MACs de cette façon. Par exemple, le MAC calculé sur une feuille serait celui naturellement calculé sur une cellule d'un index. Ensuite, le MAC calculé pour un nœud interne d'un index prendrait en compte, non seulement les valeurs de la clef de recherche et des pointeurs partant de ce nœud, mais aussi la valeur des MACs calculés dans ses fils. Cela permet, en plus de protéger l'intégrité des données dans l'index, d'assurer la non permutation de valeurs dans l'index. Par contre, lors d'une mise à jour de l'index, il faut mettre à jour toute la branche jusqu'à la racine, ce qui peut ne pas être trop coûteux dans le cas des arbres B et B+ vus à la section 1.2.3 qui ont une petite hauteur. La figure 4.1 présente un exemple de cette protection de l'intégrité dans un arbre binaire, où la fonction calculant le MAC est notée h_K. Dans cette figure, on voit en particulier qu'une modification de m_3 implique de recalculer les MACs h_3, h_1 et h_0, mais pas les autres.

FIGURE 4.1 – Protection incrémentale de l'intégrité dans un index en arbre



De façon antérieure à [Küh06], [McG05] propose d'utiliser un chiffrement authentifié, GCM vu à la section 1.1.5, pour chiffrer des ensembles de données structurés. Cet article vise un chiffrement au niveau des disques ou des fichiers et s'adapte donc à notre étude si l'on souhaite chiffrer les fichiers de la base de données.

CCM (Counter with CBC-MAC)

CCM est un mode de chiffrement authentifié à deux passes proposé par D. Whiting, R. Housley et N. Ferguson. Ce mode est construit sur le modèle authentifier puis chiffrer (AtE), vu à la section 1.1.5. Il s'appuie sur un chiffrement par bloc, par exemple AES, dont la taille de bloc est $n = 128$. CCM calcule d'abord un CBC-MAC, puis chiffre ce MAC et les données en clair avec le mode compteur (CTR). Il permet l'utilisation de données associées, c'est-à-dire de données dont il assure l'intégrité, mais qu'il ne chiffre pas. Sa caractéristique principale est que la même clef est utilisée pour le MAC et le chiffrement. La raison pour laquelle nous étudions en détail CCM est que c'est un mode utilisé dans de nombreux standards : IEEE 802.11 wireless standard, NIST Special Publication 800 - 38 C [NIS04]. Sa sécurité a été grandement étudiée. Dans [Jon02], J. Jonsson a prouvé que CCM est sûr jusqu'à $2^{\frac{n}{2}}$ requêtes de chiffrement pour le chiffrement et la protection de l'intégrité, où n est la taille du bloc, soit donc $n = 128$ lorsqu'il est utilisé avec AES. Malgré cela, la meilleure attaque contre l'intégrité assurée par CCM est en 2^n requêtes, et J. Jonsson conjecture que c'est sa sécurité réelle. Dans [FMVZ08], P.-A. Fouque, G. Martinet, F. Valette et S. Zimmer prouvent cela pour une version modifiée de CCM utilisant une fonction pseudo-aléatoire au lieu d'un chiffrement par bloc, et en utilisant deux clefs distinctes pour le chiffrement et l'authentification.

Description de CCM. Nous décrivons CCM dans l'algorithme 4.1 de façon plus détaillée qu'à la figure 1.23. Nous laissons de côté les données associées qui ne nous intéressent pas ici.

Nous notons T le tag généré par le MAC, dont la taille est t, paramètre pouvant être choisi par l'utilisateur, mais qui influe directement sur la sécurité. En effet, un tag généré aléatoirement sera valide avec une probabilité 2^{-t}. Généralement, t est un multiple de 16 bits, compris entre 32 et 128. Le standard IEEE 802.11i recommande $t = 64$ bits.

Algorithme 4.1 CCM (avec un chiffrement par bloc E de taille de blocs n)

Entrées : Une clef secrète K.
Entrées : m données $d_1, \ldots, d_m$, de tailles en octet respectives $\ell_1, \ldots, \ell_m$.
Sorties : Les m données chiffrées $(C_1, \ldots, C_{k_1}), \ldots, (C_1, \ldots, C_{k_m})$ et leurs tags correspondant $T_1, \ldots, T_m$.

pour i de 1 à m **faire**
  // **Calcul du MAC**
  // Remplir d_i avec des zéros pour obtenir un multiple de n bits
  $\mathrm{Pad}(d_i) = (M_1, \ldots, M_{k_i})$
  // Initialisation du CBC-MAC
  $B_0 = \beta(IV_i, \ell_i)$
  $X_0 \leftarrow E_K(B_0)$
  // Calcul du CBC-MAC
  pour j de 1 à k_i **faire**
    $X_j \leftarrow E_K(X_{j-1} \oplus M_j)$
  fin pour

  // **Chiffrement avec le mode CTR**
  $A_0 = \alpha(IV_i, \ell_i)$
  // Chiffrement du tag
  $T_i \leftarrow \mathrm{lsb}_t(X_{k_i} \oplus E_K(A_0))$
  pour j de 1 à k_i **faire**
    $A_j \leftarrow A_{j-1} + 1$
    $C_j \leftarrow E_K(A_j) \oplus M_j$
  fin pour
  retourner $(C_1, \ldots, C_{k_i}, T_i)$.
fin pour

Pour chiffrer chaque donnée, un IV public de v bits est utilisé. Comme cette valeur est nécessaire pour la vérification de l'intégrité et le déchiffrement, elle doit être soit stockée avec le chiffré, soit générée à partir d'autres données publiques. Ici encore la taille de cette valeur conditionne la sécurité du mode. Dans CCM, la condition $v \leq 104$ bits est néanmoins imposée car le premier bloc de 128 bits doit aussi inclure la taille de la donnée à chiffrer. On remarque que le dernier bloc de chiffré ne fait pas nécessairement 128 bits, cela venant de l'utilisation du mode CTR pour le chiffrement, mode qui

« transforme » un chiffrement par bloc en chiffrement à flot. Le message n'est réellement rempli que pour le CBC-MAC générant le tag T ; ce remplissage n'a donc pas pour effet d'augmenter la taille du chiffré.

Comment choisir les valeurs initiales ? Le principal problème dans le contexte du chiffrement de bases de données est maintenant le choix des valeurs A_0 et B_0 qui servent à initialiser les modes CBC et CTR. Ces valeurs initiales sont déterminées par le choix de l'IV et par les fonctions α et β. Dans le standard proposé par le NIST, elles sont définies comme suit :

$$\beta(IV, \ell) = 0_2 || \left(\frac{t}{16} - 1\right) || \left(\left\lceil \frac{\log_2 \ell}{8} \right\rceil - 1\right) ||IV||\ell,$$

$$\alpha(IV, \ell) = 0_5 || \left(\left\lceil \frac{\log_2 \ell}{8} \right\rceil - 1\right) ||IV||0 \ldots 0.$$

Ainsi, le premier octet de B_0 est constitué de deux zéros, suivis du nombre de mots de 16 bits formant le tag T, cela encodé sur 3 bits, ensuite suivi par la longueur en octets de ℓ, aussi encodé sur 3 bits étant donné que $\ell < 2^{64}$.

La sécurité de CCM repose alors sur l'impossibilité d'utiliser les mêmes valeurs initiales plus d'une unique fois. Plus précisément, la sécurité du mode de chiffrement authentifié est garantie si et seulement si toutes les valeurs initiales du mode CBC B_0 et tous les compteurs distincts A_j du mode CTR sont distincts pour toutes les données traitées. Il est important de remarquer que dans le standard du NIST, la forme particulière de A_0 et B_0 garantit que tous les A_j diffèrent de B_0 car tous les A_j commencent par cinq zéros, alors que B_0 contient au minimum un bit non nul parmi ses cinq premiers. De plus, ces choix de α et β assurent qu'un choix de valeurs distinctes pour les IV de chaque donnée traitée est (nécessaire et) suffisant pour assurer que toutes les valeurs A_j et B_0 de toutes les données seront bien elles aussi distinctes. Par ailleurs, nous voyons que cela implique aussi que les IV sont ainsi déterminés de façon unique pour un A_0 ou un B_0 donné.

Nous allons maintenant étudier la sécurité de la version originale de CCM, c'est-à-dire avec des valeurs initiales distinctes, ce qui correspond à ce que nous venons de voir. Nous allons aussi prendre en considération des variantes où cette condition n'est pas remplie. Dans la première variante, les IV sont choisis aléatoirement ; dans la seconde, les valeurs initiales du mode CTR sont uniques, mais ne prennent pas en compte l'IV.

Sécurité de CCM avec des IV distincts [Jon02]. Selon [Jon02], la sécurité de CCM est garantie quand les conditions suivantes sont remplies :

- toutes les valeurs prises par les compteurs A_j et les valeurs initiales du CBC-MAC B_0 sont distinctes pendant la durée de vie de la clef ;

- la valeur de l'IV est déterminée de façon unique par la connaissance de B_0 et de A_0 ;
- l'encodage précédant le chiffrement en mode CBC est « prefix-free », c'est-à-dire que le format de l'entrée du mode CBC doit garantir qu'aucun préfixe d'une entrée valide est aussi une entrée valide ; dit autrement, avec les notation de l'algorithme 4.1 si d_i et d_j sont deux données distinctes et $(B_0, M_1, \ldots, M_{k_i})$ et $(B'_0, M'_1, \ldots, M'_{k_j})$ leurs blocs formatés pour le CBC-MAC respectifs, alors soit $B_0 \neq B'_0$, soit M_ℓ et M'_ℓ sont distincts pour un ℓ inférieur à $\min\{k_i, k_j\}$.

Plus précisément, les aspects suivants sont garantis.

Le modèle de sécurité pour la confidentialité est ici le suivant. Il doit être impossible de dériver une quelconque information à partir des données chiffrées, sans avoir eu accès à la clef. Nous nous plaçons dans le cadre d'une attaque à clairs choisis, comme vu à la section 1.1.1. L'attaquant a donc accès à un oracle de chiffrement qui retourne $(C_1, \ldots, C_k, T)$ pour n'importe quelle donnée de son choix. Nous ajoutons la restriction que le même IV ne peut pas être utilisé plus d'une seule fois. L'attaquant a alors pour but de distinguer avec une probabilité significativement supérieure à $1/2$ cet oracle de chiffrement d'un oracle aléatoire qui retourne des chaînes aléatoires de même longueur. Le niveau de sécurité est défini comme le nombre de requêtes à l'oracle de chiffrement nécessaires pour construire un distingueur.

À partir de l'hypothèse que l'algorithme de chiffrement par bloc utilisé E est idéal, [Jon02] prouve que distinguer une donnée chiffrée avec CCM d'une suite aléatoire requiert plus de $2^{\frac{n}{2}}$ requêtes à l'oracle de chiffrement, où n est la taille de bloc du chiffrement employé.

Cette borne est en fait atteinte à cause de l'attaque suivante. Soit $\mathcal{M}$ un message formé de $2^{\frac{n}{2}}$ blocs identiques M :

$$\mathcal{M} = \underbrace{(M \ldots M)}_{2^{\frac{n}{2}}}.$$

Quand l'attaquant interroge l'oracle de chiffrement, il reçoit $\left(C_1, \ldots, C_{2^{\frac{n}{2}}}\right)$, où $C_i = M \oplus E_K(A_i)$. Tous les A_i sont distincts et donc tous les C_i le sont aussi. Or si l'attaquant reçoit $2^{\frac{n}{2}}$ blocs aléatoires, il y a une forte probabilité que deux d'entre eux soient égaux d'après le paradoxe des anniversaires. Ainsi, avec $2^{\frac{n}{2}}$ requêtes, l'attaquant peut distinguer avec une forte probabilité une donnée chiffrée d'une chaîne aléatoire de même longueur.

Concernant l'intégrité, le but est qu'il soit impossible de falsifier un chiffré valide sans accès à la clef. Ici l'attaquant a accès à un oracle de chiffrement

et aussi à un oracle qui vérifie si un chiffré est valide. Son but est alors de falsifier un chiffré valide.

[Jon02] prouve qu'un attaquant a nécessairement besoin d'au moins $2^{\frac{n}{2}}$ requêtes à l'oracle de chiffrement, ou de 2^t tentatives de falsification. Il en découle que la borne inférieure de sécurité pour l'intégrité dans CCM est le minimum de 2^t et $2^{\frac{n}{2}}$. Dans ce même article, l'auteur conjecture que la borne est en fait le minimum de 2^t et 2^n. [FMVZ08] montre qu'une version modifiée de CCM a bien cette dernière borne comme limite pour l'intégrité. Pour cela, CCM n'utilise plus un chiffrement par bloc mais une fonction pseudo-aléatoire. Plus précisément, soit E le chiffrement par blocs précédemment utilisé pour calculer le CBC-MAC et le chiffrement en mode compteur, n sa taille de bloc, et K l'unique clef utilisée. Soit alors (F_K) une famille de fonctions pseudo-aléatoires de l'ensemble $\{0,1\}^n$ vers lui-même, et K' une clef choisie indépendamment de K. La version modifiée de CCM utilisée dans [FMVZ08] consiste à remplacer, dans le chiffrement avec le mode CTR, E_K par $F_{K'}$, un élément de la famille F choisi à partir de la nouvelle clef K'.

Sécurité de CCM avec des IV aléatoires [FMVZ08]. Nous nous intéressons maintenant à la sécurité de CCM sous les conditions suivantes :

- la valeur de l'IV est déterminée de façon unique par B_0 et A_0,
- la valeur de l'IV est un mot de v bits aléatoire,
- l'encodage du mode CBC est « prefix-free ».

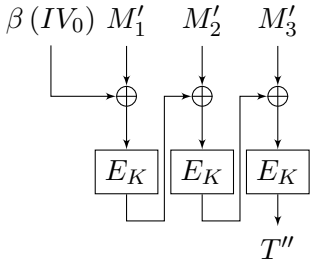
L'intérêt d'une telle variante de CCM dans le contexte du chiffrement de bases de données apparaît si l'on souhaite générer un nouvel IV pour chaque nouvelle donnée que l'on chiffre, que ce soit réellement une nouvelle donnée, ou une donnée mise à jour, sans pour autant devoir vérifier qu'il n'a jamais été utilisé. Cette dernière contrainte aurait pour effet de devoir parcourir tous les IV, et de devoir conserver les IV des données supprimées pour cette vérification.

Un exemple d'implémentation de cette variante consiste à utiliser les fonctions α et β comme définies dans le standard du NIST, à la différence suivante : si l'on chiffre plusieurs données $\{\mathcal{M}_i\}_{0\leq i\leq n-1}$, on choisit $IV_i = T_{i-1}$, c'est-à-dire que l'IV utilisé pour chiffrer une donnée est le tag généré par le CBC-MAC sur la donnée précédente. Comme la sortie du MAC est supposée indistinguable d'une valeur aléatoire, nous sommes bien dans le cas décrit ci-dessus. Dans le contexte de bases de données séquentielles, où les données ne peuvent pas être effacées, cela est particulièrement pertinent, car alors les IV n'ont pas besoin d'être stockés. Par contre, si les données pouvaient être modifiées, une mise à jour entraînerait nécessairement le rechiffrement de toutes les données suivantes.

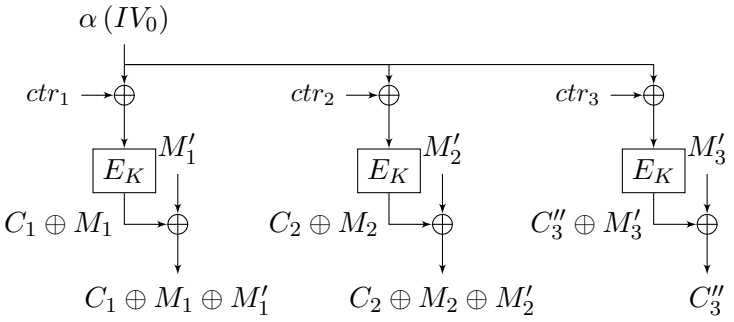
Malheureusement, cette variante est vulnérable à une attaque générique qui requiert $2^v + 2^{\frac{t+v}{2}}$ chiffrements, d'après [FMVZ08], où t est la taille du tag T et v celle de l'IV. Elle se déroule comme suit. Supposons que l'attaquant puisse obtenir les chiffrés correspondant à $2^{\frac{t+v}{2}}$ clairs de la forme (M_1, M_2, M_3) où M_1 et M_2 sont variables, M_3 est fixé, et l'IV est choisi aléatoirement. La réponse prend la forme $IV, (T, C_1, C_2, C_3)$. Alors, avec une probabilité significativement plus grande que $1/2$, l'ensemble des chiffrés contient deux éléments dont l'IV et le tag T sont identiques. Autrement dit, l'attaquant a trouvé deux couples $(IV, \mathcal{M})$ de la forme (IV_0, M_1, M_2, M_3) et (IV_0, M'_1, M'_2, M_3) qui ont le même tag T. Il en découle, étant donné que M_3 est fixé, que pour toute valeur possible de M'_3 les tags calculés pour les données (IV_0, M_1, M_2, M'_3) et (IV_0, M'_1, M'_2, M'_3) seront en fait égaux. L'attaquant demande alors à l'oracle de chiffrement les chiffrés correspondant à 2^v clairs différents (M_1, M_2, M'_3) où M_1 et M_2 sont les valeurs précédemment trouvées, où M'_3 varie tout en étant différent de M_3 et où l'IV est choisi aléatoirement. Alors, avec une forte probabilité, la valeur IV_0 sera utilisée comme IV et l'attaquant obtiendra le chiffré $(T'', C''_1, C''_2, C''_3)$ correspondant à (M_1, M_2, M'_3) commençant par IV_0. Il est clair que $C''_1 = C_1$ et $C''_2 = C_2$, où (C_1, C_2) est le chiffré de (M_1, M_2), étant donné que le même IV est utilisé pour chiffrer les deux messages. Il en découle qu'il est possible de falsifier un chiffré valide : $(T'', C_1 \oplus M_1 \oplus M'_1, C_2 \oplus M_2 \oplus M'_2, C''_3)$ est aussi un chiffré valide de (M'_1, M'_2, M'_3), comme l'illustre la figure 4.2.

FIGURE 4.2 – Attaque contre CCM avec des IV aléatoires

Calcul du tag :



Chiffrement :



Sécurité de CCM avec des IV aléatoires et une autre initialisation pour le mode compteur. Nous proposons un exemple simple de cette variante. Pour la i^{e} donnée, on choisit :

$$\begin{aligned} B_0 &= 1||T_{i-1} \\ A_0 &= 0||\textstyle\sum_{j=0}^{i-1} \ell_j, \end{aligned}$$

où ℓ_i est la taille en octet de la i^{e} donnée. Comme dans la version du NIST, le premier bit assure ici que B_0 diffère de tous les A_j. Ce choix est assez naturel dans le contexte du chiffrement de bases de données car alors le tag et le chiffré de la i^{e} donnée correspond à une valeur intermédiaire du calcul de CCM pour la donnée totale $(\mathcal{M}_1\| \ldots \|\mathcal{M}_m)$. Dans la preuve de sécurité de [Jon02], le fait que l'IV utilisé pour le CBC-MAC soit aussi utilisé pour le chiffrement en mode compteur (CTR) diminue la borne de sécurité. Il en découle qu'utiliser deux IV différents, tout en continuant de garantir que pour une clef donnée les B_0 et les A_i sont bien tous distincts deux à deux, n'invalide pas la preuve, et donc peut être envisagé. La sécurité attendue pour cette version est donc la même que la précédente.

Autres variantes avec IV aléatoires et versions modifiées de CCM. La première idée est de supprimer le premier chiffrement du CBC-MAC, c'est-à-dire imposer $X_0 = B_0$. Dans ce cas, lorsque l'on chiffre plusieurs données à la suite, l'IV du CBC-MAC devient alors réellement le tag chiffré de la donnée précédente, et cela sans le formatage ni le pré-chiffrement préconisé par le NIST.

Nous avons malheureusement trouvé une attaque assez simple contre cette version. Supposons que l'attaquant connaisse les chiffrés correspondant à deux données $\mathcal{M}_1$ et $\mathcal{M}_2$, chiffrées à partir d'un IV donné. Pour des raisons de simplicité, supposons que ces données ne font qu'un bloc et notons les donc respectivement M_1 et M_2. Soient donc (T_1, C_1), (T_2, C_2) les chiffrés correspondant, comme décrit à la figure 4.3. L'attaquant peut alors falsifier un chiffré partiellement valide pour le couple de données $(M_1, M_2 \oplus \Delta)$, chiffré à partir du même IV que précédemment, pour n'importe quel Δ. En fait, dans le résultat du chiffrement $((T_1 \oplus \Delta, C_1), (T_2, C_2 \oplus \Delta))$, le tag T_2 est un MAC valide pour la deuxième donnée, comme cela montré à la figure 4.4. Par contre le tag de la première donnée n'est pas valide, mais pour s'en rendre compte il faut alors vérifier l'intégrité de toutes les données à chaque fois, alors que l'intégrité devrait pouvoir être vérifiée individuellement pour chaque donnée.

FIGURE 4.3 – Variante de CCM sans chiffrement de l'IV

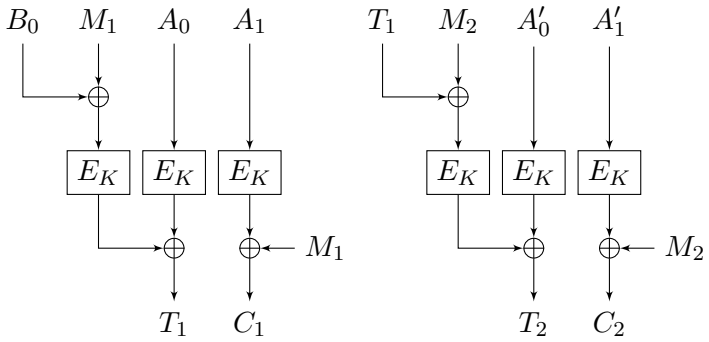
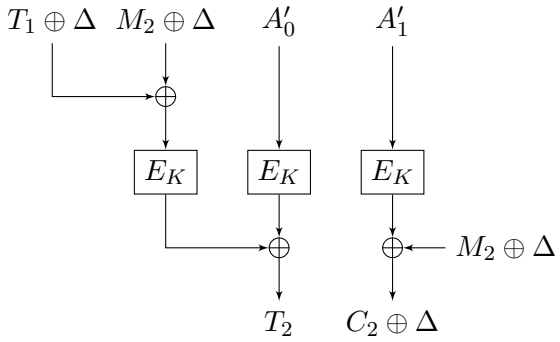


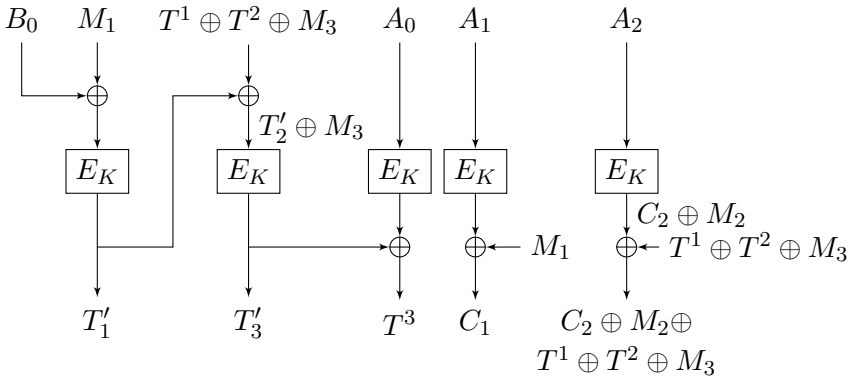
FIGURE 4.4 – Attaque de la variante de CCM sans chiffrement de l'IV



Une deuxième variante est de supprimer la longueur des données dans le bloc initial. Plus précisément, si le bloc initial B_0 est constitué uniquement de l'IV sans prendre en compte la taille du clair, un chiffré valide peut être falsifié avec $2^{\frac{2v}{3}}$ requêtes, où v représente la taille de l'IV. Cette attaque est décrite dans [FMVZ08]. Si $v < \frac{3n}{4}$, la sécurité de CCM tombe alors en dessous de la borne donnée par le paradoxe des anniversaires. Cette dernière hypothèse n'est pas totalement improbable, sachant qu'en pratique on a généralement $v < n$.

L'attaque présentée dans [FMVZ08] fonctionne comme suit. Supposons que les IV soient choisis aléatoirement et que B_0 ne contienne pas la longueur du message. Nous considérons trois types de messages : des messages d'un bloc M_1, des messages de deux blocs M_1, M_2 où M_1 est le même que précédemment, et des messages de trois blocs M_1, M_2, M_3 où M_1 et M_2 sont encore les mêmes. L'attaquant interroge alors l'oracle de chiffrement jusqu'à ce qu'une collision se produise entre les IV utilisés pour un message de chaque ensemble. Nous notons $\left(IV, T^1, C_1^1\right)$, $\left(IV, T^2, C_1^2, C_2^2\right)$ et $\left(IV, T^3, C_1^3, C_2^3, C_3^3\right)$ les chiffrés correspondant aux trois types de messages. Nous avons alors $C_1^1 = C_1^2 = C_1^3$, que nous notons alors plus simplement C_1 et $C_2^2 = C_2^3$ que nous simplifions en C_2. Nous avons alors besoin de noter T'_1, T'_2 et T'_3 les CBC-MAC de ces trois messages, c'est-à-dire avant leur chiffrement en mode compteur dans CCM. L'absence de la longueur dans les valeurs initiales fait qu'à partir d'une valeur initiale unique, A_0 ne peut aussi prendre qu'une unique valeur. D'où $T^1 \oplus T^2 = T'_1 \oplus T'_2$. Il en découle alors que le message clair $\left(M_1, T^1 \oplus T^2 \oplus M_3\right)$ a comme chiffré, pour la valeur initiale IV, $\left(T^3, C_1, C_2 \oplus M_2 \oplus T^1 \oplus T^2 \oplus M_3\right)$, comme cela est décrit à la figure 4.5.

FIGURE 4.5 – Attaque de CCM sans tenir compte de la taille des données



Dans le cas des bases de données, la taille d'un champ est souvent fixée et ne change pas. Mais, les champs n'ont généralement pas tous la même taille. Si c'était le cas, une telle attaque ne pourrait plus marcher, et donc cette variante pourrait alors être envisagée.

Finalement, une troisième solution est de chiffrer puis authentifier (EtA). En effet, comme nous l'avons vu dans la section 1.1.5, [CK01] et [Kra01] ont montré que, si l'algorithme de chiffrement sous-jacent est sémantiquement résistant à des attaques à clairs choisis et si l'algorithme d'authentification (le MAC) résiste aux attaques à messages choisis, alors le mode authentifié obtenu en combinant ses algorithmes sous la forme « chiffrer puis authentifier » est sûr.

Les deux pré-requis sont vérifiés par les modes compteur (CTR) et CBC-MAC d'AES, quand les versions des modes CTR et CBC utilisées sont les versions consolidées vues à la section 1.1.3. Il en découle que l'on peut utiliser ces modes dans ce sens pour assurer la sécurité des données.

Trois solutions suggérées pour les bases de données. Nous proposons donc trois solutions basées sur CCM pour chiffrer des bases de données. Une première solution simple et standard serait de choisir les valeurs initiales suivantes :

$$\begin{aligned} IV &= \textstyle\sum_{j=0}^{i-1} \ell_j \\ B_0 &= 1 \| n_{\text{table}} \| n_{\text{ligne}} \| n_{\text{cellule}} \| IV \| \ell_i \\ A_0 &= 0 \| n_{\text{table}} \| n_{\text{ligne}} \| n_{\text{cellule}} \| IV \| 0 \ldots 0 \end{aligned}$$

Ces valeurs initiales peuvent alors être utilisées avec CCM en utilisant AES comme chiffrement par bloc. Ainsi, toutes les conditions requises dans les preuves de sécurité de CCM sont remplies et la borne de sécurité est de $2^{\frac{n}{2}}$ requêtes, soit 2^{64} requêtes.

Algorithme 4.2 CCM modifié en utilisant « chiffrer puis authentifier ».

Entrées : Une clef secrète K.
Entrées : m données $d_1, \ldots, d_m$, de tailles en octet respectives $\ell_1, \ldots, \ell_m$.
Sorties : Le chiffré $(C_1, \ldots, C_k)$ et le tag T correspondant.

// **Chiffrement avec le mode CTR**
// Remplir $d_1, \ldots, d_m$ avec des zéros pour obtenir un multiple de n bits
$\text{Pad}(d_1, \ldots, d_m) = (M_1, \ldots, M_k)$.
$A_0 = \alpha(IV, \ell)$
pour j de 1 à k **faire**
  $A_j \leftarrow A_{j-1} + 1$
  $C_j \leftarrow E_K(A_j) \oplus M_j$
fin pour

// **Calcul du MAC**
// Initialisation du CBC-MAC
$B_0 = \beta(IV, \ell)$
$X_0 \leftarrow E_K(B_0)$
// Calcul du CBC-MAC
pour j de 1 à k **faire**
  $X_j \leftarrow E_K(X_{j-1} \oplus C_j)$
fin pour
// Calcul du tag
$T \leftarrow \text{lsb}_t(X_k)$

retourner $(C_1, \ldots, C_k, T)$.

Une seconde approche consiste à utiliser la construction « chiffrer puis authentifier ». L'algorithme 4.2 décrit une version de CCM modifiée dans ce sens. Dans cette version, nous ne calculons qu'un unique tag pour l'ensemble des données à chiffrer, mais il serait aussi possible d'en calculer un par donnée, en fonction de l'utilisation choisie. L'avantage de ces solutions est qu'il est possible de vérifier l'intégrité des données sans les déchiffrer.

Finalement, on peut calculer plusieurs MAC différents. Cette solution proposée dans l'algorithme 4.3 calcule le MAC pour chaque donnée séparément.

Algorithme 4.3 CCM avec « chiffrer puis authentifier », 2e version.

Entrées : Une clef secrète K.
Entrées : m données $d_1, \ldots, d_m$, de tailles en octet respectives $\ell_1, \ldots, \ell_m$.
Sorties : Les couples (chiffré, tag) (c_i, T_i) correspondant à chaque donnée.

pour i de 1 à m **faire**
  // **Chiffrement avec le mode CTR**
  // Remplir d_i avec des zéros pour obtenir un multiple de n bits
  $\mathrm{Pad}(d_i) = (M_1, \ldots, M_{k_i})$
  $A_0 = \alpha(IV_i, \ell_i)$
  pour j de 1 à k_i **faire**
    $A_j \leftarrow A_{j-1} + 1$
    $C_j \leftarrow E_K(A_j) \oplus M_j$
  fin pour
  $c_i = (C_1, \ldots, C_{k_i})$

  // **Calcul du MAC**
  // Initialisation du CBC-MAC
  $B_0 = \beta(IV_i, \ell_i)$
  $X_0 \leftarrow E_K(B_0)$
  // Calcul du CBC-MAC
  pour j de 1 à k_i **faire**
    $X_j \leftarrow E_K(X_{j-1} \oplus C_j)$
  fin pour
  // Calcul du tag
  $T_i \leftarrow \mathrm{lsb}_t(X_{k_i})$
  retourner (c_i, T_i).
fin pour

Chiffrement avec ajout de redondance

Nous allons maintenant étudier la sécurité de solutions de chiffrement avec ajout de redondance, et en particulier proposer des attaques contre AREA et plusieurs de ses variantes.

AREA (« Added Redundancy Explicit Authentication »). AREA (« Added Redundancy Explicit Authentication ») fut initialement proposé dans [Fru05]. Pour garantir l'intégrité, l'idée est d'utiliser un mode opératoire de chiffrement à propagation d'erreur infinie [Knu00] comme, par exemple CBC. En effet, avec ce mode, si l'on note $(m_i)_{1 \leq i \leq n}$ les blocs de clair et $(c_i)_{1 \leq i \leq n}$ les blocs de chiffré correspondant, c_i ne dépend pas seulement de m_i, mais aussi de m_1, …, m_{i-1}. AREA consiste alors à ajouter un bloc ℓ

à la fin du message clair à chiffrer. Si le mode opératoire utilisé est, comme CBC, à propagation d'erreur, une modification du chiffré peut se voir si l'on connaît préalablement certains blocs de clair. Ainsi, AREA consiste à chiffrer un message auquel on a ajouté un tel bloc ℓ à la fin, en ajoutant la valeur claire de ℓ au début du chiffré. Autrement dit, on transmet $(\ell, c_0, \ldots, c_n)$ où $(c_0, \ldots, c_n)$ est le chiffré de $(m_0, \ldots, m_{n-1}, \ell)$, comme on le voit à la figure 4.6 où le mode opératoire utilisé est le mode CBC. Le déchiffrement procède normalement, mais la valeur ℓ contenue dans le chiffré doit être comparée au dernier bloc déchiffré, comme l'illustre la figure 4.7. S'ils sont égaux, le message n'a pas été altéré, dans le cas contraire, il l'a été.

Nous allons commencer par montrer que la version d'AREA présentée dans [Fru05] n'apporte en fait aucune intégrité supplémentaire dans le cas d'une utilisation avec le mode CBC. La figure 4.6 illustre ce cas de figure. Dans le cas du mode CBC, le chiffré communiqué est $(\ell, c_{-1}, c_0, \ldots, c_{n+1})$, mais peut être réduit à $(\ell, c_0, \ldots, c_{n+1})$ si l'IV utilisé est 0, ce qui est généralement le cas avec le mode CBC.

FIGURE 4.6 – AREA [Fru05]

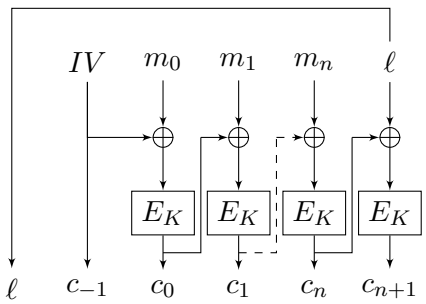
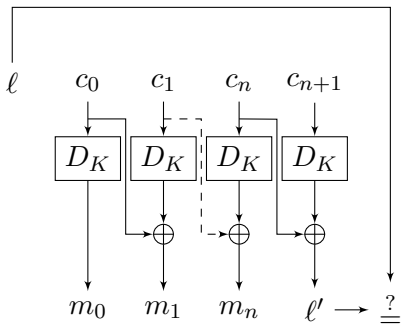


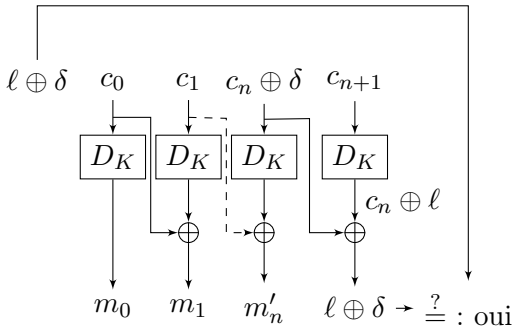
FIGURE 4.7 – Déchiffrement avec AREA



La première erreur de cette solution est de confondre la propagation d'erreur partielle du déchiffrement du mode CBC et la propagation d'erreur infinie pour le chiffrement de ce mode. En effet, lors du chiffrement, une modification d'un bloc de clair m_i modifie tous les blocs de chiffré d'indice supérieur ou égal à i. Mais, une erreur dans un bloc de chiffré c_i n'affecte que le déchiffrement des blocs c_i et c_{i+1}, étant donné que m_i ne dépend que de c_i et c_{i-1}.

Il en découle qu'une modification de n'importe quel bloc de chiffré c_i ayant un indice i inférieur ou égal à $n-1$ ne modifiera pas la valeur de ℓ lors du déchiffrement, et passera donc inaperçue. De plus, il est aussi possible de modifier le bloc c_n. Si le format du bloc ℓ n'est pas précisé autrement qu'étant une valeur aléatoire, on remarque que, pour toute différence δ, le chiffré $(\ell \oplus \delta, c_0, c_1, \ldots, c_{n-1}, c_n \oplus \delta, c_{n+1})$ est un chiffré valide dont le clair est de la forme $(m_0, \ldots, m_{n-1}, m'_n, \ell \oplus \delta)$ et diffère donc de $(m_0, \ldots, m_n, \ell)$. Cette attaque est décrite à la figure 4.8.

FIGURE 4.8 – Une attaque contre AREA



Mais la valeur de ℓ peut dépendre des caractéristiques du message clair, par exemple en étant dérivée d'un numéro de version, d'un timestamp, d'un numéro de table, colonne, ligne dans le cas de bases de données... Dans ce cas, il est encore possible de modifier le dernier bloc d'un chiffré C, cette fois à partir de la connaissance d'un autre chiffré C' valide. Soient donc deux chiffrés $C = (\ell, c_0, \ldots, c_{n+1})$ et $C' = \left(\ell', c'_0, \ldots, c'_{n+1}\right)$ et leurs clairs correspondant respectifs $M = (m_0, \ldots, m_n)$ et $M' = (m'_0, \ldots, m'_n)$. Soit alors :

$$C'' = \left(\ell, c'_0, \ldots, c'_{n-1}, c'_n \oplus \left(\ell \oplus \ell'\right), c'_{n+1}\right).$$

C'' est alors un chiffré valide pour remplacer C car le même ℓ est utilisé et :

$$\begin{aligned} E_K\left(c'_n \oplus (\ell \oplus \ell') \oplus \ell\right) &= E_K\left(c'_n \oplus \ell'\right) \\ &= c'_{n+1}, \end{aligned}$$

ce qui vérifie bien la condition imposée par AREA.

Il en découle donc que la solution AREA proposée dans [Fru05] ne peut être une solution générique pour apporter de l'intégrité en plus de la confidentialité pour n'importe quel mode de chiffrement à propagation d'erreur infinie.

« Block-Level AREA ». La même idée est utilisée dans [ETS+06] sous le nom PE-ICE (« Parallelized Encryption and Integrity Checking Engine »). Ici, le but était de garantir la confidentialité et l'intégrité des données échangées entre un SoC (« System on Chip », système sur puce) et sa mémoire externe. Le bloc de mémoire à chiffrer fait 256 bits. Pour se faire, il est découpé en trois blocs de 96, 96 et 64 bits, et le dernier est rempli pour atteindre aussi les 96 bits. Les trois blocs se voient alors ajouter 32 bits de valeurs qui serviront à vérifier l'intégrité. Le chiffrement se fait alors avec AES en mode ECB.

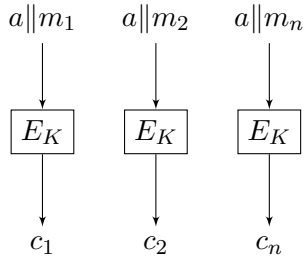
Cette méthode est renommée « Block-Level AREA » dans [ECL+07], qui l'utilise cette fois pour assurer la confidentialité et l'intégrité des données transférées entre un SoC et une RAM (« Random Access Memory »).

Un tel ajout n'a pas d'effet sur la confidentialité ; en effet, cela est juste un cas particulier de remplissage, comme étudié au paragraphe 1.1.2.

Il convient donc d'étudier l'intégrité garantie par un tel procédé. Le modèle est le suivant. On souhaite chiffrer, avec un algorithme de chiffrement par bloc, par exemple AES, présenté à la page 19, travaillant sur des blocs de ℓ bits, une donnée de taille $\ell - m$ en lui accolant une valeur de taille m. [ETS+06] et [ECL+07] prétendent qu'un échange d'une valeur ainsi chiffrée par un bloc calculé aléatoirement ne peut alors se faire qu'avec une chance sur 2^{-m}. Comme cela est expliqué à la section 9.6.5 du [MvOV97], il faut remarquer qu'il n'y a pas ici à proprement parler de séparation entre la sécurité de la confidentialité et celle de l'intégrité. Plus précisément, la sécurité est bornée par celle de l'algorithme de chiffrement, et donc une recherche exhaustive sur ses clefs permet non seulement d'attaquer la confidentialité, mais aussi l'intégrité.

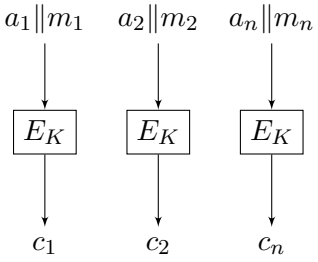
Pour des messages de taille supérieure à ℓ bits, [ETS+06] propose d'utiliser le mode ECB de manière similaire, comme décrite à la figure 4.9.

FIGURE 4.9 – Mode ECB-AREA « naïf »



Le mode décrit à la figure 4.9 présente néanmoins une trop grande faiblesse. Le fait que tous les ajouts[2] soient identiques permet des interversions de blocs, et expose les égalités entre les m_i. Nous considérerons donc une version plus robuste de ECB-AREA décrite à la figure 4.10, où l'on utilise des ajouts différents pour chiffrer chaque bloc, par exemple en utilisant un compteur, ou l'adresse des blocs. On supposera aussi dans la suite de l'exposé que deux ajouts identiques ne peuvent pas être utilisés avec la même clef, même pour chiffrer deux clairs différents.

FIGURE 4.10 – Mode ECB-AREA



De plus, il est nécessaire que l'ajout respecte certaines contraintes, par exemple en contenant la taille du message à chiffrer. En effet, sans un tel encodage, si le chiffré (c_1, c_2) de $(a\|m_1, a\|m_2)$ est connu, alors c_1 est trivialement un chiffré valide de $(a\|m_1)$. Donc « Block-Level AREA » n'est pas sûr si l'encodage du message clair n'est pas un encodage sans préfixe.

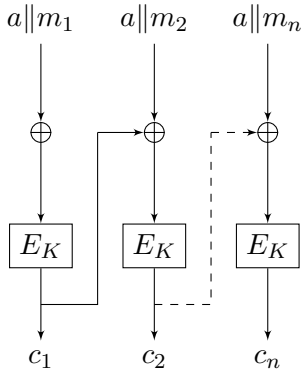
Alors, il convient de comparer le coût de cette proposition par rapport aux propositions existantes. Prenons, par exemple, le cas présenté dans [ETS+06]. Pour protéger 256 bits de données, « Block-Level AREA » requiert ici le chif-

[2] Nous utilisons dans la suite le terme d'« ajout » à la place de « redondance » pour insister sur le fait que cette valeur ne dépend pas de la donnée M à chiffrer.

frement de trois blocs avec l'AES. Une méthode classique de chiffrement authentifié aurait coûté, si celui-ci était à deux passes, quatre chiffrements avec AES, d'où un coût plus élevé d'un tiers par rapport à « Block-Level AREA ». OCB, présenté à la section 1.1.5, aurait lui aussi requis trois chiffrements, étant donné qu'il nécessite un chiffrement par bloc, plus un chiffrement pour l'authentification. Et un chiffrement authentifié basé sur la méthode Encrypt-then-MAC [BN00, ISO09] où le MAC serait calculé avec HMAC ne nécessiterait que deux chiffrements et le hachage de deux blocs. On voit donc que « Block-Level AREA » n'apporte finalement pas un réel gain de temps de calcul si on compare aux solutions alternatives existantes. De plus, pour stocker le résultat, « Block-Level AREA » nécessite trois blocs entiers, alors que les autres solutions proposées ne nécessitent que deux blocs, plus un bout du troisième pour stocker le tag. La taille de celui-ci conditionne l'intégrité. Si une sécurité de 2^{32} est suffisante, comme dans le cas de [ETS+06], il suffit d'avoir un tag de 32 bits, soit le quart de la taille additionnelle requise par « Block-Level AREA ».

Une variante de « Block-Level AREA », appelée CBC-AREA est proposée dans [ABG] et décrite à la figure 4.11. Comme dans « Block-Level AREA », l'idée est de découper la donnée M à chiffrer en blocs plus petits que ceux pris en entrée par l'algorithme de chiffrement par bloc E. Il est ainsi possible d'y ajouter une valeur a pour compléter le bloc.

FIGURE 4.11 – Mode CBC-AREA



Un numéro de version v peut aussi être ajouté au premier bloc pour s'assurer que la valeur ainsi ajoutée au premier bloc est bien distincte pour chaque donnée chiffrée. Dans la solution proposée dans [ABG], a fait 32 bits et v 16 bits. Sinon, il est aussi possible d'inclure ce numéro de version dans a et ainsi s'assurer que deux chiffrés différents n'utilisent jamais deux valeurs d'ajout identiques. C'est ce cas que nous étudierons dans la suite de cette section.

Pour faciliter les explications dans la suite de l'étude des modes CBC-AREA et ECB-AREA, nous utiliserons les notations suivantes :

- $m = |a_i| = |a|$, la taille de l'ajout,
- A l'ajout total : $A = (\underbrace{a, \ldots, a}_{n})$ pour CBC-AREA et $A = (a_1, \ldots, a_n)$ pour ECB-AREA,
- un schéma de chiffrement ainsi construit $\mathcal{E}$ prend en argument une clef K, un message M et un ajout A et retourne un chiffré C, en utilisant un algorithme de chiffrement par bloc sous-jacent E :

$$C = \mathcal{E}(K, M, A).$$

On peut aussi remarquer qu'ainsi $|A| = n \times m$.

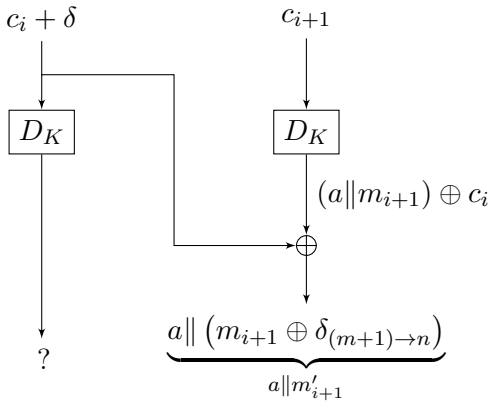
Une première remarque sur ces modes est que la taille nécessaire pour stocker un chiffré dépasse rapidement celle d'un chiffré avec un mode authentifié conventionnel. Soit t la taille du tag dans un chiffrement authentifié conventionnel. La taille des chiffrés générés par CBC-AREA et ECB-AREA dépasse celle d'un chiffré classique dès que $n \times m \geq t$. Nous avons vu que les valeurs habituelles sont $t = 64$ bits, par exemple pour CCM, et $m = 32$ bits. Il en découle que la taille du chiffré ne sera plus petite si l'on utilise CBC-AREA ou ECB-AREA que pour un clair de moins de deux blocs de 96 bits, soit 192 bits. Notons que cette comparaison ne tient compte que des tailles usuelles et ne se fait pas à sécurité égale.

Une seconde remarque immédiate sur ces deux modes est qu'une altération aléatoire d'un unique bloc de chiffré affectera deux blocs de clair dans le cas de CBC-AREA, à l'exception du dernier, contre un seul dans celui de ECB-AREA.

Nous allons évaluer la résistance de ces deux modes CBC-AREA et ECB-AREA à deux types d'attaques. Le premier type d'attaques consiste, pour un attaquant, à essayer de modifier un chiffré existant, sans infirmer sa validité. Autrement dit, à partir d'une donnée chiffrée C et de la taille de l'ajout m, l'attaquant doit trouver une donnée C', différente de C, valide, dont le clair correspondant a un ajout A identique à celui de C. Dans le cas du mode ECB-AREA, en supposant l'algorithme de chiffrement utilisé E comme sûr, si l'attaquant altère un bloc de chiffré, après déchiffrement, le clair sera valide avec probabilité 2^{-m}, et ceci quel que soit le bloc de chiffré modifié. Si C était chiffré avec le mode CBC-AREA, l'altération du dernier bloc produit trivialement un clair valide avec la probabilité 2^{-m}. Ensuite, si l'on souhaite changer un autre bloc de chiffré c_i en $c_i \oplus \delta$ de façon totalement aléatoire, le chaînage propre au mode CBC implique une probabilité de succès de seulement 2^{-2m}, car ainsi les deux blocs m_i et m_{i+1} seront impactés. Il est néanmoins possible de faire une meilleure attaque. En effet, si

la modification ne porte que sur les $n - m$ derniers bits de c_i, par exemple avec $\delta = \underbrace{0, \ldots, 0}_{m} \| \delta_{m+1}, \ldots, \delta_n$ et $c'_i = c_i + \delta$, alors m'_i aura une probabilité 2^{-m} d'être valide et m'_{i+1} sera lui valide, car ses m premiers bits seront bien toujours égaux à a, comme l'illustre la figure 4.12. Il en découle que la probabilité de succès d'une telle attaque est identique dans les deux cas et donc que le mode CBC-AREA n'apporte pas plus de sécurité que le mode ECB-AREA contre ce type d'attaque.

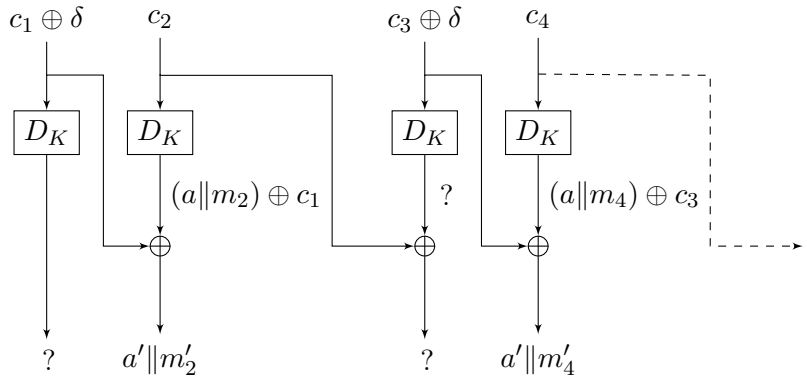
FIGURE 4.12 – Une attaque contre CBC-AREA



Le second type d'attaques que l'on peut envisager est la génération de chiffrés valides à partir d'autres chiffrés connus, en modifiant la valeur de l'ajout. Un attaquant a ici à sa disposition un chiffré C calculé à partir d'un ajout A. Il doit trouver un chiffré C' valide pour une valeur d'ajout A', différente de celle de A. Dans le cas du mode ECB-AREA, chaque bloc étant chiffré indépendamment, la probabilité d'obtenir un chiffré valide est de $2^{-|A|} = 2^{-n \times m}$. Par contre, dans le cas du mode CBC-AREA, il est possible d'exploiter le chaînage du mode CBC pour avoir une plus grande probabilité de succès. Dans ce cas, nous avons $A = (\underbrace{a, \ldots, a}_{n})$ et $A' = (\underbrace{a', \ldots, a'}_{n})$. Notons alors $\delta = (a \oplus a') \| 0$. Soit alors $C' = c_1 \oplus \delta, c_2, c_3 \oplus \delta, c_4, \ldots, c_n$ et $M' = m'_0, \ldots, m'_n$ le clair obtenu après déchiffrement de C'. Comme l'illustre la figure 4.13, les blocs de clair d'indice pair sont tous valides et ceux d'indice impair ont une probabilité 2^{-m} de l'être. Une autre possibilité pour cette attaque consiste à insérer les différences sur les blocs d'indices pairs de chiffré. Cela est néanmoins moins intéressant. Les blocs d'indice impair sont alors valides et ceux d'indice pair ont une probabilité 2^{-m} de l'être. Par contre, le premier bloc sera alors toujours invalide si son chiffré est c_1 et a une probabilité 2^{-m} de l'être sinon. Donc, dans le cas où on insère la différence

aux blocs d'indice pair, la probabilité de succès de cette attaque sur le mode CBC-AREA est $2^{-\lfloor \frac{n+1}{2} \rfloor m}$. Pour ce second type d'attaques CBC-AREA assure donc une sécurité nettement moins bonne que ECB-AREA.

FIGURE 4.13 – Une (autre) attaque contre CBC-AREA



Il découle de ces résultats que le mode CBC-AREA ne présente pas de gain de sécurité par rapport à ECB-AREA, et est même dans certains cas nettement plus vulnérable.

Chiffrement avec redondance. Il existe néanmoins dans la littérature cryptographique [MvOV97, AB01] des pistes pour combiner chiffrement et ajout de redondance dans le but de garantir aussi l'intégrité. L'idée ici est toujours de chiffrer $M\|\ell$, mais en faisant, cette fois, dépendre ℓ de M, c'est-à-dire avec $\ell = h(M)$. La redondance ℓ est donc une somme de contrôle (« checksum »), un code de détection de manipulation (MDC, « Manipulation Detection Code »), un MAC... Nous ne nous concentrerons ici que sur les cas où ℓ n'est pas calculé avec un MAC, cela étant déjà étudié à la section 1.1.5. Ce cas est étudié dans la section 9.6.5 du chapitre 9 du *Handbook of applied cryptography* [MvOV97]. La première remarque est que si la fonction h est sans clef (somme de contrôle ou MDC), la sécurité de l'ensemble est limitée par celle du chiffrement, en effet, il suffit de récupérer la clef utilisée dans le chiffrement pour pouvoir accéder au message et forger des messages. La confidentialité et l'intégrité ne sont pas ici assurées de façons indépendantes. De plus, ce type de construction est génériquement vulnérable aux attaques à clairs connus.

Deux exemples utilisant le mode opératoire CBC avec des codes de détection de manipulation h différents sont proposés dans cet ouvrage. Le premier cas, appelé CBCC, consiste à utiliser un XOR des différents blocs comme redondance, soit $\ell = h(M) = \bigoplus_{i=1}^{n} m_i$. Cette construction ne garantit évi-

demment pas l'intégrité du message car un échange de deux blocs ne peut pas être détecté, pas plus qu'un ajout de deux blocs identiques.

Une seconde solution consiste à utiliser une somme modulo $2^k - 1$, où k est la taille d'un bloc, c'est-à-dire $\ell = h(M) = \sum_{i=1}^{n} m_i \mod 2^k - 1$. Cette solution protège contre l'ajout de deux blocs identiques.

Les modes authentifiés à une passe OCB et PCBC vus à la section 1.1.5 sont légèrement plus complexes, mais ont une construction similaire. En particulier, ils ne requièrent guère plus d'appels à la fonction E que le nombre de blocs à chiffrer.

[AB01] explore la notion de chiffrement avec redondance plus en profondeur, et en particulier, s'intéresse aux niveaux de sécurité prouvés que l'on peut atteindre avec différentes constructions génériques et avec quelques exemples précis. Les niveaux de sécurité requis pour le chiffrement sont IND-CPA, NM-CPA et IND-CCA, définis à la section 1.1.1. Ce papier considère deux types de redondance : la redondance publique, c'est-à-dire que la fonction h utilisée ne dépend pas d'une clef, ou que celle-ci est publique, et la redondance privée, où la fonction h utilisée est paramétrée par une clef privée. Génériquement, il existe des chiffrements symétriques IND-CPA, IND-CPA et NM-CPA avec lesquels, quelle que soit la redondance publique utilisée, l'intégrité n'est pas assurée. De même, il existe des chiffrements symétriques IND-CPA avec lesquels, quelle que soit la redondance privée utilisée, l'intégrité n'est pas assurée. Notons que cela n'implique pas que, pour un chiffrement et une redondance publique donnés, le schéma correspondant soit faible. Ce résultat de [AB01] dit seulement qu'il n'existe pas de construction générique de ce type. Par contre, en rajoutant des conditions de sécurité pour la redondance privée utilisée, il existe des chiffrements symétriques IND-CPA et NM-CPA qui assurent l'intégrité, quelle que soit la redondance vérifiant ces critères utilisée.

Une des solutions dont la sécurité est prouvée est l'utilisation, comme fonction h, de HMAC, décrit au paragraphe 1.1.4, où la clef utilisée dans le MAC est publique. Sinon, il est aussi possible d'utiliser une fonction h utilisant cette fois une clef secrète, mais elle ne nécessite pas les mêmes besoins de sécurité. Les fonctions vérifiant le niveau de sécurité AXU (« Almost XOR Universal ») sont des candidats adéquats. [Kra94] en propose une famille basée sur les LFSR, et donc particulièrement efficace, comme nous l'avons vu à la section 1.1.2.

Nous avons donc vu plusieurs solutions pour chiffrer de façon sûre la table d'une base de données. Une autre piste intéressante, mais pour laquelle aucun standard n'existe actuellement, serait d'avoir un mode de chiffrement adaptable authentifié, ce qui nous permettrait de pouvoir cumuler les avantages des solutions précédentes.

4.2 Comment accélérer les requêtes : les principes

Une fois la table correctement chiffrée arrive le point central du chiffrement des bases de données, c'est-à-dire l'accélération des requêtes. En fait, grâce aux index, les bases de données permettent de pouvoir rechercher des informations dans les tables de façon efficace. Nous allons donc voir comment procéder pour accélérer les requêtes avec une base de données chiffrée. Nous allons dans un premier temps passer les solutions possibles en revue avant de les explorer plus en détail ultérieurement dans ce chapitre. Nous verrons qu'il existe tout d'abord des solutions simples mais non satisfaisantes. Ensuite une première solution acceptable est de chiffrer un index « normal » correctement. Puis nous verrons qu'il existe des chiffrements particuliers dédiés aux bases de données. Finalement, il est aussi possible d'adapter des algorithmes existants, en particulier en ajoutant des données de façon à aider les recherches.

4.2.1 Pourquoi protéger l'index ?

Lorsque l'on souhaite accélérer les recherches dans une base de données, comme nous l'avons étudié dans la section 1.2 et plus précisément dans la section 1.2.3, il est naturel d'utiliser des index. L'index est l'outil fondamental en base de données pour accélérer les requêtes et ainsi effectuer des recherches efficaces dans la base.

La première idée, certes naïve, serait de faire un index sur les données en clair, c'est-à-dire l'index qui serait fait si l'on ne chiffrait pas du tout la base de données. Si l'on fait cela, on révèle non seulement les clairs correspondant aux données chiffrées, mais aussi leur position dans la table, grâce aux pointeurs liant l'index à la table. Bien que cela permette d'avoir, dans les requêtes, une efficacité similaire au cas où rien n'est chiffré, cette façon de procéder n'est donc pas du tout intéressante vu qu'elle n'apporte aucune sécurité mais détériore quand même les performances, la table étant désormais chiffrée.

Il en découle qu'il faut faire autre chose pour accélérer les requêtes lorsque l'on souhaite chiffrer la base de données et que chiffrer la table de façon sûre (bon algorithme de chiffrement, IV bien choisis...) n'est pas suffisant pour assurer la qualité de la protection de la base de donnée. Si l'on souhaite faire un index, celui-ci, ainsi que le lien entre la table et l'index, doivent aussi être protégés. En particulier, il faut ici bien faire attention au niveau auquel se situe le chiffrement. Si celui-ci est au niveau applicatif, il n'y a naturellement aucune clef protégeant les données de la base sur le serveur. En particulier, les pointeurs entre l'index et la base ne seront pas chiffrés dans ce cas, sauf si le temps de latence entre le client et le serveur est très faible. Hormis ce cas, il en découle, que par exemple, si l'index reste en clair, un attaquant n'aurait même pas besoin de s'attaquer au chiffrement de la table pour la déchiffrer.

Si le chiffrement est au niveau de la base de données, il devient alors plus aisé de chiffrer les pointeurs liant l'index à la base de donnée, et ainsi limiter l'impact d'un chiffrement de l'index n'offrant que peu de garanties.

La première chose à faire est donc de comprendre ce qu'un attaquant pourra retirer d'un index ou d'une autre solution permettant d'accélérer les requêtes. Un éventuel attaquant essayera d'obtenir des informations sur la base de données à partir de deux principaux endroits : l'ordinateur de l'utilisateur, et le serveur. Si l'attaquant se situe au niveau du poste client, il n'a, a priori, pas accès aux index, donc leur chiffrement n'affecte pas la sécurité globale. Par contre, dans ce cas, il peut légitimement accéder à toute donnée et les protections sur le serveur deviennent caduques. Si l'attaquant se situe au niveau du serveur, il pourra voir les index. C'est donc dans ce cas qu'il est primordial de limiter la portée de l'utilisation des informations révélées par l'index.

Après avoir exposé dans [HMI02] les défis crytographiques à relever pour parvenir à protéger les bases de données externalisées en assurant une bonne efficacité du système de gestion de base de données, les mêmes auteurs détaillent dans [HHIM07] une revue de l'évolution de la recherche dans ce domaine. Ils remarquent alors que presque toutes les solutions que l'on peut trouver dans la littérature suggèrent l'utilisation de moyens de chiffrement pour protéger les données. Comme nous l'avons vu, ces données peuvent être chiffrées de nombreuses façons différentes, à différents niveaux, et à différentes granularités. Il en découle que le point clef pour rendre ces données utilisables est l'implémentation des fonctionnalités permettant d'effectuer des requêtes et modifications sur des données chiffrées.

[HHIM07] sépare les techniques en deux catégories :

- les approches basées sur de nouvelles techniques de chiffrement,
- les approches basées sur la dissimulation d'information.

La première approche consiste à chiffrer de façon particulière la table, en général pour effectuer ensuite un index de façon transparente dessus, mais cela est un choix indépendant. La seconde approche consiste à construire un index en place, en ajoutant à chaque n-uplet des méta-données. Nous proposerons plus loin une troisième façon de procéder, qui est de créer un index dissocié, c'est-à-dire, si l'on considère que la table chiffrée est composé de relations $(i, E_K(i, m))$, l'index sera composé de couples $(F_{K'}(m), G_{K''}(i))$, où G permet d'empêcher un lien direct entre l'index et la table grâce à la clef secrète K''.

Nous nous concentrerons sur les solutions à base de chiffrement symétrique qui sont les seules suffisamment rapides pour être susceptibles d'avoir une réelle application pratique pour cette problématique. Mais, il existe aussi des propositions basées sur du chiffrement à clef publique. En particulier,

[BBO07] propose un chiffrement déterministe basé sur RSA-OAEP permettant des recherches efficaces, ainsi que des notions de sécurité pour mesurer sa résistance théorique. Une des limitations principales de cette proposition réside dans l'utilisation d'un algorithme de chiffrement lent qui n'autorise pas un traitement suffisamment rapide des données dans la plupart des cas.

Avant d'étudier les deux approches détaillées dans [HHIM07], nous allons voir les solutions pour chiffrer un index qui serait créé sur les données en clair avant d'être chiffré. L'idée ici est donc de partir d'un index $\mathcal{I}$ créé sur les données de la table en clair. Cet index peut être de n'importe quel type vu à la section 1.2.3, et en particulier un arbre ou une table de hachage. Dans tous les cas, il y a deux éléments constitutifs distincts : les clefs de recherche et les pointeurs. Comme nous l'avons déjà vu, les premiers peuvent être un haché d'une valeur de la table, une valeur intermédiaire, ou une valeur de la table. Les seconds sont soit des pointeurs vers un autre élément de l'index, soit des pointeurs vers la table. Pour chiffrer cela, il faut tout d'abord choisir le niveau de chiffrement. Il peut y en avoir ici principalement trois : chiffrement global de l'index, chiffrement au niveau des entrées de l'index, de la forme {clef de recherche, pointeur} et chiffrement de chaque élément distinctement. Le premier choix n'est évidemment pas celui auquel nous allons nous intéresser. Pour choisir entre les deux autres, il faut tenir compte du niveau du chiffrement : niveau de la base de données ou niveau applicatif. À partir de cela, nous étudierons les différentes possibilités dans la section 4.3.

4.2.2 Utiliser des chiffrements particuliers sur la table

Comme nous le verrons à la section suivante, chiffrer l'index normal de façon sûre comme évoqué précédemment est relativement lent. La communauté de bases de données, et dans une moindre mesure celle de cryptographie, a étudié la conception de chiffrements qui auraient des propriétés particulières permettant de pouvoir parcourir les index sans déchiffrer. Cela s'applique plutôt aux index sous forme d'arbre où il faut généralement comparer la valeur recherchée avec celle contenue dans les nœuds et les feuilles de l'arbre, ainsi qu'aux tables de hachage où il faut comparer le haché déduit de la valeur recherchée avec ceux contenus dans la table de hachage servant d'index. Ces comparaisons sont évidement moins chères dans les cas où aucun déchiffrement n'est à effectuer. D'où des propositions d'algorithmes de chiffrement permettant de telles fonctionnalités. Nous étudierons les principales propositions en ce sens, ainsi que leur sécurité à la section 4.4.

4.2.3 Dégrader des algorithmes existant en ajoutant de la redondance

Il est possible de chiffrer correctement l'index, comme nous le verrons à la section 4.3, mais aussi d'utiliser d'autres propriétés pour accélérer les requêtes. Nous verrons à la section 4.5 qu'il y a plusieurs façons de faire cela, mais l'idée principale est d'ajouter de l'information en plus du chiffré. Cette information supplémentaire sert à rendre les requêtes plus efficaces. Elle peut être de nature diverse, comme par exemple indiquer un intervalle dans laquelle une valeur se situe.

4.2.4 Utiliser des index sécurisés

Finalement, si l'on se contente de recherches sur des cas d'égalité, il est aussi possible d'utiliser la notion d'index sécurisé que nous avions introduite à la section 3.3.1. Nous détaillerons cette piste à la section 4.6.1.

4.3 Chiffrer l'index

4.3.1 Propositions pour chiffrer un index

La littérature concernant le chiffrement des bases de données propose aussi des solutions pour chiffrer un index.

[EWSG04] propose l'utilisation de chiffrements standards. Les auteurs de cet article conseillent d'utiliser deux façons de chiffrer distinctes pour la table et pour l'index. La particularité principale présentée ici est la volonté d'assurer un minimum l'intégrité des données, ou du moins s'assurer qu'elles n'ont pas été déplacées. Ici la base est constituée de tables t, qui sont composées de lignes r et de colonnes c. De façon similaire l'index contient des lignes i. Si M_{trc} est la valeur à chiffrer dans la table, et qu'elle se situe à la ligne i de l'index, sa valeur chiffrée dans la table sera alors $E_k\left(M_{trc} \| \mu(t,r,c)\right)$ et sa valeur dans l'index sera $E_k\left(M_{trc} \| i\right)$, où E est l'algorithme de chiffrement choisi et où la fonction μ doit répondre à certaines conditions décrites dans [EWSG04]. Cette façon de procéder a plusieurs limitations. La première est l'augmentation de la taille du chiffré par rapport à celle du clair. Ensuite, si le clair est plus gros que la taille du bloc géré par E, le chiffrement ainsi proposé sera déterministe pour tous les blocs sauf le dernier. Dans le cas contraire, si E est un chiffrement par bloc et si $M_{trc} \| i$ est plus petit que la taille d'un bloc, l'index ne permet même pas d'effectuer des comparaisons sur le chiffré pour les cas d'égalités, car pour i différent de j, on aura $E_k\left(M \| i\right) \neq E_k\left(M \| j\right)$.

[Küh06] apporte une étude de sécurité plus poussée de [EWSG04] et expose des attaques contre ce schéma. Il propose aussi une façon de le corriger en utilisant du chiffrement authentifié, que nous avons vu à la section 1.1.5.

L'idée est ici d'utiliser un IV pour rendre le chiffrement non déterministe, et d'utiliser les valeurs définissant la position des données (t, r, c) comme des valeurs associées, au sens du chiffrement authentifié avec données associées (AEAD). Pour rappel, un tel chiffrement permet non seulement de chiffrer et d'authentifier des données, mais aussi d'authentifier des données qui ne sont pas chiffrées. C'est ce que [Küh06] propose de faire avec le triplet (t, r, c) dans le cas du chiffrement de la table, et avec l'indice i du nœud dans un arbre ou de la cellule dans une table de hachage dans le cas du chiffrement de l'index. En effet, les méthodes proposées pour chiffrer la table et l'index sont identiques. Le procédé obtenu bénéficie alors de la sécurité du chiffrement authentifié utilisé, et en particulier, il assure l'intégrité des données. Par contre, il présente deux principaux écueils : l'augmentation de la taille du chiffré par rapport au clair, ainsi que la nécessité de déchiffrer pour effectuer des comparaisons dans l'index. Mais ce problème peut être aisément résolu grâce à un chiffrement utilisant un mode adaptable, comme vu à la section 1.1.2, si l'on ne souhaite pas garantir l'intégrité des données.

Les filtres de Bloom décrits à la section 1.2.3 sont des outils courants dans le domaine des bases de données. Il est donc naturel de les retrouver dans certaines propositions. En particulier, [Goh03] propose de s'aider de filtres de Bloom pour classer les données avant de chiffrer les filtres pour construire l'index. Une implication indésirable de cette façon de procéder est la présence de faux positifs dans les résultats.

4.3.2 Chiffrer l'index de façon sûre

La première solution pour chiffrer l'index est d'utiliser une des techniques vues pour chiffrer la table à la section 4.1. Ici, la valeur publique S dont sera déduit l'IV ou le tweak sera aussi définie par la position de l'élément, la table indexée, ainsi que l'indice de la ou des colonnes indexées. Par exemple dans un arbre, on utilisera une numérotation prédéfinie pour l'ordre des nœuds. Il faut cependant faire attention à ne pas utiliser la même clef, ou s'assurer qu'aucune valeur S utilisée ici ne corresponde à une valeur utilisée pour chiffrer la table dans le cas où l'on utilise le même algorithme de chiffrement dans les deux cas car cela pourrait permettre d'identifier des valeurs identiques entre la table et l'index. La pire configuration possible, serait d'utiliser les mêmes chiffrés dans la table et l'index, ce qui aurait pour effet, dans le cas d'un index en arbre de révéler l'ordre des valeurs des clairs.

Concernant l'IV ou le tweak, nous avons néanmoins les mêmes limitations que celles expliquées à la section 4.1.2, c'est-à-dire qu'il ne faut en aucun cas utiliser deux fois le même IV pour chiffrer deux valeurs différentes. Comme nous l'avons aussi vu, utiliser deux fois le même tweak est possible. Le but d'un index étant d'être performant, nous choisirons donc de chiffrer en utilisant un mode opératoire adaptable, avec un tweak dépendant de la valeur

suivante :

$$n_{\text{base}} \| n_{\text{table}} \| n_{\text{colonne indexée}} \| n_{\text{cellule dans l'index}}$$

Nous avions vu à la section 1.2.3 qu'un élément d'un index était en fait un couple {clef de recherche, pointeur}, ou un couple {clef de recherche, pointeurs} si plusieurs pointeurs ont la même clef de recherche. Idéalement, il faut donc ici chiffrer le couple {clef de recherche, pointeur}, et pas seulement la clef de recherche.

À la section 4.1, nous avions fait une distinction entre les bases classiques et les bases séquentielles. Cette différentiation est encore pertinente ici. En particulier, dans le cas des bases de données séquentielles, il est à la fois plus simple et plus efficace d'utiliser un mode opératoire classique. En effet, dans ce cas précis, la séquentialisation de la base assure que le même IV ne sera jamais utilisé deux fois s'il inclut le numéro de version de la base.

Les solutions proposées ici apportent une très bonne sécurité, identique à celle apportée pour le chiffrement de la table. Par contre, parcourir l'index n'est pas gratuit. Dans le cas où le chiffrement est effectué au niveau de la base de donnée, il faut effectuer des déchiffrements à chaque requête. Dans le cas d'un index en arbre, il en faut un par nœud parcouru pendant la requête. Dans le cas d'un chiffrement au niveau applicatif, de tels déchiffrements ne peuvent avoir lieu sur le serveur de bases de données, et il faut donc une communication entre celui-ci et le client pour chaque nœud parcouru pendant la requête pour pouvoir déchiffrer les données de l'index. Cela n'est donc envisageable que dans le cas où les communications sont très peu coûteuses.

4.3.3 Chiffrer l'index en autorisant certaines requêtes

La solution précédente a ses limites en termes de performances. Pour cela, il peut être intéressant de la modifier légèrement. Supposons que l'index est chiffré avec une granularité telle que son chiffré est composé de couples de la forme $(E_K(IV, m_i), E_K(IV', i))$ où les (i, m_i) correspondent aux couples (clef, valeur) pour la colonne considérée. Il est alors possible de permettre des requêtes portant sur des égalités, à défaut d'autoriser les comparaisons. Pour cela, il faut que la valeur publique S qui détermine l'IV ne dépende plus que de la base, de la table et de la colonne indexée, et non plus des éléments chiffrés. Elle est ainsi commune à tout l'index. Alors pour chercher un élément dans cet index, il suffit de le chiffrer et de chercher cette valeur dans l'index.

Si l'on n'est intéressé que par des recherches portant sur des égalités, le type d'index utilisé pourra être une table de hachage, que nous avons décrite à la section 1.2.3. L'idée ici est de chiffrer la clef de recherche qui est le haché des valeurs, séparément des pointeurs. Ne pas chiffrer les pointeurs n'est pas recommandé car, alors, il est aisé pour un attaquant d'obtenir la répartition des valeurs prises par l'élément indexé dans la table. En effet, les pointeurs permettent alors d'associer trivialement des lignes dans la table avec une

entrée de la table de hachage. Il est donc fortement préconisé de chiffrer ces pointeurs.

4.3.4 Une variante

Une façon d'améliorer la sécurité dans ce cas est d'autoriser plusieurs chiffrés possibles pour un clair donné. La requête peut ensuite se faire de deux façons :

- calculer tous les chiffrés correspondant à la donnée claire, et effectuer ensuite des requêtes pour chacune de ces valeurs,
- utiliser un chiffrement à flot et chiffrer le début du clair et n'effectuer la requête que sur cette partie, d'où des fausses réponses détectées lors de la réception des données par le client.

La première solution peut être implémentée en concaténant le début de la valeur claire chiffrée comme dans la section précédente avec la fin chiffrée de façon sûre. Par exemple, pour une donnée M et un entier k, soit $M_{0\to(|M|-k-1)}$ ses $|M|-k$ premiers bits et $M_{(|M|-k)\to(|M|-1)}$ ses k derniers bits. Le chiffré de M se calcule alors comme cela :

$$C = E_K\left(IV, M_{0\to(|M|-k-1)}\right) \| E_K\left(IV', M_{(|M|-k)\to(|M|-1)}\right),$$

où IV est commun à tout l'index alors que IV' diffère pour chaque donnée. Les chiffrés possibles de M sont alors les éléments de l'ensemble suivant :

$$\left\{E_K\left(IV, M_{0\to(|M|-k-1)}\right) \| x \mid x \in \{0,1\}^k\right\}.$$

4.4 Chiffrements à fonctionnalités particulières

Les deux scénarios correspondant à ce sujet ayant motivé le plus de recherches sont les suivants :

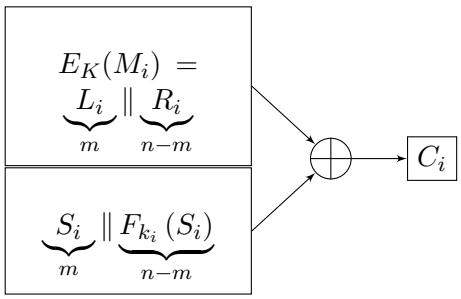
- la recherche par mot clef sur un document chiffré, soit, par exemple, comme expliqué dans [HHIM07], le cas d'un serveur email stockant les emails chiffrés d'utilisateurs qui souhaitent pouvoir effectuer des recherches par mots clefs sur leurs emails,
- l'évaluation de requête sur des bases de données chiffrées, qui est le problème qui nous intéresse tout particulièrement.

Bien que le premier sujet ne nous intéresse pas directement, il peut être source d'idée pour notre problématique.

4.4.1 Recherche par mot clef sur un document chiffré

[SWP00] propose la première façon de chiffrer un document tout en permettant de chercher un mot au sein du texte chiffré. Ce modèle de chiffrement se situe au niveau applicatif, contrairement à ceux vus aux deux sections précédentes. Ici, le serveur ne doit donc pas avoir besoin de la clef et ne doit pas connaître les mots ainsi cherchés. On suppose ici que les mots M_i composant le document ont la même longueur. Il s'agit alors de calculer un chiffré C_i correspondant à M_i qui autorise une recherche sur les C_i, sans pour autant révéler la distribution des mots. Notons n la longueur de M_i, et m un entier plus petit que n. Nous avons aussi besoin d'un algorithme de chiffrement par blocs E opérant sur des blocs de n bits, paramétré par une clef K, d'un algorithme de chiffrement à flot générant des suites chiffrantes S_i de longueur m pour chaque M_i, d'une fonction pseudo-aléatoire $F : \mathcal{K} \times \mathbb{F}_2^m \to \mathbb{F}_2^{n-m}$, et d'une fonction pseudo-aléatoire $f : \mathcal{K} \times \mathbb{F}_2^m \to \mathcal{K}$, où $\mathcal{K}$ est l'espace des clefs possibles. Le chiffré C_i de M_i est alors calculé comme décrit à la figure 4.14, où $k_i = f_K(L_i)$ et où K est la clef utilisée.

FIGURE 4.14 – Chiffrement selon [SWP00]



Il est alors aisé d'effectuer une recherche sur un document chiffré de cette façon. Pour cela il suffit de fournir au serveur $E_K(M_i)$ ainsi que le k_i correspondant. Le serveur peut alors chercher si $C_i \oplus E_K(M_i)$ est de la forme $s \| F_{k_i}(s)$ pour un s donné. S'il y a N mots à parcourir, cela se fait en $O(N2^m)$. Le serveur ne découvrira ainsi que la répartition des mots cherchés, et rien sur les autres. Par contre, si les mots ont une taille distincte, le remplissage à effectuer aura un coût.

Notons que [CM05] apporte une autre approche à ce problème, un index de mots-clefs. L'idée de cet article est de créer un index sous la forme d'une matrice contenant des 1 ou des 0 en fonction de la présence du mot m_i dans le fichier $\mathcal{F}_j$. Cet index est alors envoyé après application d'un masque au serveur. En donnant certaines informations au serveur, il est ensuite possible d'accéder aux fichiers désirés. Cette solution est indépendante du chiffrement des fichiers sur le serveur, mais nécessite la création d'un index dont la mise

à jour n'est pas aisée. Cette méthode est assez similaire à l'utilisation de filtres de Bloom.

4.4.2 Modèles dédiés aux bases de données

Nous allons commencer par passer en revue plusieurs nouvelles techniques de chiffrement dédiées à la protection des bases de données. L'idée principale ici est de permettre certains types d'opérations sur les chiffrés. Parmi les premières techniques mises en avant dans ce but, on trouve les chiffrements homomorphiques développés dans [RAD78]. Les chiffrements pleinement homomorphiques [Gen09] permettent des opérations booléennes de base sur des chiffrés, et donc tout type d'opération, y compris des comparaisons, mais cela n'est pas actuellement praticable.

Un des premiers types de chiffrement proposés autorisant les comparaison sur les chiffrés est un chiffrement préservant l'ordre, « Order Preserving Encryption » (OPE), proposé initialement dans [AKSX04]. C'est un chiffrement déterministe, il préserve donc l'égalité. Son fonctionnement consiste à chiffrer la base, puis à créer un index sur les tables chiffrées. L'idée est, en répartissant les données à chiffrer par paquets, de les répartir uniformément dans un premier temps, puis selon une distribution choisie par l'utilisateur. La clef consiste alors en les transformations nécessaires pour passer ainsi des valeurs claires aux valeurs chiffrées. La taille de la clef, d'après les auteurs, est de quelques kilo-octets, et le chiffré est plus grand que le clair d'environ quatre octets. Pour chiffrer une table avec ce chiffrement, il faut connaître les données à l'avance pour déterminer les transformations. De plus, les mises à jour ne sont possibles que si le clair conserve la même distribution, dans le cas contraire des chevauchements peuvent apparaître et le déchiffrement n'est plus assuré. Mais la limitation principale de ce chiffrement est sa sécurité. Tout d'abord, comme l'indique son nom, ce chiffrement conserve l'ordre. Il en découle qu'il est vulnérable à une attaque à clairs connus. Ainsi, une attaque où l'on disposerait des données en clair utilisées pour chiffrer initialement la table, ainsi que des chiffrés correspondant permet d'obtenir immédiatement la clef, celle-ci n'étant en fait que les coefficients des deux transformations appliquées aux données. Effectivement, les transformations permettant de passer d'un ensemble de clairs à leurs chiffrés dont la répartition est prédéterminée sont définies par ces deux données. En les connaissant, il est donc possible de calculer ces fonctions, qui forment la clef de ce chiffrement.

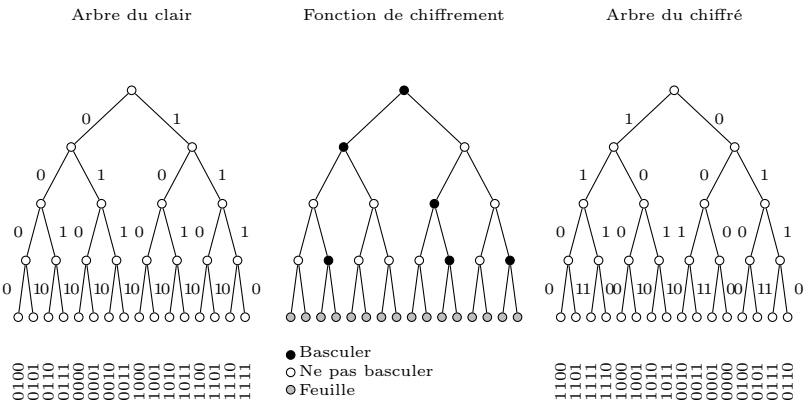
[ÖSC03] propose diverses variantes de [AKSX04], et en particulier des chiffrements qui préservent les différences. De plus, les familles de fonctions de chiffrement préservant l'ordre présentées dans ce papier ont un paramètre permettant d'ajuster le compromis entre la sécurité prétendue de cette fonction et sa vitesse de calcul.

[BCLO09] fait suite à [AKSX04] et propose une étude de sécurité du schéma « Order Preserving Encryption », ainsi qu'un tel chiffrement vérifiant

la meilleure sécurité atteignable. Les auteurs de [BCLO09] montrent dans un premier temps qu'aucun schéma de type OPE ne peut atteindre un niveau de sécurité IND-CPA, même avec une relaxation directe. Ils proposent donc un modèle de sécurité s'appuyant sur la notion de fonction pseudo aléatoire. Finalement, ils présentent un schéma OPE vérifiant cette notion de sécurité, mais qui nécessite l'utilisation de grandes fonctions pseudo aléatoires secrètes qu'il faut stocker pour pouvoir ensuite déchiffrer.

Un autre type de chiffrement autorisant certaines comparaisons sur les chiffrés est le chiffrement préservant les préfixes, « Prefix Preserving Encryption » (PPE), proposé dans [LO05]. Comme le nom de ce chiffrement l'indique, deux messages ayant un préfixe commun de ℓ bits auront des chiffrés qui seront identiques sur leurs ℓ premiers bits, puis différeront. Pour chiffrer, [LO05] utilise des arbres binaires complets. Plus précisément, si on veut chiffrer des messages de $\mathbb{F}_2^n$, la fonction de chiffrement est un arbre binaire complet de hauteur n dont les nœuds contiennent l'information « basculer » ou « ne pas basculer » et pour chiffrer un message il suffit de changer la valeur des bits en parcourant l'arbre quand on passe sur un nœud demandant de « basculer ». Un exemple de fonctionnement est illustré par la figure 4.15, issue de [LO05]. Il en découle que la fonction de chiffrement, qui fait ici office de clef, est composé de $2^n - 1$ variables binaires. Cela est évidement énorme, même si l'on cherche à chiffrer des petits messages, ce qui est fréquent dans le contexte des bases de données. Par exemple, pour un champ texte de deux caractères (ce qui est adapté pour les codes des pays) la taille de la clef sera $2^{16} - 1$, soit 65535 bits. Par contre, la sécurité d'un tel schéma n'est pas meilleure que celle de ceux vus précédemment. En particulier, comme le reconnaissent les auteurs de [LO05], ce schéma préserve les statistiques et n'est donc pas adapté pour protéger des données sensibles.

FIGURE 4.15 – Un exemple de chiffrement préservant les préfixes



4.5 Dégradation d'algorithmes existants par ajout d'information

Comme expliqué dans [HHIM07], cette méthode consiste à ajouter de l'information supplémentaire au chiffré, mais cela peut donc révéler des informations au serveur. L'idée centrale ici est donc de limiter la quantité d'information dévoilée, tout en lui maintenant son utilité. Les techniques classiques pour contrôler la diffusion de l'information sont :

- perturbation : ajout d'une valeur aléatoire à la valeur réelle,
- généralisation : remplacement de la vraie valeur par une valeur plus générale, comme par exemple l'intervalle dans lequel elle se situe,
- échange : échange de valeurs d'un attribut donné pour deux identifiants.

La technique la plus utilisée est la généralisation. Nous allons en voir quelques exemples.

[HILM02] suggère d'ajouter à une table des colonnes pour chaque élément de la table sur lequel on souhaite faire efficacement des recherches. Le contenu d'une ligne chiffrée devient alors le chiffré du n-uplet, suivi de valeurs faisant office d'index. Cette valeur d'index est calculée par une fonction appliquée à la donnée en clair qui peut être de deux types, soit aléatoire, soit préservant l'ordre. Dans le premier cas, seules les recherches par égalité peuvent être effectuées sans déchiffrement. Mais dans ce cas l'utilisation d'un système de chiffrement non déterministe ne sert plus à rien. Dans le second, les recherches par intervalle ne nécessitent un déchiffrement que pour les lignes sélectionnées grâce à l'index. Cet exemple tombe clairement dans la catégorie de généralisation évoquée précédemment, et la fonction utilisée est une projection sur une liste d'intervalles, ordonnés ou non. Du point de vue de la sécurité, un tel chiffrement ne peut pas être acceptable pour chiffrer des données sensibles, en particulier car des valeurs proches seront clairement identifiables en regardant la table chiffrée. Cette proposition a été ensuite étayée par d'autres publications, comme par exemple [DdVJ+03].

[ZLN08] propose un index particulier qui est en fait constitué d'informations déguisées. L'idée est ici de remplacer une ligne d'une table données en clair `(id, field1, field2)` en une ligne chiffrée `(id, field1, encrypted_field2, index)` où nous ne souhaitons protéger que le deuxième champ et où l'index est en fait un entier permettant d'accélérer les recherches sur ce deuxième champ. Ce numéro appelé `index` ici est calculé de la façon suivante. Si $M = a_1 a_2 \dots a_m$ est une chaîne de caractères d'un élément du deuxième champ, nous divisons M en groupes de taille k. Par exemple, si $k = 2$, « France » sera divisé en « Fr », « ra », « an », « nc », et « ce ». Pour

chaque élément ainsi obtenu, nous calculons une valeur n_i correspondante :

$$n_i = \text{hash}(a_i \dots a_{i+k-1} \| K) \mod 10,$$

où K est une clef. Cela donne naturellement un chiffre et en concaténant les n_i dans l'ordre du mot nous obtenons un nombre N. Or pour deux mots égaux, les valeurs N correspondant vont aussi être égales. Cette valeur va donc être transformée en une valeur $I = F(N, \ell)$, où ℓ est le numéro de ligne où l'on stocke I et F une permutation. Cela rend ce chiffrement non déterministe. Pour effectuer une requête sur un tel index, il faut dans un premier temps appliquer F^{-1} à I pour obtenir N. Il suffit alors de comparer N à la valeur recherchée correspondante N'. Dans notre exemple, nous avons choisi $k = 2$, donc si N et N' ont un chiffre en commun, cela peut indiquer qu'il peuvent avoir une suite de deux lettres en commun. S'ils ont deux chiffres consécutifs en commun, on peut espérer alors trois lettres consécutives en commun, et ainsi de suite. Il est donc possible de rechercher des mots commençant et/ou finissant par une chaîne de caractères donnée, bien qu'il puisse y avoir des faux positifs. Nous voyons bien ici que l'index est en fait une généralisation du mot et que de l'information est révélée au serveur via ce biais, d'autant plus que F^{-1} est publique et non paramétrée par une clef. Ainsi, le calcul du haché gagnerait à être modifié en un MAC, comme vu au paragraphe 1.1.4.

4.6 Une solution pour les recherches d'égalité

Comme nous l'avons déjà vu à la section 1.2.3, le but d'un index est de faciliter la recherche d'une donnée dans une base de données. Plus précisément, c'est un ensemble structuré de données contenant :

$$\{(F(m_p), p) \mid p \in \mathcal{P}\}, \tag{4.1}$$

où $\{m_p\}_{p\in\mathcal{P}}$ sont les données de la base, c'est-à-dire les données que l'on souhaite indexer, et p le pointeur vers cette donnée $\{m_p\}$ dans la base. $\mathcal{P}$ désigne l'ensemble de tous les pointeurs vers les n-uplets dans la base de données que nous considérons. En utilisant un index $\mathcal{I}$, à partir d'une valeur donnée m, il devrait être plus rapide de récupérer tous les pointeurs p tels que $m_p = m$, et ainsi les n-uplets correspondant dans la base, que d'utiliser la base seule où nous devrions comparer m à chaque donnée de son champ, ce qui coûte un nombre de comparaisons égal au nombre de n-uplets dans la base. Finalement, comme nous l'avons déjà évoqué à la section 1.2.3, la taille de l'index ne se limite pas toujours à $\{(F(m_p), p) \mid p \in \mathcal{P}\}$, car si cet ensemble est toujours inclus dans l'index, celui-ci peut contenir des données additionnelles permettant d'accélérer les recherches.

Nous avons aussi déjà vu que les types d'index les plus usuels sont les arbres B et les tables de hachage, présentés à la section 1.2.3. Ces deux types

d'index sont non seulement structurés mais aussi ordonnés. En effet, un chemin dans un arbre B suit les résultats de comparaisons, et une table de hachage est ordonnée selon ses clefs. En particulier, c'est cet ordonnancement qui améliore les temps de recherche. Dans le cas des arbres B, les feuilles contiennent les valeurs m_p de la table, et une liste associée contenant les pointeurs correspondant. Les nœuds intérieurs de l'arbre permettent d'augmenter la vitesse de recherche. En effet, en général, effectuer une recherche en utilisant un arbre ne requiert qu'un nombre logarithmique de comparaisons. Dans le cas d'une table de hachage, l'index est une table contenant $h(m_p)$ et, pour chaque valeur possible de $h(m_p)$, la liste de pointeurs associés. Ce cas ne présente pas de stockage de données additionnelles, mais nécessite le calcul de $h(m)$.

Dans l'étude qui suit nous nous limiterons au cas où il est nécessaire que les n-uplets correspondant à différentes valeurs m soient systématiquement distingués par l'index. Cela a néanmoins l'inconvénient d'augmenter la taille de stockage nécessaire pour représenter l'index. Autrement dit, nous ne prendrons en considération que les index où la fonction F définie dans la relation (4.1) est injective, c'est-à-dire :

$$\forall m \neq m',\ F(m) \neq F\left(m'\right).$$

Cela implique en particulier que les index basés simplement sur un filtre de Bloom, notion définie à la section 1.2.3, sont exclus de cette discussion.

4.6.1 Deux concepts différents

Une fois les index dédiés aux requêtes par intervalle mis de côté, il ne reste que trois principales catégories d'index :

- index primaire, où tous les m_p de la base de données sont distincts,
- index secondaire, où plusieurs occurrences sont autorisées,
- index multi-attributs.

Nous avons déjà vu à plusieurs reprises dans ce manuscrit l'importance de protéger un index lorsque l'on chiffre une base de donnée. Nous avons également vu à la section 3.3.1 qu'il y a essentiellement deux façons de construire un index sûr :

- index chiffré : l'index est ici construit sur les données en clair, c'est-à-dire qu'il garde la structure ordonnée de l'index en clair et simplement consiste à en chiffrer les données ; c'est le cas que nous avons détaillé à la section 4.3 ;
- index sécurisé : les données sont transformées, par exemple chiffrées, hachées…, et un index est ensuite construit sur ces valeurs modifiées et forme ainsi une nouvelle structure ordonnée.

Il est cependant possible d'unifier les notations pour ces cas. En effet, dans les deux cas, l'index contient des valeurs $\{(F(m_p), F'(p)) \,|\, p \in \mathcal{P}\}$, où nous expliciterons les conditions sur F et F' plus loin. Les différences entre un tel index et ceux décrits précédemment avec la relation (4.1) sont d'une part qu'ici le but de F n'est plus uniquement d'améliorer les requêtes mais aussi de protéger la confidentialité des données, et d'autre part l'addition d'une nouvelle fonction F'. Un index chiffré, comme défini ci-dessus, consiste essentiellement à construire un index sur des données en clair, puis à chiffrer son contenu. Ainsi, ce n'est pas strictement un index comme défini à la section 1.2.3 et par la relation (4.1). Au contraire, un index sécurisé se construit en commençant par transformer les valeurs, par exemple en les chiffrant, puis en construisant l'index à partir de ces valeurs transformées. C'est donc bien un index au sens de la section 1.2.3 et de la relation (4.1), car les paires $\{(F(m_p), F'(p))\}$ sont ordonnées selon les valeurs de $F(m_p)$ dans l'index. De plus, il est intéressant de remarquer que c'est en fait une généralisation de la relation (4.1) car celle-ci suit cette définition en restreignant F' à l'identité et sans obliger F à vérifier des conditions de sécurité.

Même si ces deux approches semblent similaires à première vue, leurs implications divergent grandement. Comme nous l'avons longuement vu précédemment, la première notion a été grandement étudiée dans la littérature et de nombreux algorithmes de chiffrement ont été inventés dans ce but, comme par exemple les algorithmes de chiffrement préservant l'ordre ou les préfixes, dont l'algorithme FCE dont nous présenterons une cryptanalyse au chapitre suivant. Son principal avantage, comparée à la seconde approche, est qu'elle permet des requêtes par intervalle, ce qui est indispensable dans de nombreuses applications. Mais elle présente aussi de nombreux inconvénients. Ainsi, les comparaisons pour chercher parmi les données nécessitent le déchiffrement des données contenues dans les nœuds ou l'utilisation d'algorithmes de chiffrement spéciaux qui évitent ces déchiffrements mais sont, en contrepartie, faibles. Cette première notion impose aussi à F' d'être une fonction de chiffrement, sinon, à partir de l'index, l'ordre des données dans la table serait révélé. Nous allons voir que la seconde approche nous offre la latitude pour éviter ces inconvénients.

Pour anticiper les sections suivantes, nous pouvons déjà lister les avantages de l'utilisation d'index sécurisés par rapport aux index chiffrés :

- coût de recherche plus faible, car aucun déchiffrement n'est requis,
- coût de mise à jour de l'index plus faible, en particulier pour les insertions et les suppressions,
- nul besoin de chiffrer les pointeurs vers les n-uplets, sans pour autant laisser fuir de l'information,
- très faible coût d'intégration dans des moteurs de bases de données existants.

Il faut malgré tout rappeler son principal inconvénient, c'est-à-dire son incapacité à gérer des requêtes par intervalle.

Jusqu'à la fin de cette section, nous nous concentrerons sur la seconde approche, les index sécurisés. La première étape pour cela est de présenter les exigences pour définir la fonction F :

- elle doit être publique d'après les principes de Kerckhoffs [Ker83], mais elle peut être paramétrée par une clef,
- elle doit être à sens unique, c'est-à-dire que la connaissance de $F(m)$ ne doit pas donner d'indication sur m,
- elle doit résister aux attaques par dictionnaire, par exemple dans le cas où les valeurs de m possibles se limitent aux 36 680 communes françaises[3], il devrait être impossible pour un attaquant de précalculer tous les $F(m)$,
- son calcul doit être rapide côté client, c'est-à-dire en logiciel,
- elle doit être non-déterministe, et en particulier, en regardant l'index, un attaquant ne devrait pas être capable de connaître la distribution des valeurs en clair,
- les faux positifs ne sont pas les bienvenus, et sont en fait interdits comme nous l'avons vu précédemment,
- les faux négatifs sont interdits.

Le troisième argument a pour effet d'imposer à F d'être paramétrée par une clef secrète.

4.6.2 Index primaires

Si l'on restreint l'étude aux index primaires, la condition précédemment vue imposant à F d'être non-déterministe n'est plus pertinente, étant donné que des égalités parmi les valeurs indexées ne peuvent alors plus se produire. Il en découle qu'utiliser n'importe quel algorithme de chiffrement symétrique, comme par exemple AES-CBC est suffisant. Mais il faut noter que la fonction F n'est pas obligatoirement bijective, dans la mesure où la transformation inverse n'intervient jamais. Il en découle que le coût pour les recherches et les insertions est d'un unique chiffrement.

Il convient alors de s'interroger maintenant sur les conditions portant sur F', et en particulier sur ce que révélerait F' si elle était l'identité. Pour rappel, dans le cas d'un index chiffré, si F' était l'identité, cela faisait apparaître

[3] Nombre de communes en France au premier janvier 2011, selon http://www.insee.fr/fr/methodes/nomenclatures/cog/telechargement.asp

l'ordre des n-uplets dans la table. Dans le cas des index sécurisés, cela révélerait aussi l'ordre des n-uplets, mais cette fois dans l'ordre de leurs valeurs chiffrées et non de leurs valeurs claires, comme dans le cas précédent. Il en résulte que l'information ainsi offerte à un attaquant ne lui sera d'aucune utilité et que donc rien ne s'oppose ici à ce que F' soit l'identité.

Nous allons maintenant présenter les algorithmes dont on a besoin pour utiliser une base de donnée ainsi protégée. Nous supposons que nous disposons des algorithmes classiques pour construire un index de notre choix, arbre B ou table de hachage, y insérer des éléments, en supprimer et y effectuer une recherche. Tout cela sur une table et un index en clair. Voici les prototypes de ces fonctions dans le pseudo-code que nous utiliserons par la suite :

- `index build_index(set {(m_p, p)})`
- `index insert_in_index(couple (m_p, p), index I)`
- `index delete_from_index(couple (m_p, p), index I)`
- `pointer search_index(value m, index I)`

Nous présentons maintenant comment adapter ces fonctions au cas des index sécurisés. Pour la clarté de la lecture, nous supposerons que les portées de la fonction F et de la clef K sont globales. Le préfixe `us` utilisé ici signifie « unique secure (index) ». C'est le préfixe que nous utiliserons pour les index primaires sécurisés.

```
us_index us_build_index(set {(m_p, p)})
{
    return build_index(set {(F_k(m_p), p)});
}

us_index us_insert_in_index(couple (m_p, p), us_index I)
{
    return insert_in_index(couple (F_k(m_p), p), s_index I);
}

us_index us_delete_from_index(couple (m_p, p), us_index I)
{
    return delete_from_index(couple (F_k(m_p), p), us_index I);
}

pointer us_search_index(value m, us_index I)
{
    return search_index(value F_k(m_p), us_index I);
}
```

Le surcoût par rapport aux fonctions utilisées s'il n'y avait pas de chiffrement n'est donc que d'un unique chiffrement par appel de fonction à chaque fois.

4.6.3 Index secondaires

Dans ce cas, il n'est plus possible de laisser de côté la condition qui impose à F d'être non-déterministe. En effet, si F était déterministe, cela aurait pour résultat de rendre visible à un attaquant le nombre d'éléments correspondant à la feuille d'un arbre B ou à une case d'une table de hachage. Il nous faut donc une structure pour stocker ces pointeurs de telle sorte que ces regroupements soient cachés. Cela ne peut se faire en utilisant un simple tableau ou une liste stockée dans la structure contenant la feuille. Il faut donc que la feuille, ou la case de la table de hachage, ne contienne qu'un pointeur vers la structure désirée. Pour pouvoir poursuivre cette étude, nous avons besoin de noter n le nombre de valeurs distinctes dans l'index, soit $n = \#\{F_k(m_p) \,|\, p \in \mathcal{P}\}$. Nous allons donc énumérer les propriétés que nous souhaitons pour cette structure :

- sa taille ne doit pas être visible pour un attaquant,
- les insertions et suppressions dans l'index doivent être peu chères, au pire en $O(n)$ requêtes à l'algorithme de chiffrement et/ou de déchiffrement,
- récupérer tous les pointeurs correspondant à une valeur donnée doit être rapide, c'est-à-dire pas plus cher que $O(n)$ déchiffrements.

Le premier point interdit l'utilisation d'un tableau, sa taille étant visible sur le disque. Mais une liste chaînée chiffrée contenant comme valeurs les pointeurs vers les n-uplets de la table satisfait toutes ces conditions. En effet, sa taille n'est pas visible, l'insertion est moins chère qu'avec un tableau en ne demandant que $O(1)$ chiffrement au lieu de $O(n)$, et finalement les suppressions et la récupération de pointeurs se font au pire en $O(n)$ chiffrements et/ou de déchiffrements.

Le surcoût total en termes de chiffrement lorsque l'on cherche tous les n-uplets correspondant à une valeur dans un index secondaire est le suivant :

- un chiffrement d'une valeur, qui correspond au chiffrement déjà nécessaire dans le cas de l'index primaire,
- $2 \times n$ déchiffrements : un pour chaque pointeur vers un n-uplet et un pour chaque pointeur vers l'élément suivant dans la liste chaînée.

Pour mettre l'index à jour, les coûts sont les suivants :

- un chiffrement de valeur, un chiffrement de pointeur,

- pour les insertions, la construction d'un élément de la liste, ce qui inclut le chiffrement d'un autre pointeur,
- pour les suppressions, une recherche dans la liste, soit au pire $2 \times n$ déchiffrements, la suppression d'un élément de la liste, qui a le même coût que dans le cas sans chiffrement.

Comme nous l'avions fait dans le cas des index primaires, nous allons maintenant décrire les algorithmes adéquats. Pour rappel, nous avions utilisé le préfixe `us` (« unique secure (index) ») pour les index primaires sécurisés. Nous utiliserons ici le préfixe `s` (« (secondary) secure (index) ») pour les index secondaires. Ainsi, un index sécurisé `I` du type `s_index` contient un index primaire sécurisé `I'` de type `us_index` qui pointe vers n listes chaînées `L` de type `s_chained_list`.

```
s_index s_insert_in_index(couple (m_p, p), s_index I)
{
    p_c = us_search_index(value m, us_index I.I');
    if (p_c == NULL) {
        p_c = new s_chained_list(p, NULL);
        I.I' = us_insert_in_index(couple (m_p, p_c),
                                  us_index I.I');
    } else {
        p_aux = p_c;
        p_c = new s_chained_list(p, p_c);
        I.I' = us_update_in_index(couple (m_p, p_c),
                                  us_index I.I');
    }
    return I;
}

s_index s_delete_from_index(couple (m_p, p), s_index I)
{
    p_c = us_search_index(value m, us_index I.I');
    if (p_c.next == NULL) {
        I.I' = us_delete_from_index(couple (m_p, p),
                                    us_index I.I');
    } else {
        s_remove_from_chained_list(p_c, p);
    }
    return I;
}

s_chained_list s_search_index(value m, s_index I)
{
```

```
    return us_search_index(m_p, s_index I.I');
}
```

4.6.4 Index secondaires séquentiels

Dans certains cas [ABG, Guo11], il est plus approprié, voire nécessaire, d'utiliser des bases de données séquentielles dans lesquelles aucune mise à jour n'est faite. Le choix des listes chaînées n'est alors plus particulièrement pertinent, la nécessité de pouvoir effectuer des mises à jour rapides de l'index n'étant plus d'actualité, comme c'était le cas précédemment. Il devient alors intéressant d'envisager l'utilisation d'un tableau qui cacherait la distribution des données. Une telle solution est un tableau à deux colonnes. La première contient les pointeurs vers le n-uplet correspondant dans la table, et la seconde contient un pointeur vers l'élément précédent de la table avec la même valeur dans le tableau. Nous avons naturellement besoin de chiffrer la seconde colonne du tableau pour cacher la distribution des valeurs. Ainsi l'index est formé d'un tableau de la forme suivante :

TABLE 4.1 – Index secondaire séquentiel

`pointer`	`previous`
p_0	E_K (`NULL`)
$\vdots$	$\vdots$
p_i	E_K (previous (m_{p_i}))
$\vdots$	$\vdots$

où :

$$\text{previous}\,(m_{p_i}) = \max_j \left\{ j < i \text{ et } m_{p_j} == m_{p_i} \right\}.$$

Avec une telle solution, on remarque que si E_K est déterministe, cela révèle le nombre de valeurs distinctes dans la table. En effet, soit une donnée m_p de la table, et $i = \min_j \left\{ m_{p_j} == m_p \right\}$ alors la colonne `previous` correspondant à p_i est la seule et unique à contenir E_K (`NULL`). Il en découle qu'il y a exactement autant de valeurs E_K (`NULL`) dans la colonne `previous` que de valeurs distinctes dans la table, et en particulier que si E est déterministe, ce nombre est ainsi révélé par cette colonne, dont la première valeur est nécessairement de cette forme.

On peut imaginer deux variantes de ce système conçues pour diminuer la taille de l'index. La seconde colonne du tableau peut être remplacée par :

- le numéro de la ligne dans le tableau de l'élément précédent de même valeur,

- le nombre de lignes séparant de l'élément précédent de même valeur.

Dans la deuxième variante, si nous utilisons un chiffrement déterministe, la distribution des écarts sera évidente, et, en partant du début du tableau où les écarts possibles sont fortement limités, il est possible de retrouver tous les écarts et ainsi la distribution des valeurs. Il en découle que le chiffrement utilisé ne peut pas être déterministe et doit donc soit utiliser un IV distinct à chaque ligne, soit être un chiffrement adaptable dont le tweak diffère d'une ligne à l'autre. Par exemple, l'IV ou le tweak pourraient être déduit de l'adresse ou de l'indice de la ligne. De plus, cela interdit pour des raisons évidentes aussi d'utiliser le même chiffrement avec la même clef que pour chiffrer la table. Dans la première variante, il faut faire attention à ne pas chiffrer les pointeurs vers les n-uplets dans la table avec le même chiffrement déterministe que dans l'index primaire car cela pourrait résulter en des collisions dont il serait facile de déduire certaines valeurs des pointeurs. Finalement, la première possibilité nécessite plus de place, mais ne présente aucun des inconvénients précédents.

Nous avons donc proposé des solutions pour le chiffrement des bases de données et la sécurisation des index grâce à la notion d'index sécurisé définie à la section 4.6.1. L'idée de base est de créer l'index s'il est primaire, ou sa partie primaire sinon, sur les données chiffrées de façon déterministe. Ensuite, nous avons étendu cette solution aux index secondaires en proposant une structure qui permet de cacher les répartitions, une liste chaînée chiffrée. Finalement, nous avons adapté notre proposition au cas des bases de données séquentielles qui permettent d'employer un simple tableau comme index, à condition d'utiliser un chiffrement qui n'est pas déterministe pour chiffrer la colonne permettant de passer en revue tous les éléments ayant la même valeur dans la base.

Chapitre 5

Un mauvais exemple de chiffrement dédié : FCE

Nous allons maintenant étudier un chiffrement dédié aux bases de données, et en particulier nous allons exhiber ses multiples faiblesses.

5.1 Description de FCE

Le chiffrement FCE [GZ07] (Fast Comparison Encryption) a été construit dans le contexte du système de gestion de base de données orienté colonnes « open-source » C-Store[1]. C-Store se distingue principalement des systèmes de gestion de base de données que nous avons vus précédemment en organisant les données par colonnes au lieu de les stocker par lignes. FCE est un algorithme de chiffrement au niveau base de données. Son utilisation est détaillée dans [GZ07] ; le principe est de chiffrer la colonne à protéger avec FCE, et de chiffrer aussi l'index avec FCE. Dans le cas où l'index est un arbre, les pointeurs des nœuds internes ne sont pas chiffrés. Par contre, les pointeurs des feuilles, doivent être chiffrés pour ne pas révéler l'ordre des valeurs de la colonne, ce qui réduirait la sécurité de FCE à celle d'un OPE. FCE supporte aussi les index par table de hachage, mais FCE étant conçu pour supporter les recherches par intervalle, les index basés sur les arbres sont plus appropriés.

La motivation des auteurs de FCE est de définir un chiffrement sûr qui autorise néanmoins des recherches efficaces dans l'index. Pour ce faire, ils définissent le concept de « Early Stopping » qui consiste, pour un algorithme de chiffrement, à ne pas déchiffrer intégralement deux messages pour être capable de les comparer. Plus précisément, avec FCE, il est possible de faire des comparaisons après seulement des déchiffrement partiels. Le concept est le suivant. La comparaison de deux messages chiffrés commence par l'octet

[1] http://db.csail.mit.edu/projects/cstore/

de poids fort, avance octet par octet et s'arrête au premier octet où les deux chiffrés diffèrent. À ce moment, pour déterminer le plus grand du plus petit, il suffit de déchiffrer ces deux octets qui diffèrent. Cette fonctionnalité est naturellement offerte par tous les chiffrements à flots additifs, ainsi que par certains modes opérant sur des chiffrements par blocs comme le mode CBC. D'ailleurs, FCE n'est en fait qu'un chiffrement à flot synchrone et additif dont le seul IV est le numéro de la page.

Avant de décrire l'algorithme de chiffrement, nous allons commencer par définir les notations que nous allons utiliser. Le texte clair divisé en pages de p octets, est noté $\{m_i\}_{0\leq i<p}$. Le texte chiffré correspondant est $\{c_i\}_{0\leq i<p}$. Il y a une unique clef maître K, de longueur k bits, pour toute la base de données. Comme nous l'avons déjà évoqué, cette clef est connue de l'administrateur de la base de données. À chaque page de texte clair correspond un polynôme de permutation $P(x) = ax^3 + bx^2 + cx + d \mod p$, dérivé du numéro de la page et de la clef maître K. La suite chiffrante est ensuite calculée à partir de ce polynôme et de la clef maître K.

Il est intéressant de considérer les paramètres utilisés dans le cas d'une utilisation de FCE avec C-Store, comme préconisé dans [GZ07]. Ce sont les suivants :

- taille de la clef K : $k = 2^{15}$ bits (32 kilo-bits),
- taille de la page : $p = 2^{16}$ octets (64 kilo-octets),
- taille des coefficients a, b, c, d : 64 bits par page de 64 kilo-octets.

Nous allons maintenant décrire le fonctionnement de FCE, d'abord la version publiée dans [GZ07], puis une variante suggérée ultérieurement par un des auteurs de FCE, T. Ge [Ge10].

5.1.1 Première version

Le chiffrement est effectué page par page. Son initialisation requiert un algorithme de chiffrement E pour générer le polynôme de permutation P à partir de la clef K et du numéro de la page j. Le choix de E n'est pas détaillé dans [GZ07], mais n'importe quel chiffrement par blocs, comme par exemple AES peut être utilisé. Avant de présenter l'algorithme 5.1, nous devons d'abord définir une notation.

Définition 9. Étant donné un entier d_i, nous notons $K_{\{d_i \to d_i+7\}}$ l'octet de clef commençant au $d_i{}^{\text{e}}$ bit :

$$K_0, K_1, \ldots, \underbrace{K_{d_i}, \ldots, K_{d_i+7}}_{v_i = K_{\{d_i \to d_i+7\}}}, \ldots, K_{k-1}$$

La clef maître K est considérée comme une chaîne de bits circulaire, c'est-à-dire que les numéros des bits sont définis modulo sa longueur k.

Algorithme 5.1 Fast Comparison Encryption (FCE)

Entrées :

- Une page de texte clair (p octets $\{m_i\}_{i\in\{0,\dots,p-1\}}$).
- Le numéro de la page j.
- La clef maître K de longueur k bits.

Sorties :

- La page de chiffré correspondante (p octets $\{c_i\}_{i\in\{0,\dots,p-1\}}$).

// Chargement de l'IV :
// génération du polynôme de permutation pour la page j
$(a,b,c,d) \leftarrow E(j,K)$ avec $a,b,c,d \in [0,p-1]$
$P(x) = ax^3 + bx^2 + cx + d \mod p$

// Chiffrement de cette page
pour tout i de 0 à $p-1$ **faire**
  $d_i = P(i) \mod k$
  $c_i = m_i \oplus K_{\{d_i \to d_i+7\}}$
fin pour

5.1.2 Variante

Suite à notre cryptanalyse, les auteurs de FCE ont aussi suggéré une variante de cet algorithme de chiffrement [Ge10], dont un des effets est de limiter les valeurs possibles du nombre d'octets par page, p, aux puissances de nombres premiers ; pour des raisons évidentes nous nous concentrerons sur les valeurs $p = 2^n$. Soit Q un polynôme primitif de $\mathbb{F}_2$ et ω une racine de Q. Alors il existe une bijection naturelle ϕ entre $[0, 2^n - 1]$ et $GF(2^n)$: à l'élément $x = \sum_{i=0}^{n-1} x_i 2^i$ de $[0, 2^n - 1]$ correspond l'élément $x = \sum_{i=0}^{n-1} x_i \omega^i$ de $GF(2^n)$, où $\forall i \in [0, n-1],\ x_i \in \{0, 1\}$:

$$\begin{array}{rccc} \phi : & [0, 2^n - 1] & \to & GF(2^n) \\ & x = \displaystyle\sum_{i=0}^{n-1} x_i 2^i & \mapsto & x = \displaystyle\sum_{i=0}^{n-1} x_i \omega^i. \end{array}$$

De plus, pour tous les entiers i et j, nous avons :

$$\phi(i \oplus j) = \phi(i) + \phi(j), \tag{5.1}$$

où $\oplus$ représente le ou exclusif bit à bit quand les entiers sont identifiés à des mots de n bits, et $+$ est l'addition dans le corps fini $GF(2^n)$. P est alors défini comme un polynôme de permutation de degré 3 sur $GF(2^n)$ et le modulo k est pris sur la valeur de $P(x)$ projeté sur $[0, 2^n - 1]$, c'est-à-dire sur la valeur de $\phi^{-1}(P(x))$.

Plus précisément, la procédure de chiffrement de l'algorithme 5.1 devient maintenant :

Algorithme 5.2 Chiffrement d'une page avec la variante de FCE

// Chiffrement de la page
pour tout i de 0 à $p-1$ **faire**
  $d_i = \phi^{-1}(P(\phi(i))) \mod k$
  $c_i = m_i \oplus K_{\{d_i \to d_i+7\}}$
fin pour

5.1.3 Sécurité prétendue de FCE

En fait, FCE est une version pratique, mais affaiblie de r-FCE. La différence entre ces deux algorithmes est, qu'au lieu d'utiliser un polynôme de permutation P dérivé de la clef K et du numéro de la page à chiffrer, r-FCE utilise une permutation aléatoire de $[0, \ldots, p-1]$ dans le cas de la première version de FCE ou une permutation de $GF(2^n)$ dans le cas de sa variante. Grâce à cette propriété, l'algorithme idéal r-FCE est prouvé INFO-CPA-DB, un nouveau modèle de sécurité contre les attaques à clairs choisis, basé sur la notion d'entropie, et défini dans [GZ07] par Ge et Zdonik pour vanter les mérites de FCE par rapport aux chiffrements de type OPE. Les auteurs précisent néanmoins que la relation entre cette nouvelle notion de sécurité et les ressources nécessaires, en temps et en espace, requises pour casser les schémas vérifiant ce niveau de sécurité est encore inconnue. Il est important de noter que les attaques que nous présenterons ci-dessous ne semble pas affecter la sécurité de r-FCE, étant donnée qu'elles exploitent principalement le fait que FCE utilise des polynômes de permutation de degré faible au lieu d'utiliser des permutations idéales.

Malheureusement, r-FCE n'est pas utilisable en pratique, comme le font justement remarquer ses auteurs, car stocker une telle permutation nécessite de stocker au moins $\log_2(p!)$ bits aléatoires. Par exemple, en utilisant, comme dans l'exemple de C-Store $p = 64$ kilo-octets, $\log_2(p!) \sim 954037$ ce qui représente clairement un trop gros surcoût. C'est pourquoi les auteurs de [GZ07] proposent d'utiliser des polynômes de permutation de degré 3 à la place de ces permutations idéales.

Selon les auteurs de FCE, la notion de sécurité INFO-CPA-DB permet d'assurer que non seulement r-FCE, mais aussi FCE est aussi sûr que le chiffrement par bloc E utilisé pour générer le polynôme de permutation P. Les attaques exposées ci-après vont infirmer cette assertion.

5.2 Attaques

5.2.1 Remarques préliminaires sur FCE

Gestion des mises à jour

Les auteurs de FCE précisent dans [GZ07] que C-Store étant optimisée en lecture, ils ne se soucient pas des problèmes de mise à jour dans FCE. Plus précisément, vu que ce type de modification est rare, ils suggèrent de les faire par groupe et de générer de nouveaux coefficients a, b, c et d pour chaque mise à jour. Mais, ils ne mentionnent pas comment effectuer cela. Il faudrait alors rajouter un paramètre à passer au chiffrement par blocs E, un IV défini par exemple à parti d'un numéro de mise à jour, d'un timestamp... qui doit être stocké avec la page concernée.

Par contre, si on souhaite utiliser FCE sans cette légère modification, si un octet est mis à jour d'une valeur m_i à une valeur m'_i, comme nous l'avons mentionné à la section 1.1.3, un éventuel attaquant qui aurait accès à la base de donnée chiffrée peut obtenir $c_i \oplus c'_i$ qui est donc égal à $m_i \oplus m'_i$, ce qui est une faiblesse de conception de cet algorithme.

Génération des polynômes de permutation P

S'il est bien précisé dans [GZ07] qu'il est nécessaire que les polynômes P soient des permutations, leur façon de les générer ne semble pas satisfaisante. Selon Ge et Zdonik, il suffit de tirer au hasard quatre coefficients $a, b, c, d \in [0, p-1]$ pour que le polynôme obtenu $P(x) = ax^3 + bx^2 + cx + d$ soit calculatoirement indistinguable d'une permutation aléatoire. Or, il est aisé de déterminer si un tel polynôme est une permutation ou non en un temps de calcul assez faible. En effet, son évaluation sur $\sqrt{p}$ valeurs possibles d'entrée est suffisante.

Malgré cela, dans le contexte de la première version de FCE, il est aisé de s'assurer que P est bien un polynôme de permutation. En effet, un polynôme $P(x) = ax^3 + bx^2 + cx + d \mod 2^n$ est une permutation si et seulement si a et b sont pairs et si c est impair [Riv99]. Par contre, dans le cas de la variante de FCE où les calculs se font dans $GF(2^n)$, la classification des polynômes de permutation dans un corps fini est un problème de recherche ouvert. Cette variante du système semble donc difficile à mettre en œuvre en pratique.

Taille de la clef et attaques naïves

Pour pouvoir comparer FCE à d'autres algorithmes de chiffrement symétriques, il faut considérer une implémentation pratique de FCE. Nous rappelons donc la taille proposée pour la clef dans le cas d'une utilisation pour C-Store : $k = 2^{15}$ bits (32 kilo-bits). Cela est énorme comparé aux tailles de clefs généralement utilisées pour du chiffrement symétrique (par exemple

128 ou 256 bits pour AES). En cryptographie symétrique, la sécurité attendue pour un chiffrement est la taille de sa clef, autrement dit, la recherche exhaustive de la clef devrait être la meilleure attaque. Dans le cas de FCE, les auteurs prétendent seulement que FCE est aussi sûr que l'algorithme E utilisé.

Nous considérerons dans la suite que nous avons à notre disposition une page de chiffré et la page correspondante de clair. Autrement dit, nous supposons que nous connaissons, pour une des pages, la suite chiffrante qui est constituée des octets de la clef K ordonnés selon l'ordre défini par le polynôme de permutation P. La première idée simple pour récupérer la clef est de procéder à une recherche exhaustive sur les quadruplets possibles (a, b, c, d). Pour chaque polynôme, nous calculons alors ses valeurs sur l'ensemble $[0, p-1]$ et essayons de reconstruire la clef à partir de la suite chiffrante. Si tous les morceaux correspondent, nous dérivons ensuite le polynôme de cette clef et du numéro de la page. Si nous retombons bien sur le polynôme à partir duquel nous avons reconstruit la clef, c'est que le couple (clef, polynôme) est bien celui que nous cherchions. Le coût de cette attaque est 2^{5p}, soit 2^{80} avec les valeurs proposées dans le cas d'une utilisation avec C-Store, en considérant que les morceaux de la clef ne s'assembleront correctement qu'avec le bon polynôme.

Nous remarquons que cette attaque est meilleure qu'une recherche exhaustive sur la clef de 2^{15} bits et qu'elle est indépendante du chiffrement E utilisé, en particulier, si E est un AES avec une clef de 128 ou 256 bits, FCE est alors nettement moins sûr que le chiffrement E utilisé. Nous remarquons donc que le paramètre de sécurité de FCE n'est ni la taille de sa clef, ni le paramètre de sécurité du chiffrement E utilisé, mais les coefficients d'un des polynômes de permutation, ce qui contredit au passage la sécurité annoncée par les auteurs de FCE.

Néanmoins, cette attaque naïve ne nous permet pas encore de monter une attaque pratique contre FCE. D'où l'intérêt des cryptanalyses suivantes qui, elles, proposent des attaques pratiques contre FCE.

5.2.2 Cryptanalyse de FCE

L'attaque que nous allons présenter maintenant est une attaque à clair choisi, comme définie précédemment. Pour récupérer l'intégralité de la suite chiffrante, nous avons besoin d'une page entière de clair et de la page chiffrée correspondante. À partir de ces deux pages, nous allons récupérer la clef, et ainsi tous les polynômes de permutation des différentes pages de la base de donnée, de façon nettement plus rapide que l'attaque naïve présentée précédemment.

Relation entre les paramètres p et k

Avec l'ensemble de paramètres proposé, la taille de la clef $k = 2^{15}$ en bits est la moitié du nombre d'octets $p = 2^{16}$ de chaque page. Ainsi, le polynôme $P(x) = ax^3 + bx^2 + cx + d \mod 2^{16}$ est en fait seulement utilisé modulo 2^{15}, étant donné que $d_i = P(i) \mod 2^{15}$. Comme $\forall i \in [0, 2^{15} - 1],\ P(i + 2^{15}) = P(i) \mod 2^{15}$, nous aurions pu directement définir P modulo 2^{15} au lieu de modulo 2^{16}.

De plus, cette propriété implique $\forall i \in [0, 2^{16} - 1],\ d_i = d_{i+2^{15}}$ et ainsi seulement une demi page de suite chiffrante, soit 2^{15} octets de texte clair et chiffré, est nécessaire pour récupérer l'intégralité de la suite chiffrante, la seconde moitié étant exactement identique à la première. Cela implique aussi que, si nous avons une page chiffrée entière, nous avons ainsi les relations linéaires sur les bits de la page clair. Plus précisément, $m_{i+2^{15}} \oplus m_i = c_{i+2^{15}} \oplus c_i$.

Bien sûr, même sans cette propriété malvenue, c'est-à-dire si nous considérons une version de FCE avec $p = k$, notre attaque fonctionne, mais nécessite alors une page entière de clair et la page correspondante de chiffré.

Principe de l'attaque

Dans cette partie, nous utiliserons la notation $k = 2^\kappa$ pour la taille de la clef, et ainsi nous avons $\kappa = 15$ pour l'ensemble de paramètres de FCE.

Comme nous l'avons vu précédemment, il est beaucoup plus rapide de faire une recherche exhaustive sur les coefficients du polynôme de permutation que sur la clef. L'idée de l'attaque est de réduire l'ensemble des polynômes sur lequel nous effectuons cette recherche.

La première réduction est effectuée grâce à la remarque suivante. Pour une page donnée, les couples (clef, polynôme)

$$\begin{cases} K' &= K \ggg d', \\ P'(x) &= ax^3 + bx^2 + cx + (d - d') \end{cases}$$

sont équivalents pour tout d', c'est-à-dire qu'étant donnée une page de clair, n'importe lequel de ces couples aboutira à la même page de chiffré. En conséquence, nous rechercherons $\widetilde{K} = K \ggg d$ et $\widetilde{P}(x) = ax^3 + bx^2 + cx$ et nous étudierons d seulement une fois $\widetilde{K}$ et $\widetilde{P}$ trouvés.

Notre problème se pose alors ainsi. Nous voulons trouver (a, b, c) à partir de la connaissance de $\widetilde{K}_{\{\widetilde{P}(i) \to \widetilde{P}(i)+7\}} = K_{\{P(i) \to P(i)+7\}}$ où $\widetilde{P}(i) = ai^3 + bi^2 + ci \mod 2^\kappa$.

Ce problème est facilité par l'information donnée par $\widetilde{K}_{\{\widetilde{P}(i) \to \widetilde{P}(i)+7\}}$ sur les antécédents des éléments successifs par $\widetilde{P}$. Par analyse de la suite chiffrante, nous allons en effet déterminer un ensemble de triplets (α, β, γ) parmi lesquels touts les antécédents de $(1, 2, 3)$ par $\widetilde{P}$ sont présents ; même s'il reste encore des triplets en trop.

Ensuite, pour chaque triplet possible (α, β, γ) de cet ensemble, nous cherchons, dans l'anneau $\mathbb{Z}/2^\kappa\mathbb{Z}$, une solution au système de Vandermonde suivant :

$$\begin{pmatrix} \alpha^3 & \alpha^2 & \alpha \\ \beta^3 & \beta^2 & \beta \\ \gamma^3 & \gamma^2 & \gamma \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} 2^\kappa. \tag{5.2}$$

Nous pouvons même éliminer tous les triplets où α ou γ sont pairs, car il est évident à partir de ce système qu'ils doivent être tous les deux impairs étant donné que c l'est.

Nous devons maintenant résoudre ce système de Vandermonde. Dans ce but, on note $M = \begin{pmatrix} \alpha^3 & \alpha^2 & \alpha \\ \beta^3 & \beta^2 & \beta \\ \gamma^3 & \gamma^2 & \gamma \end{pmatrix}$. Ainsi nous cherchons une solution du système suivant, avec M donné :

$$M \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} 2^\kappa.$$

Une fois multiplié par la transposée de la comatrice de M, ${}^tM^C$, nous avons :

$$\det M \begin{pmatrix} a \\ b \\ c \end{pmatrix} - {}^tM^C \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} 2^\kappa = {}^tM^C \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix},$$

qui peut être simplifié de cette façon :

$$\det M \begin{pmatrix} a \\ b \\ c \end{pmatrix} + 2^\kappa \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \begin{pmatrix} m_a \\ m_b \\ m_b \end{pmatrix}.$$

Ainsi, chaque coefficient $\theta \in \{a, b, c\}$ est une solution d'une équation à résoudre de la forme $\det(M)\theta + 2^\kappa\lambda = m_\theta$. Un tel système a une solution si et seulement si $\operatorname{pgcd}(\det M,\ 2^\kappa)$ divise m_θ. Soient (u, v) les coefficients de Bézout correspondant, c'est-à-dire, une paire d'entiers satisfaisant $\det(M)u + v2^\kappa = \operatorname{pgcd}(\det M,\ 2^\kappa)$. Quand le système a une solution, les solutions de l'équation sont :

$$\theta = u\frac{m_\theta}{\operatorname{pgcd}(\det M,\ 2^\kappa)} + n\frac{2^\kappa}{\operatorname{pgcd}(\det M,\ 2^\kappa)},$$

où $n \in \mathbb{Z}$.

Cela ne donne généralement que peu de solutions car, la plupart du temps, le pgcd est égal à 4. Mais il peut devenir plus grand et alors la recherche du triplet (a, b, c) devient trop coûteuse. Il y a alors deux façons de résoudre ce problème. La première consiste à utiliser aussi les antécédents

de $4, 5 \ldots$ et à s'assurer d'obtenir un système ayant moins de solutions. La seconde façon de procéder est d'ignorer simplement le triplet (i, j, k) qui mène au grand pgcd. La probabilité d'éliminer le bon triplet (a, b, c) est seulement $\frac{1}{2^{45}}$. Cette probabilité étant très faible, on procéde de la sorte dans un premier temps, sachant qu'on ne rencontre un tel triplet (i, j, k) qu'en moyenne deux fois par attaque. Si l'attaque échoue, il est toujours possible de revenir au cas précédent et de tester ces triplets.

Nous avons maintenant obtenu un ensemble de valeurs possibles pour (a, b, c) et pouvons passer à une recherche, comme décrit précédemment, mais limitée à cet ensemble restreint.

Description détaillée de l'attaque

L'algorithme de l'attaque est détaillé dans l'algorithme 5.3. La première étape consiste à calculer tous les $v_i = m_i \oplus c_i$. Les v_i ainsi obtenus correspondent à tous les octets de la clef K commençant respectivement aux bits $P(0), P(1), \ldots, P(2^\kappa - 1)$ avec $P(x) = ax^3 + bx^2 + cx + d \mod 2^\kappa$, mais sans leur ordre. Cela nous permet de déterminer un petit ensemble de triplets (α, β, γ) parmi lesquels se trouvent toutes les antécédents potentiels de $(1, 2, 3)$ par $\widetilde{P}$.

Si $\widetilde{P}$ est le polynôme de permutation recherché, v_0 nous donne de l'information sur α tel que $\widetilde{P}(\alpha) = 1$. En fait, les 7 premiers bits de v_α correspondent aux 7 derniers bits de v_0 car nous avons : $v_0 = \widetilde{K}_0, \widetilde{K}_1, \ldots, \widetilde{K}_7$ et $v_\alpha = K_{\{\widetilde{P}(\alpha) \to \widetilde{P}(\alpha)+7\}} = K_{\{1 \to 8\}} = \widetilde{K}_1, \ldots, \widetilde{K}_7, \widetilde{K}_8$. La figure 5.1 décrit bien le phénomène utilisé.

FIGURE 5.1 – Chevauchement des octets successifs de suite chiffrante

$$v_0 = \widetilde{K}_0, \underbrace{\widetilde{K}_1, \ldots, \widetilde{K}_7}$$

$$v_\alpha = \overbrace{\widetilde{K}_{\widetilde{P}(\alpha)}, \underbrace{\ldots, \widetilde{K}_{\widetilde{P}(\alpha)+6}}, \widetilde{K}_{\widetilde{P}(\alpha)+7}}$$

$$v_\beta = \overbrace{\widetilde{K}_{\widetilde{P}(\beta)}, \underbrace{\ldots, \widetilde{K}_{\widetilde{P}(\beta)+6}}, \widetilde{K}_{\widetilde{P}(\beta)+7}}$$

$$v_\gamma = \overbrace{\widetilde{K}_{\widetilde{P}(\gamma)}, \ldots, \widetilde{K}_{\widetilde{P}(\gamma)+6}}, \widetilde{K}_{\widetilde{P}(\gamma)+7}$$

Algorithme 5.3 Attaque de FCE

Entrées :
- Une demi page de clair, $\{m_i\}_{i\in\{0,\dots,2^\kappa-1\}}$.
- Une demi page de chiffré, $\{c_i\}_{i\in\{0,\dots,2^\kappa-1\}}$.

Sorties :
- La clef.

// Étape 1 : récupération de la suite chiffrante
pour tout i de 0 à $2^\kappa - 1$ **faire**
  $v_i \leftarrow m_i \oplus c_i$
fin pour
// Étape 2 : trouver les antécédents de $(1,2,3)$ par P
$\mathcal{E}_1 = \left\{\alpha \text{ impair } | v_{\alpha\{0\to 6\}} = v_0 \gg 1\right\}$
$\mathcal{E}_2(x) = \left\{\beta \ | v_{\beta\{0\to 6\}} = (v_0 \gg 2, x)\right\}$
$\mathcal{E}_3(x,y) = \left\{\gamma \text{ impair } | v_{\gamma\{0\to 6\}} = (v_0 \gg 3, x, y)\right\}$

// Étape 3 : filtrer les (a,b,c)
$\mathcal{L} \leftarrow \emptyset$
pour tout $\alpha \in \mathcal{E}_1$ **faire**
  $x_\alpha \leftarrow$ bit de poids fort de v_α
  pour tout $\beta \in \mathcal{E}_2(x_\alpha)$ **faire**
    $y_\beta \leftarrow$ bit de poids fort de v_β
    pour tout $\gamma \in \mathcal{E}_3(x_\alpha, y_\beta)$ **faire**
      si le système (5.2) a une solution (a,b,c) **alors**
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{(a,b,c)\}$
      fin si
    fin pour
  fin pour
fin pour

// Étape 4 : réduire $\mathcal{L}$ et trouver d
pour tout $(a,b,c) \in \mathcal{L}$ **faire**
  si nous réussissons à assembler $\widetilde{K}$ **alors**
    pour tout $d \in \{0,\dots,2^\kappa - 1\}$ **faire**
      $K \leftarrow \left(\widetilde{K} \ggg d\right)$
      $\left(\widetilde{a},\widetilde{b},\widetilde{c},\widetilde{d}\right) \leftarrow E(j,K)$ (j, le numéro de la page)
      si $\left(\widetilde{a},\widetilde{b},\widetilde{c},\widetilde{d}\right) == (a,b,c,d)$ **alors**
        retourner K
      fin si
    fin pour
  fin si
fin pour

Cela nous permet de construire trois ensembles, $\mathcal{E}_1$, $\mathcal{E}_2$ et $\mathcal{E}_3$, qui contiennent respectivement les antécédents potentiels de 1, 2 et 3 par $\widetilde{P}$. Plus précisément, pour $x, y \in \{0, 1\}$, nous définissons :

$$\begin{array}{lcl} \mathcal{E}_1 & = & \left\{\alpha \text{ impair } | v_{\alpha\{0\to 6\}} = v_0 \gg 1\right\} \\ \mathcal{E}_2(x) & = & \left\{\beta \ | v_{\beta\{0\to 6\}} = (v_0 \gg 2, x)\right\} \\ \mathcal{E}_3(x,y) & = & \left\{\gamma \text{ impair } | v_{\gamma\{0\to 6\}} = (v_0 \gg 3, x, y)\right\} \end{array}$$

Leur taille moyenne est donnée par le nombre total de valeurs possibles pour les antécédents divisé par 2^7 car 7 bits de l'octet de suite chiffrante correspondante sont connus. Pour les paramètres donnés, nous avons alors $|\mathcal{E}_1| \sim 2^7$, $|\mathcal{E}_2(x)| \sim 2^8$,et $|\mathcal{E}_3(x,y)| \sim 2^7$.

La troisième étape consiste à résoudre les systèmes de Vandermonde donnés par les triplets (α, β, γ) provenant de $\mathcal{E}_1$, $\mathcal{E}_2$ et $\mathcal{E}_3$. Cela nous donne un petit ensemble $\mathcal{L}$ des valeurs possibles pour (a, b, c), dans lequel nous savons que le triplet recherché figure.

Dans la quatrième et dernière étape, nous essayons de construire la clef qui correspond à chaque triplet restant, et, en cas de succès, nous cherchons alors le bon d correspondant à cette page.

Complexité

Nous devons maintenant évaluer le coût de cette attaque. La table 5.1 résume les coûts respectifs des quatre étapes de l'attaque.

TABLE 5.1 – Coût de l'attaque

Étape	Coût	
	théorique	pratique
1	k XORs	2^{15}
2	$7k$ masques et comparaisons	7.2^{15}
3	$\left(\frac{k}{2^7}\right)^3$ résolutions de systèmes 3×3	2^{22}
4	k appels au chiffrement E	2^{15}

Ainsi, le coût total de l'attaque est 2^{15} appels à l'algorithme de chiffrement E plus $\sim 2^{25}$ multiplications sur 16 bits.

Comme le montre cette attaque, l'hypothèse selon laquelle FCE était aussi sûr que le chiffrement par blocs utilisé est clairement erronée. En effet, nous voyons que notre attaque, quel que soit le chiffrement par blocs utilisé, qu'il soit sûr comme AES ou plus faible comme DES, fonctionne avec la même complexité, la seule différence perceptible étant dans la complexité des appels à l'algorithme de chiffrement.

Simulations

Nous avons effectué 300 attaques sur deux ordinateurs différents (150 attaques sur chacun). Nous avons adopté la méthode consistant à ignorer les triplets (i, j, k) quand ils aboutissaient à un pgcd plus grand que 16. Le taux de succès est de 100%. Le nombre de triplets ignorés varie entre 0 et 6 dans chaque attaque et vaut en moyenne 1,85. Le temps pris par l'attaque est détaillé dans le tableau 5.2.

TABLE 5.2 – Temps des attaques (en secondes)

Type de processeur	Temps		
	minimum	maximum	moyen
Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz	283 s	784 s	514 s
Intel(R) Xeon(R) CPU 5120 @ 1.86GHz	479 s	1295 s	828 s

Conclusion

Cette attaque met en évidence le manque de sécurité de l'algorithme FCE, car il est possible de récupérer la clef et de décrypter l'intégralité du contenu de la base de données chiffrée en seulement quelques minutes à partir de la seule connaissance de 2^{15} octets de clair et de chiffré avec les paramètres proposés.

5.2.3 Cryptanalyse de la variante de FCE

La variante de FCE [Ge10] utilisant des polynômes dans le corps fini $GF(2^n)$ a des implications sur la cryptanalyse, qui doit être adaptée en fonction. La première différence est que nous avons maintenant besoin d'une page entière de clair et de la page associée de chiffré pour pouvoir récupérer la clef et l'intégralité du contenu de la base de données en clair.

Principe de l'attaque

Nous utiliserons dans cette partie les notations suivantes : $k = 2^\kappa$ pour la taille de la clef et $p = 2^n$; il en découle $\kappa = 15$ et $n = 16$ avec l'ensemble de paramètres proposés pour FCE.

Comme nous l'avons déjà vu, il est beaucoup plus efficace de procéder à une recherche exhaustive sur les polynômes que sur la clef.

Ici, le coefficient du terme constant d du polynôme P joue un rôle particulier. En effet, pour une page donnée, pour $P(x) = \widetilde{P}(x) + d$, avec $d = P(0)$,

on déduit de (5.1) que :

$$d_i = \phi^{-1}\left(P\left(\phi(i)\right)\right) = \phi^{-1}\left(\widetilde{P}\left(\phi(i)\right)\right) \oplus \phi^{-1}\left(d\right) = \widetilde{d}_i \oplus d_0,$$

où d_0 désigne l'entier $\phi^{-1}(d)$.

Notre problème est le suivant. Nous voulons retrouver (a, b, c) à partir de la connaissance de $K_{\{d_i \to d_i+7\}} = \widetilde{K}_{\{\widetilde{d}_i \oplus d_0 \to (\widetilde{d}_i \oplus d_0)+7\}}$. En réalité, nous allons montrer que le triplet (a, b, c) peut être retrouvé à partir de la suite chiffrante par une recherche exhaustive sur les bits de poids faible de d_0.

La connaissance de tous les $\widetilde{K}_{\{\widetilde{d}_i \oplus d_0 \to (\widetilde{d}_i \oplus d_0)+7\}}$ nous fournit des informations sur les antécédents par $\widetilde{P}$ des éléments « successifs ». Pour une valeur donnée de d_0, cela nous donne en fait des ensembles de triplets (α, β, γ), parmi lesquels se trouvent tous les antécédents de $(1, \omega, 1+\omega)$ par $\widetilde{P}$.

Nous souhaitons alors trouver des ensembles de antécédents potentiels de $(1, \omega, 1+\omega)$. Le tableau 5.3 nous montre les relations entre $d_0 = \phi^{-1}(d)$ mod 4 et les valeurs de $\phi^{-1}(1+d)$, $\phi^{-1}(\omega+d)$ et $\phi^{-1}(1+\omega+d)$.

TABLE 5.3 – Relations entre les restes modulo 4 de d_0 et les antécédents de ϕ

d_0 mod 4	$\phi^{-1}(1+d)$ mod 4	$\phi^{-1}(\omega+d)$ mod 4	$\phi^{-1}(1+\omega+d)$ mod 4
0	d_0+1	d_0+2	d_0+3
1	d_0-1	d_0+2	d_0+1
2	d_0+1	d_0-2	d_0-1
3	d_0-1	d_0-2	d_0-3

De cette façon, en fonction des deux derniers bits de d_0, les antécédents de $(1, \omega, 1+\omega)$ seront dans l'un des ensembles suivants.

$$\begin{aligned}
\mathcal{E}_{-3}(t,u) &= \left\{\delta \mid v_{\delta\{1\to 7\}} = \left(u, t, v_{0\{0\to 4\}}\right)\right\}\\
\mathcal{E}_{-2}(t) &= \left\{\delta \mid v_{\delta\{1\to 7\}} = \left(t, v_{0\{0\to 5\}}\right)\right\}\\
\mathcal{E}_{-1} &= \left\{\delta \mid v_{\delta\{1\to 7\}} = v_{0\{0\to 6\}}\right\}\\
\mathcal{E}_{1} &= \left\{\delta \mid v_{\delta\{0\to 6\}} = v_{0\{1\to 7\}}\right\}\\
\mathcal{E}_{2}(x) &= \left\{\delta \mid v_{\delta\{0\to 6\}} = \left(v_{0\{2\to 7\}}, x\right)\right\}\\
\mathcal{E}_{3}(x,y) &= \left\{\delta \mid v_{\delta\{0\to 6\}} = \left(v_{0\{3\to 7\}}, x, y\right)\right\}
\end{aligned}$$

Lorsque nous essayons de reconstruire la clef K, nous choisirons les éléments dans l'ensemble adéquat, selon le tableau 5.3 précédent. La taille moyenne de ces ensembles est donnée par le nombre de toutes les valeurs possibles pour les antécédents divisé par 2^7 car 7 bits de l'octet correspondant de la suite chiffrante sont connus. Pour les paramètres considérés ici, nous avons alors $|\mathcal{E}_i| \sim 2^8$ pour tout i.

Ensuite, pour chaque valeur de $d_0 \mod 4$, nous résolvons les systèmes de Vandermonde dans $GF(2^n)$. Ils sont donnés par les triplets (α, β, γ) pris dans les ensembles $\mathcal{E}_i$ déterminés par $d_0 \mod 4$:

$$\begin{pmatrix} \alpha^3 & \alpha^2 & \alpha \\ \beta^3 & \beta^2 & \beta \\ \gamma^3 & \gamma^2 & \gamma \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ \omega \\ 1+\omega \end{pmatrix}. \tag{5.3}$$

Cela nous donne 4 ensembles $\mathcal{L}_{i,\ 0 \le i \le 3}$ de valeurs possibles pour (a, b, c). Une solution consiste à directement construire une ensemble plus grand : $\mathcal{L} = \bigcup_{i \in \{0 \ldots 3\}} \{(a, b, c, i) | (a, b, c) \in \mathcal{L}_i\}$. Une autre possibilité est de construire un nouvel ensemble $\mathcal{E}_4$, sur le même principe que pour $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$, où sont stockés les éléments j tels que $P(j) = 4$ est une éventualité crédible d'après les propriétés de chevauchement de la suite chiffrante. Nous cherchons alors un élément j appartenant à $\mathcal{E}_4$ tel que $\phi^{-1}(\widetilde{P}(\phi(j))) \mod 4 = i$, où $\widetilde{P}$ est généré à partir de $(a, b, c) \in \mathcal{L}_i$. Cela réduira la taille $\mathcal{L}$ légèrement, mais pas de façon significative.

Finalement, nous parcourons $\mathcal{L}$ et pour chaque quadruplet (a, b, c, δ), avec $0 \le \delta \le 3$, nous essayons de reconstruire la clef K pour chaque d tel que $\phi^{-1}(d) \equiv \delta \mod 4$. En cas de succès, nous n'avons plus qu'à vérifier si le quadruplet (a, b, c, d) correspond bien à celui dérivé de la clef que nous venons de construire et du numéro de la page.

L'algorithme de l'attaque sur la variante de FCE est détaillé en trois parties dans les algorithmes 5.4, 5.5, et 5.6.

Algorithme 5.4 Attaque de la variante de FCE - 1re partie

Entrées :
- Une page de clair, $\{m_i\}_{i \in \{0, \ldots, 2^\kappa - 1\}}$.
- Une page de chiffré, $\{c_i\}_{i \in \{0, \ldots, 2^\kappa - 1\}}$.

Sorties :
- La clef.

// Étape 1 : récupération de la suite chiffrante
pour tout i de 0 à $2^p - 1$ **faire**
  $v_i \leftarrow m_i \oplus c_i$
fin pour

// Étape 2 : calculer les ensembles pour tout $(x, y, u, t) \in GF(2^n)$

$$\begin{aligned}
\mathcal{E}_{-3}(t,u) &= \left\{\delta \mid v_{\delta\{0\to 6\}} = \left(u, t, v_{0\{0\to 4\}}\right)\right\} \\
\mathcal{E}_{-2}(t) &= \left\{\delta \mid v_{\delta\{1\to 7\}} = \left(t, v_{0\{0\to 5\}}\right)\right\} \\
\mathcal{E}_{-1} &= \left\{\delta \mid v_{\delta\{1\to 7\}} = v_{0\{0\to 6\}}\right\} \\
\mathcal{E}_{1} &= \left\{\delta \mid v_{\delta\{1\to 7\}} = v_{0\{1\to 7\}}\right\} \\
\mathcal{E}_{2}(x) &= \left\{\delta \mid v_{\delta\{0\to 6\}} = \left(v_{0\{2\to 7\}}, x\right)\right\} \\
\mathcal{E}_{3}(x,y) &= \left\{\delta \mid v_{\delta\{0\to 6\}} = \left(v_{0\{3\to 7\}}, x, y\right)\right\}
\end{aligned}$$

Algorithme 5.5 Attaque de la variante de FCE - 2e partie

// Étape 3 : filtrer les (a, b, c)
// Cas $d_0 \mod 4 = 0$
pour tout $\alpha \in \mathcal{E}_1$ **faire**
  $x_\alpha \leftarrow$ bit de poids fort de v_α
  pour tout $\beta \in \mathcal{E}_2(x_\alpha)$ **faire**
    $y_\beta \leftarrow$ bit de poids fort de v_β
    pour tout $\gamma \in \mathcal{E}_3(x_\alpha, y_\beta)$ **faire**
      Résoudre le système (5.3)
      pour tout $i \in \mathcal{E}_4$ **faire**
        si $ai^3 + bi^2 + ci \mod 4 = 0$ **alors**
          $\mathcal{L} \leftarrow \mathcal{L} \cup \{(a, b, c, 0)\}$
          continuer
        fin si
      fin pour
    fin pour
  fin pour
fin pour

// Cas $d_1 \mod 4 = 1$
pour tout $\alpha \in \mathcal{E}_{-1}$ **faire**
  pour tout $\gamma \in \mathcal{E}_1$ **faire**
    $x_\gamma \leftarrow$ bit de poids fort de v_γ
    pour tout $\beta \in \mathcal{E}_2(x_\gamma)$ **faire**
      Résoudre le système (5.3)
      pour tout $i \in \mathcal{E}_4$ **faire**
        si $ai^3 + bi^2 + ci \mod 4 = 1$ **alors**
          $\mathcal{L} \leftarrow \mathcal{L} \cup \{(a, b, c, 1)\}$
          continuer
        fin si
      fin pour
    fin pour
  fin pour
fin pour

Algorithme 5.6 Attaque de la variante de FCE - 3e partie

// Cas $d_2 \mod 4 = 2$
pour tout $\gamma \in \mathcal{E}_{-1}$ **faire**
  $x_\gamma \leftarrow$ bit de poids faible de v_γ
  pour tout $\beta \in \mathcal{E}_{-2}(x_\gamma)$ **faire**
    pour tout $\alpha \in \mathcal{E}_1$ **faire**
      Résoudre le système (5.3)
      pour tout $i \in \mathcal{E}_4$ **faire**
        si $ai^3 + bi^2 + ci \mod 4 = 2$ **alors**
          $\mathcal{L} \leftarrow \mathcal{L} \cup \{(a, b, c, 2)\}$
          continuer
        fin si
      fin pour
    fin pour
  fin pour
fin pour
// Cas $d_3 \mod 4 = 3$
pour tout $\alpha \in \mathcal{E}_{-1}$ **faire**
  $x_\alpha \leftarrow$ bit de poids faible de v_α
  pour tout $\beta \in \mathcal{E}_{-2}(x_\alpha)$ **faire**
    $y_\beta \leftarrow$ bit de poids faible de v_β
    pour tout $\gamma \in \mathcal{E}_{-3}(x_\alpha, y_\beta)$ **faire**
      Résoudre le système (5.3)
      pour tout $i \in \mathcal{E}_4$ **faire**
        si $ai^3 + bi^2 + ci \mod 4 = 3$ **alors**
          $\mathcal{L} \leftarrow \mathcal{L} \cup \{(a, b, c, 3)\}$
          continuer
        fin si
      fin pour
    fin pour
  fin pour
fin pour
// Étape 4 : réduire $\mathcal{L}$ et trouver d
pour tout $(a, b, c, \delta) \in \mathcal{L}$ **faire**
  pour tout d tel que $d \mod 4 = \delta$ **faire**
    si nous réussissons à assembler K **alors**
      $(a', b', c', d') \leftarrow E(j, K)$ (j, le numéro de la page)
      si $(a', b', c', d') == (a, b, c, d)$ **alors**
        retourner K
      fin si
    fin si
  fin pour
fin pour

Complexité

Nous devons maintenant évaluer le coût de cette attaque. Le tableau 5.4 résume le coût des quatre étapes de l'attaque

TABLE 5.4 – Coût de l'attaque sur la variante de FCE

Étape	Coût	
	théorique	pratique
1	p XORs	2^{16}
2	$13p$ masques et comparaisons	13.2^{16}
3	$4\left(\frac{p}{2^7}\right)^3$ 3×3 résolutions de systèmes (opérations bit à bit)	2^{29}
4	$\lvert\mathcal{L}\rvert p = 2^{16}\left(\frac{p}{2^7}\right)^3$ reconstruction de la clef	2^{43}

En pratique, il n'y a qu'un appel à l'algorithme de chiffrement par bloc E. Cela est dû au fait que la probabilité de reconstruire correctement une mauvaise clef est extrêmement faible. Ainsi, le coût principal de l'attaque réside dans les 2^{43} tentatives de reconstruire la clef.

L'augmentation de la complexité de l'attaque vient du fait que nous ne pouvons pas décorréler la recherche sur d de celle sur le triplet (a, b, c) dans cette variante. Mais le coût de l'attaque est néanmoins encore beaucoup plus faible que le coût de l'attaque naïve, qui consiste ici à essayer de reconstruire la clef pour tous les quadruplets possibles (a, b, c, d) et coûte donc 2^{64} tentatives de reconstruction de la clef.

Comme le montre notre attaque l'affirmation selon laquelle FCE est aussi sûr que le algorithme de chiffrement par blocs utilisé pour générer le polynôme de permutation à partir de la clef et du numéro de la page est également faux pour cette variante. Comme pour la version originale, notre attaque sur la variante de FCE fonctionne quelle que soit la sécurité du chiffrement E utilisé.

Simulations

Ici, il est plus difficile d'effectuer des simulations que pour la version « classique » de FCE, étant donné que la complexité est nettement supérieure. Les temps de nos tests varient entre 50 minutes dans le meilleur cas et 8 jours dans le pire cas.

Jusqu'à l'étape 3, l'algorithme de l'attaque est toujours très efficace et met moins d'une minute. La partie la plus lente de notre attaque est clairement la quatrième. Nos simulations montrent que le pire scénario, c'est-à-dire quand à la fois (a, b, c) et d se situent à la fin de la liste des possibilités, met

moins de 60 jours pour voir les calculs aboutis. Mais ces résultats pourraient bénéficier d'optimisations logicielles, ainsi que de parallélisation.

Il en résulte que, même si cette variante de FCE est plus robuste, elle n'assure pas une protection satisfaisante contre notre attaque.

Conclusions

Aucune des deux variantes de FCE proposées par T. Ge et S. Zdonik n'offre une sécurité satisfaisante. Au mieux, il faut 2^{16} octets de clair/chiffré pour récupérer la clef et l'intégralité des données chiffrées avec celle-ci en au maximum quelques jours.

Une solution raisonnable pour remplacer FCE est d'utiliser un chiffrement à flot sûr, comme par exemple ceux du portfolio eSTREAM [ECR05], avec un IV fixé par page. Cela serait moins efficace en temps que FCE, mais aurait un surcoût de stockage fortement réduit, ainsi qu'une réelle sécurité.

Conclusion

Notre analyse a donc permis une étude approfondie du chiffrement des bases de données. Ce travail a permis d'exhiber les composants cryptographiques nécessaires à une telle protection, de rendre compte de la législation en vigueur l'encadrant, en particulier celle concernant les données de santé, de passer en revue les produits existants, et finalement d'analyser la sécurité et les fonctionnalités offertes par diverses propositions, émanant des communautés de base de données ou de cryptographie.

Cela nous a permis d'étudier la sécurité de plusieurs propositions dont AREA, et de développer des cryptanalyses de solutions faibles comme FCE. Nous avons alors proposé des solutions sûres pour quelques cas simples (recherches dans la base de données limitées au cas d'égalité, recherches par intervalle avec déchiffrement à chaque nœud de l'arbre de l'index...).

Néanmoins, il reste encore de nombreuses pistes à explorer dans ce domaine. On peut citer en particulier l'étude du modèle de sécurité des chiffrements adaptables « narrow blocks » qui n'est actuellement pas précisément défini, comme travail théorique. D'un point de vue plus pratique, il serait aussi intéressant d'implémenter les solutions que nous proposons dans des moteurs de bases de données « open-source », ce qui permettrait, entre autres, d'avoir une meilleure idée des performances atteintes.

Annexe A

Introduction au droit des TIC

L'expression « technologies de l'information et de la communication » (TIC) désigne l'acquisition, le traitement, le stockage et la diffusion d'information vocale, imagée, textuelle et numérique par une combinaison d'informatique et de télécommunication. Pour encadrer ces techniques relativement récentes, dont font partie la cryptographie, l'utilisation de bases de données et l'hébergement à distance, le législateur a créé plusieurs lois que nous allons rapidement passer en revue.

A.1 Les principales lois sur les TIC

Après quelques généralités sur le droit et en particulier sur le droit des TIC, nous verrons les principales lois régissant les TIC.

A.1.1 Qu'est-ce que le droit ?

Le droit est constitué d'une grande variété de principes. Ils découlent de textes, de la jurisprudence, de la doctrine, des contrats et de la coutume. Les textes sont eux-mêmes multiples. Tout d'abord les textes nationaux comprennent la constitution, la loi et les textes d'application : décrets, arrêtés. . . . Mais ils ne sont pas les seuls, car il existe aussi des textes internationaux, principalement le droit communautaire et les conventions internationales signées par la France.

Tous ces éléments de droit n'ont pas la même importance. En fait, ils sont même hiérarchisés. Dans la figure A.1, on voit les deux principaux ordres ; la France suit le droit latin.

TABLE A.1 – Hiérarchie des éléments de droit

	Droit latin	**Droit anglo-saxon**	
↑	Contrat	Contrat	↑
	Doctrine	Doctrine	
	Jurisprudence	Lois et réglementation	
	Usages, déontologie, normes	Jurisprudence	
	Lois et réglementation	Usages, déontologie, normes	
	Traités internationaux	Traités internationaux	
	Constitution	Constitution	

A.1.2 Évolution du droit des TIC.

Comme les autres textes de droit, ceux concernant les TIC évoluent. La première grande loi encadrant les TIC est certainement la loi informatique et libertés du 6 janvier 1978 ; ensuite la loi Godfrain de 1988 apporte de nouveaux éléments concernant le droit de la sécurité des systèmes d'information. Finalement, le rythme s'est fortement accéléré dans les années 2000 avec l'avènement d'Internet. De façon non exhaustive, mais dans l'ordre chronologique, les lois suivantes impactent grandement le droit des TIC :

- loi n° 2001-1062 du 15 novembre 2001 relative à la sécurité quotidienne[1],
- loi n° 2002-1094 du 29 août 2002 d'orientation et de programmation pour la sécurité intérieure[2],
- loi n° 2003-239 du 18 mars 2003 pour la sécurité intérieure[3],
- loi n° 2004-575 du 21 juin 2004 pour la confiance dans l'économie numérique[4],
- loi n° 2004-669 du 9 juillet 2004 relative aux communications électroniques et aux services de communication audiovisuelle[5],
- loi n° 2004-801 du 6 août 2004 relative à la protection des personnes physiques à l'égard des traitements de données à caractère personnel[6]

[1] JORF n° 266 du 16 novembre 2001, page 18215, texte n° 1
[2] JORF du 30 août 2002, page 14398, texte n° 1
[3] JORF n° 66 du 19 mars 2003, page 4761, texte n° 1
[4] JORF n° 143 du 22 juin 2004, page 11168, texte n° 2
[5] JORF n° 159 du 10 juillet 2004, page 12483, texte n° 1
[6] JORF n° 182 du 7 août 2004, page 14063, texte n° 2

et modifiant la loi n° 78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés[7],

- loi n° 2009-669 du 12 juin 2009 favorisant la diffusion et la protection de la création sur internet[8]...

A.1.3 Communications électroniques.

Le droit des télécommunications a évolué vers un droit des communications électroniques. Alors que la loi du 29 décembre 1990 sur la réglementation des télécommunications définissait le terme « télécommunication », la loi du 9 juillet 2004 relative aux communications électroniques et aux services de communication audiovisuelle définie elle les « communications électroniques » de la façon suivante :

> « On entend par communications électroniques les émissions, transmissions ou réceptions de signes, de signaux, d'écrits, d'images ou de sons par voie électromagnétique. »

La tendance est à la libéralisation des communications électroniques, tout en garantissant un service public et en maintenant des régulateurs (ARCEP...). Ce genre de conditions ressemble à celles qui encadrent les hébergements de données de santé.

A.2 Loi informatique et libertés (1978) [L78]

La loi du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés [L78], modifiée par la loi du 6 août 2004 relative à la protection des personnes physiques à l'égard des traitements de données à caractère personnel, a pour but d'encadrer et de contrôler le développement de l'informatique. Elle vise principalement à défendre les libertés publiques et la vie privée des personnes physiques.

A.2.1 À quoi s'applique la loi ?

La loi informatique et libertés s'applique tout d'abord aux données personnelles définies comme suit :

> « Toute information relative à une personne physique identifiée ou qui peut être identifiée, directement ou indirectement, par référence à un numéro d'identification ou à un ou plusieurs éléments qui lui sont propres.[9] »

[7] JORF du 7 janvier 1978, page 227
[8] JORF n° 0135 du 13 juin 2009, page 9666, texte n° 2
[9] Article 2, alinéa 2 de la loi informatique et libertés du 6 janvier 1978 [L78].

Par contre, les données relatives aux personnes morales ne sont nullement protégées par la loi.

La notion de données personnelles est assez large. Dès qu'elles permettent de remonter à une personne et qu'il reste alors une possibilité de l'identifier, elles seront considérées comme personnelles. Celles relatives à l'état civil, comme le nom, le prénom, le sexe, les date et lieu de naissance, l'adresse... en font bien sûr parties. Tout comme toute donnée permettant d'établir un lien avec la personne physique concernée, comme par exemple un numéro d'immatriculation d'une automobile, un numéro de sécurité sociale..., mais aussi les caractéristiques physiques, comme les empreintes digitales, l'ADN, l'iris...

La loi informatique et libertés s'intéresse aux traitements automatisés et aux traitements non automatisés. Les seconds sont définis comme « tout ensemble structuré et stable de données à caractère personnel accessible selon des critères déterminés[10] », définition mettant en avant le critère d'organisation du fichier.

La définition de traitements automatisés, qui dépasse largement le cadre d'un simple fichier, et qui inclut en particulier le DMP, est la suivante :

> « Toute opération ou tout ensemble d'opérations portant sur de telles données, quel que soit le procédé utilisé, et notamment la collecte, l'enregistrement, l'organisation, la conservation, l'adaptation ou la modification, l'extraction, la consultation, l'utilisation, la communication par transmission, diffusion ou toute autre forme de mise à disposition, le rapprochement ou l'interconnexion, ainsi que le verrouillage, l'effacement ou la destruction.[11] »

A.2.2 La CNIL.

La loi informatique et libertés a mis en place la Commission nationale de l'informatique et des libertés[12] (Cnil) dont elle fixe les missions et pouvoirs, ainsi que son mode de fonctionnement.

La Cnil a tout d'abord une mission d'information, qui vise à informer les personnes concernées de leurs droits et obligations, à tenir à jour et publique la liste des traitements automatisés déclarés, et de présenter annuellement un rapport public rendant compte de l'exécution de sa mission. La Cnil émet aussi des conseils et recommandations sur demandes faites par des organisations professionnelles ou des institutions.

Elle a ensuite une mission de contrôle. La Cnil doit donc aussi veiller à la conformité des traitements vis-à-vis de la loi. Même si ce n'est pas

[10] Article 2, alinéa 1 de la loi informatique et libertés du 6 janvier 1978 [L78].
[11] Article 2, alinéa 3 de la loi informatique et libertés du 6 janvier 1978 [L78].
[12] Chapitre 3 de la loi informatique et libertés du 6 janvier 1978 [L78].

explicitement prévu par la loi, la Cnil s'accorde d'un pouvoir d'autosaisine. L'entrave à son action est un délit puni par la loi.

La Cnil a aussi un pouvoir réglementaire qui se limite à :

- l'établissement de son règlement intérieur,
- la production de normes simplifiées pour encadrer les traitements de données personnelles les plus courants,
- la formulation de règlements types pour assurer la sécurité des systèmes.

Finalement, la Cnil a le pouvoir de prononcer des sanctions administratives : mises en demeure, publication des avertissements et sanctions émises, injonction de cessation du traitement, sanctions pécuniaires. Un recours contre de telles sanctions se fait devant le Conseil d'État.

A.2.3 Obligations et droits induits par cette loi.

La loi informatique et libertés induit des obligations au responsable du traitement et des droits à la personne physique concernée.

Le responsable du traitement est défini comme la personne, l'autorité publique, le service ou l'organisme qui détermine les finalités et les moyens du traitement. Il a trois principales obligations :

- le traitement doit être licite,
- le responsable du traitement doit déterminer les finalités et les conséquences du traitement sur la qualité des données personnelles,
- il doit assurer la sécurité et la confidentialité des données.

De plus, la transmission des données personnelles hors Union européenne est strictement encadrée.

La personne concernée se voit accorder plusieurs droits. Le premier est qu'elle doit être informée de plusieurs informations (identité du responsable du traitement, finalité du traitement, droit d'opposition, droit d'interrogation...), cette liste étant définie par la loi. La personne concernée dispose aussi d'un droit d'interrogation sur ses données, ainsi qu'un droit de rectification. Finalement, elle a un droit d'opposition à ce que ses données à caractère personnel fassent l'objet d'un traitement.

A.3 Loi sur la confiance dans l'économie numérique (2004) [L04a]

A.3.1 Contexte.

La loi du 21 juin 2004 pour la confiance dans l'économie numérique transpose la directive européenne du 8 juin 2000 sur le commerce électronique, tout

en y ajoutant de nombreux nouveaux éléments. Cette transposition aurait pourtant dû être faite avant le 17 janvier 2002. Elle traite en particulier de la communication publique en ligne, de la publicité par voie électronique, du commerce électronique, des contrats et des écrits électroniques, des moyens et prestations de cryptologie et de la lutte contre la cybercriminalité.

A.3.2 Apports.

La LCEN apporte à la fois de nouvelles responsabilités et de nouvelles procédures.

Selon la LCEN, « on entend par communication au public en ligne toute communication audiovisuelle transmise sur demande individuelle formulée par un procédé de communication. » Dans ce cadre, de nouvelles responsabilités incombent aux fournisseurs d'accès, aux hébergeurs, aux éditeurs de services de communication en ligne, aux transporteurs et aux prestataires de « caching »[13].

Pour mettre en œuvre ces responsabilités, la LCEN instaure les procédures suivantes :

- procédure de notification des abus, auprès de l'hébergeur,
- « référé cessation de diffusion », auprès de tous les acteurs,
- requête en cessation de diffusion,
- requête en saisie de contrefaçon.

A.3.3 Nouveautés liées à la cryptographie.

La LCEN apporte aussi de nombreuses nouveautés en ce qui concerne les contrats et écrits électroniques, ainsi que la cryptologie et la certification électronique.

Concernant la signature électronique, la LCEN modifie le Code civil en précisant que « lorsqu'un écrit est exigé pour la validité d'un acte juridique, il peut être établi et conservé sous forme électronique dans les conditions prévues aux articles 1316-1 et 1316-4 et, lorsqu'un acte authentique est requis, au second alinéa de l'article 1317. » De plus, il en est de même en ce qui concerne la signature.

Dans le cadre de contrats électroniques, leurs « conditions contractuelles applicables » doivent être fournies par la personne qui propose, par voie électronique, la fourniture de biens ou la prestation de services. Elles doivent de plus pouvoir être conservées et reproduites.

[13] Le « caching » est ainsi défini par l'article 13 de la directive du 8 juin 2000 sur le commerce électronique : « le stockage automatique, intermédiaire et temporaire de l'information fait dans le seul but de rendre plus efficace la transmission ultérieure de l'information à la demande d'autres destinataires de service ».

La LCEN modifie de façon notable la partie concernant la cryptologie de la loi du 26 juillet 1996 de réglementation des télécommunications. En effet, elle libéralise sans aucune réserve l'utilisation des moyens de cryptologie. La LCEN encadre aussi le transfert et l'importation de moyens de cryptologie vers les pays de l'Union européenne de façon un peu plus stricte.

Finalement, cette loi précise l'étendue de la responsabilité des prestataires de certification électronique en les assortissant de sanctions pénales dissuasives, pouvant aboutir à des peines d'emprisonnement allant d'un à deux ans.

Bibliographie

[AB01] J. H. An et M. Bellare : Does Encryption with Redundancy Provide Authenticity? *In Advances in Cryptology - EUROCRYPT 2001*, volume 2045 de *Lecture Notes in Computer Science*, pages 512–528. Springer, 2001.

[ABC+10] N. Anciaux, L. Bouganim, A. Canteaut, S. Jacob, P. Pucheral et M. Videau : Identification des carences technologiques à la mise en œuvre des textes juridiques sur les données à caractère personnel. Livrable 5 du projet DEMOTIS, juin 2010. Version 1.0.

[ABG] N. Anciaux, L. Bouganim et Y. Guo : Cryptographic building blocks for database confidentiality and integrity. Working paper.

[ACGS88] W. Alexi, B. Chor, O. Goldreich et C.P. Shnorr : RSA and Rabin functions: certain parts are as hard as the whole. *SIAM Journal on Computing*, 17:194–209, 1988.

[AKSX04] R. Agrawal, J. Kiernan, R. Srikant et Y. Xu : Order preserving encryption for numeric data. *In ACM SIGMOD Conference 2004*, pages 563–574. ACM, 2004.

[AR99] Y. Aumann et M.O. Rabin : Information theoretically secure communication in the limited storage space model. *In Advances in Cryptology - CRYPTO 1999*, volume 1666 de *Lecture Notes in Computer Science*, pages 65–79. Springer-Verlag, 1999.

[Bar08] E. Barbry : *DTIC cours de droit - Introduction générale*. Télécom ParisTech, 2008. Polycopié de cours.

[BBKN01] M. Bellare, A. Boldyreva, L. R. Knudsen et C. Namprempre : Online ciphers and the hash-CBC construction. *In Advances in Cryptology - CRYPTO 2001*, volume 2139 de *Lecture Notes in Computer Science*, pages 292–309. Springer, 2001.

[BBO07] M. Bellare, A. Boldyreva et A. O'Neill : Deterministic and efficiently searchable encryption. *In Advances in Cryptology*

- *CRYPTO 2007*, volume 4622 de *Lecture Notes in Computer Science*, pages 535–552. Springer, 2007.

[BCJ+05] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet et W. Jalby : Collisions of SHA-0 and reduced SHA-1. *In Advances in Cryptology - EUROCRYPT 2005*, volume 3494 de *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.

[BCK96] M. Bellare, R. Canetti et H. Krawczyk : Keying hash functions for message authentication. *In Advances in cryptology - CRYPTO 1996*, volume 1109 de *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.

[BCLO09] A. Boldyreva, N. Chenette, Y. Lee et A. O'Neill : Order-preserving symmetric encryption. *In Advances in Cryptology - EUROCRYPT 2009*, volume 5479 de *Lecture Notes in Computer Science*, pages 224–241. Springer, 2009.

[BDJR97] M. Bellare, A. Desai, E. Jokipii et P. Rogaway : A concrete security treatment of symmetric encryption. *In Symposium on Foundations of Computer Science - FOCS 1997*, pages 394–403. IEEE Computer Society, 1997.

[BDPA10] G. Bertoni, J. Daemen, M. Peeters et G. Van Assche : Sponge-based pseudo-random number generators. *In Cryptographic Hardware and Embedded Systems - CHES 2010*, volume 6225 de *Lecture Notes in Computer Science*, pages 33–47. Springer, 2010.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval et P. Rogaway : Relations among notions of security for public-key encryption schemes. *In Advances in Cryptology - CRYPTO 1998*, volume 1462 de *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.

[BG10] L. Bouganim et Y. Guo : Database encryption. *In Encyclopedia of Cryptography and Security*. Springer, 2e édition, 2010.

[BHK+99] J. Black, S. Halevi, H. Krawczyk, T. Krovetz et P. Rogaway : UMAC: Fast and Secure Message Authentication. *In Advances in Cryptology - CRYPTO 1999*, volume 1666 de *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.

[Bih94] E. Biham : New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, 7:229–246, 1994.

[BKN09] A. Biryukov, D. Khovratovich et I. Nikolic : Distinguisher and related-key attack on the full AES-256. *In Advances in Cryptology - CRYPTO 2009*, volume 5677 de *Lecture Notes in Computer Science*, pages 231–249. Springer, 2009.

[BKR94] M. Bellare, J. Kilian et P. Rogaway : The security of cipher block chaining. *In Advances in cryptology - CRYPTO 1994*, volume 839 de *Lecture Notes in Computer Science*, pages 341–358. Springer, 1994.

[Blo70] B. H. Bloom : Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, juillet 1970.

[BN00] M. Bellare et C. Namprempre : Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *In Advances in Cryptology - ASIACRYPT 2000*, volume 1976 de *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[BR05] M. Bellare et P. Rogaway : Introduction to modern cryptography. http://cseweb.ucsd.edu/~mihir/cse207/classnotes.html, 2005.

[BRW04] M. Bellare, P. Rogaway et D. Wagner : The EAX mode of operation. *In Fast Software Encryption - FSE 2004*, volume 3017 de *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004. http://www.cs.ucdavis.edu/~rogaway/papers/eax.pdf.

[BS91] E. Biham et A. Shamir : Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[Can06] A. Canteaut : *Analyse et conception de chiffrements à clef secrète*. Mémoire d'habilitation à diriger des recherches, Université Paris 6, septembre 2006. http://www-rocq.inria.fr/codes/Anne.Canteaut/canteaut-hdr.pdf.

[Can08] A. Canteaut : Le chiffrement à flot. Cours à Télécom ParisTech, janvier 2008.

[CK01] R. Canetti et H. Krawczyk : Analysis of key-exchange protocols and their use for building secure channels. *In Advances in Cryptology - EUROCRYPT 2001*, volume 2045 de *Lecture Notes in Computer Science*, pages 453 – 474. Springer, 2001.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest et C. Stein : Binary search trees. *In Introduction to Algorithms*, chapitre 12, page 272. MIT Press, 3e édition, 2009.

[CM05] Y.-C. Chang et M. Mitzenmacher : Privacy preserving keyword searches on remote encrypted data. *In Applied Cryptography and Network Security - ACNS 2005*, volume 3531 de *Lecture Notes in Computer Science*, pages 442–455, 2005.

[Cod70] E. F. Codd : A relational model of data for large shared data banks. *Communications of the ACM*, 13:377–387, juin 1970.

[Com79] D. Comer : Ubiquitous B-Tree. *ACM Computing Surveys*, 11:121–137, juin 1979.

[Dam89] I. Damgård : A design principle for hash functions. *In Advances in cryptology - CRYPTO 1989*, volume 435 de *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[DCS04] DCSSI : La défense en profondeur appliquée aux système d'information. Mémento, Direction centrale de la sécurité des systèmes d'information (DCSSI), juillet 2004. Version 1.1, http://www.ssi.gouv.fr/IMG/pdf/mementodep-v1-1.pdf.

[DDN00] D. Dolev, C. Dwork et M. Naor : Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[DdVJ+03] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi et P. Samarati : Balancing confidentiality and efficiency in untrusted relational DBMSs. *In ACM Conference on Computer and Communications Security - CCS 2003*, pages 93–102. ACM, 2003.

[Dob96] H. Dobbertin : Cryptanalysis of MD4. *In Fast Software Encryption - FSE 1996*, volume 1039 de *Lecture Notes in Computer Science*, pages 53–69. Springer, 1996.

[Dor11] L. Dorrendorf : Protecting drive encryption systems against memory attacks. http://eprint.iacr.org/2011/221, mai 2011.

[ECL+07] R. Elbaz, D. Champagne, R. B. Lee, L. Torres, G. Sassatelli et P. Guillemin : TEC-Tree: A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks. *In Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 de *Lecture Notes in Computer Science*, pages 289–302. Springer, 2007.

[ECR05] ECRYPT - European Network of Excellence in Cryptology : The eSTREAM Stream Cipher Project. http://www.ecrypt.eu.org/stream/, 2005.

[ECR10] European Network of Excellence in Cryptology II ECRYPT II : ECRYPT II yearly report on algorithms and keysizes. rapport annuel D.SPA.7, Katholieke Universiteit Leuven (KUL), mars 2010.

[EFF98] Electronic Frontier Foundation EFF : *Cracking DES*. O'Reilly Media, juillet 1998.

[EJ02] P. Ekdahl et T. Johansson : A new version of the stream cipher SNOW. *In Selected Areas in Cryptography - SAC 2002*, volume 2295 de *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 2002.

[EJT07] H. Englund, T. Johansson et M. S. Turan : A framework for chosen IV statistical analysis of stream ciphers. *In Progress in Cryptology - INDOCRYPT 2007*, volume 4859 de *Lecture Notes in Computer Science*, pages 268–281. Springer, 2007.

[ETS+06] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, M. Bardouillet et A. Martinez : A parallelized way to provide data encryption and integrity checking on a processor-memory bus. *In Design Automation Conference - DAC 2006*, pages 506–509. ACM, 2006.

[EWSG04] Y. Elovici, R. Waisenberg, E. Shmueli et E. Gudes : A structure preserving database encryption scheme. *In Secure Data Management - VLDB 2004*, volume 3178 de *Lecture Notes in Computer Science*, pages 28–40. Springer, 2004.

[Fag07] P.-L. Fagniez : Le masquage d'informations par le patient dans son DMP. Rapport au ministre de la santé et des solidarités, Mission confiée au député Pierre-Louis Fagniez, janvier 2007.

[FIP77] FIPS PUB 46 : Data encryption standard (DES). FIPS Publications, janvier 1977. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[FIP93] FIPS PUB 180 : Secure hash standard. FIPS Publications, mai 1993.

[FIP95] FIPS PUB 180-1 : Secure hash standard. FIPS Publications, avril 1995. http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[FIP01] FIPS PUB 197 : Advanced encryption standard (AES). FIPS Publications, novembre 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[FIP02] FIPS PUB 180-2 : Secure hash standard. FIPS Publications, août 2002. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[FMVZ08] P.-A. Fouque, G. Martinet, F. Valette et S. Zimmer : On the security of the CCM encryption mode and of a slight variant. *In Applied Cryptography and Network Security - ACNS 2008*, volume 5037 de *Lecture Notes in Computer Science*, pages 411–428. Springer, 2008.

[Fru05] C. Fruhwirth : New methods in hard disk encryption. Vienna University of Technology, juillet 2005. http://clemens.endorphin.org/cryptography.

[GD01] V. D. Gligor et P. Donescu : Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. *In Fast Software Encryption - FSE 2001*, volume 2355 de *Lecture Notes in Computer Science*, pages 92–108. Springer, 2001.

[Ge10] T. Ge, janvier, février 2010 : Communication privée.

[Gen09] C. Gentry : Fully homomorphic encryption using ideal lattices. *In ACM Symposium on Theory of Computing - STOC 2009*, pages 169–178. ACM, 2009.

[GGR10] F. Guihéry, G. Guiheux et F. Rémi : Trusted computing : Limitations actuelles et perspectives. *In Symposium sur la sécurité des technologies de l'information et des communications - SSTIC 2010*, juin 2010. http://www.sstic.org/media/SSTIC2010/SSTIC-actes/Trusted_Computing_Limitations_actuelles_et_perspec/SSTIC2010-Article-Trusted_Computing_Limitations_actuelles_et_perspectives-remi_guihery_guiheux_.pdf.

[Gil03] H. Gilbert : The security of "One-Block-to-Many" Modes of Operation. *In Fast Software Encryption - FSE 2003*, volume 2887 de *Lecture Notes in Computer Science*, pages 376–395. Springer-Verlag, 2003.

[GJ09] Y. Guo et S. Jacob : Confidentialité et intégrité des grandes bases de données. *In Journées « Codage et Cryptographie » 2009*, Fréjus, Var, France, octobre 2009.

[GM84] S. Goldwasser et S. Micali : Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GN06] P. Gutmann et C. Ney : Shopping for an encrypted filesystem: Secret Candidates. Linux Magazine, n° 72, novembre 2006. http://www.linux-magazine.com/w3/issue/72/Encrypted_Filesystems_Review.pdf.

[Goh03] E.-J. Goh : Secure indexes. Technical report, Stanford University, 2003. http://crypto.stanford.edu/~eujin/papers/secureindex/.

[Gol06] D. Gollmann : *Computer Security*. John Wiley & Sons, 3e édition, 2006.

[Gra08] L. Granboulan : Symmetric cryptography and cryptanalysis basics. Cours au MPRI, janvier 2008.

[Guo11] Y. Guo : *Confidentialité et intégrité de bases de données embarquées*. Thèse de doctorat, Université de Versailles Saint-Quentin-en-Yvelines, décembre 2011.

[GZ07] T. Ge et S. Zdonik : Fast, secure encryption for indexing in a column-oriented DBMS. *In International Conference on Data Engineering - ICDE 2007*, pages 676–685. IEEE, 2007.

[Hal04] S. Halevi : EME*: Extending EME to Handle Arbitrary-Length Messages with Associated Data. *In Progress in Cryptology - INDOCRYPT 2004*, volume 3348 de *Lecture Notes in Computer Science*, pages 315–327. Springer, 2004.

[HHIM07] H. Hacigümüs, B. Hore, B. R. Iyer et S. Mehrotra : Search on encrypted data. *In Secure Data Management in Decentralized Systems*, volume 33 de *Advances in Information Security*, pages 383–425. Springer, 2007.

[HILM02] H. Hacigümüs, B. R. Iyer, C. Li et S. Mehrotra : Executing SQL over encrypted data in the database-service-provider model. *In ACM SIGMOD Conference 2002*, pages 216–227. ACM, 2002.

[HIM03] H. Hacigümüs, B. R. Iyer et S. Mehrotra : Ensuring the integrity of encrypted databases in the database-as-a-service model. *In Data and Applications Security and Privacy - DBSec 2003*, pages 61–74. Kluwer, 2003.

[Hit01] Hitachi, Ld : Mugi specification, 2001. http://www.sdl.hitachi.co.jp/crypto/mugi/mugi_spe.pdf.

[HJ08] M. Hell et T. Johansson : Breaking the F-FCSR-H stream cipher in real time. *In Advances in Cryptology - ASIACRYPT 2008*, pages 557–569. Springer-Verlag, 2008.

[HJM05] M. Hell, T. Johansson et W. Meier : Grain: A Stream Cipher for Constrained Environments. Soumission au projet eSTREAM [ECR05], 2005. http://www.ecrypt.eu.org/stream/.

[HK97] R. Canetti H. Krawczyk, M. Bellare : HMAC: Keyed-Hashing for Message Authentication. Internet RFC 2104, février 1997.

[HMI02] H. Hacigümüs, S. Mehrotra et B. R. Iyer : Providing database as a service. *In International Conference on Data Engineering - ICDE 2002*, pages 29–39. IEEE Computer Society, février 2002.

[HR03] S. Halevi et P. Rogaway : A tweakable enciphering mode. *In Advances in Cryptology - CRYPTO 2003*, volume 2729 de *Lecture Notes in Computer Science*, pages 482–499. Springer, 2003.

[HR04] S. Halevi et P. Rogaway : A parallelizable enciphering mode. *In Topics in Cryptology - CT-RSA 2004*, volume 2964 de *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.

[HSH+08] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum et E. W. Felten : Lest we remember: cold boot attacks on encryption keys. *In Proceedings of the 17th conference on Security symposium*, pages 45–60. USENIX Association, 2008.

[IEE08] IEEE Computer Society : IEEE standard for cryptographic protection of data on block-oriented storage devices. IEEE Standard 1619-2007, avril 2008. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04493450.

[IEE11] IEEE Computer Society : IEEE standard for wide-block encryption for shared storage media. IEEE Standard 1619.2-2010, mars 2011. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5729263.

[ISO05] ISO/IEC 18033-4 : Information technology - Security techniques - Encryption algorithms - Part 4: Stream ciphers, juin 2005. Amd 1 : 2009.

[ISO09] ISO/IEC : Information technology - security techniques - authenticated encryption, 2009.

[Iwa06] T. Iwata : New blockcipher modes of operation with beyond the birthday bound security. *In Fast Software Encryption - FSE 2006*, volume 4047 de *Lecture Notes in Computer Science*, pages 310–327. Springer, 2006.

[Iwa08] T. Iwata : Authenticated encryption mode for beyond the birthday bound security. *In Progress in Cryptology - AFRICACRYPT 2008*, volume 5023 de *Lecture Notes in Computer Science*, pages 125–142. Springer, 2008.

[IY09a] T. Iwata et K. Yasuda : BTM: A Single-Key, Inverse-Cipher-Free Mode for Deterministic Authenticated Encryption. *In Selected Areas in Cryptography - SAC 2009*, volume 5867 de *Lecture Notes in Computer Science*, pages 313–330. Springer, 2009.

[IY09b] T. Iwata et K. Yasuda : HBS: A Single-Key Mode of Operation for Deterministic Authenticated Encryption. *In Fast Software Encryption - FSE 2009*, volume 5665 de *Lecture Notes in Computer Science*, pages 394–415. Springer, 2009.

[Jac08] S. Jacob : Analyse de la résistance aux attaques algébriques des fonctions de filtrage augmentées. Mémoire de master, MPRI, master de l'École polytechnique, novembre 2008.

[Jac10a] S. Jacob : Cryptanalysis of a fast encryption scheme for database. *In IEEE International Symposium on Information Theory - ISIT 2010*, pages 2468–2472, Austin, USA, juin 2010. http://www-rocq.inria.fr/secret/Stephane.Jacob/documents/article-ieee.pdf.

[Jac10b] S. Jacob : Cryptanalysis of a fast encryption scheme for databases and of its variant. http://eprint.iacr.org/2010/554, octobre 2010.

[Jon02] J. Jonsson : The security of CTR + CBC-MAC. *In Selected Areas in Cryptography - SAC 2002*, volume 2595 de *Lecture Notes in Computer Science*, pages 76–93. Springer, 2002.

[Jut01] C. S. Jutla : Encryption modes with almost free message integrity. *In Advances in Cryptology - EUROCRYPT 2001*, volume 2045 de *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001.

[Ker83] A. Kerckhoffs : La cryptographie militaire. *Journal des sciences militaires*, 9:5–83, 161–191, janvier, février 1883.

[KG97] A. Klapper et M. Goresky : Feedback shift registers, 2-adic span, and combiners with memory. *Journal of Cryptology*, 10: 111–147, 1997.

[Knu98] D. Knuth : Binary tree searching. *In The Art of Computer Programming, Volume 3: Sorting and Searching*, chapitre 6.2.2, pages 426–458. Addison-Wesley, 2e édition, 1998.

[Knu00] L. Knudsen : Block chaining modes of operation. Reports in informatics, ISSN 0333-3590, report no 207, University of Bergen, octobre 2000. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/workshop1/papers/knudsen-iiaes.pdf.

[Koh89] J. T. Kohl : The use of encryption in kerberos for network authentication (invited). *In Advances in cryptology - CRYPTO 1989*, pages 35–43. Springer-Verlag New York, Inc., 1989.

[KR07] L. R. Knudsen et V. Rijmen : Known-key distinguishers for some block ciphers. *In Advances in Cryptology - ASIACRYPT 2007*, volume 4833 de *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.

[KR11] T. Krovetz et P. Rogaway : The software performance of authenticated-encryption modes. *In Fast Software Encryption - FSE 2011*, Lecture Notes in Computer Science. Springer, 2011.

[Kra94] H. Krawczyk : LFSR-based hashing and authentication. *In Advances in cryptology - CRYPTO 1994*, volume 839 de *Lecture Notes in Computer Science*, pages 129–139. Springer, 1994.

[Kra01] H. Krawczyk : The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). *In Advances in Cryptology - CRYPTO 2001*, volume 2139 de *Lecture Notes in Computer Science*, pages 310–331. Springer, 2001.

[Kro06] T. Krovetz : Message authentication on 64-bit architectures. *In Selected Areas in Cryptography - SAC 2006*, volume 4356 de *Lecture Notes in Computer Science*, pages 327–341. Springer, 2006.

[Küh06] U. Kühn : Analysis of a database and index encryption scheme - problems and fixes. *In Secure Data Management - VLDB 2006*, volume 4165 de *Lecture Notes in Computer Science*, pages 146–159. Springer, 2006.

[KVW04] T. Kohno, J. Viega et D. Whiting : CWC: A High-Performance Conventional Authenticated Encryption Mode. *In*

Fast Software Encryption - FSE 2004, volume 3017 de *Lecture Notes in Computer Science*, pages 408–426. Springer, 2004.

[KY00a] J. Katz et M. Yung : Complete characterization of security notions for probabilistic private-key encryption. *In ACM Symposium on Theory of Computing - STOC 2000*, pages 245–254. ACM, 2000.

[KY00b] J. Katz et M. Yung : Unforgeable encryption and chosen ciphertext secure modes of operation. *In Fast Software Encryption - FSE 2000*, volume 1978 de *Lecture Notes in Computer Science*, pages 284–299. Springer, 2000.

[L04a] Loi n° 2004-575 du 21 juin 2004 pour la confiance dans l'économie numérique. JORF n° 143 du 22 juin 2004. Version en vigueur : http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000005789847&dateTexte=vig.

[L04b] Loi n° 2004-810 du 13 août 2004 relative à l'assurance maladie (1). JORF n° 190 du 17 août 2004. Version en vigueur : http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000005824192&dateTexte=20110705=&dateTexte=vig.

[L78] Loi n° 78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés. JORF du 7 janvier 1978. Version en vigueur : http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000006068624&dateTexte=vig.

[Leu08] G. Leurent : MD4 is not one-way. *In Fast Software Encryption - FSE 2008*, volume 5086 de *Lecture Notes in Computer Science*, pages 412–428. Springer, 2008.

[LO05] J. Li et E. Omiecinski : Efficiency and security trade-off in supporting range queries on encrypted databases. *In Data and Applications Security XIX*, volume 3654 de *Lecture Notes in Computer Science*, pages 925–925. Springer Berlin / Heidelberg, 2005.

[LRW02] M. Liskov, R. L. Rivest et D. Wagner : Tweakable block ciphers. *In Advances in Cryptology - CRYPTO 2002*, volume 2442 de *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[Luc05] S. Lucks : Two-pass authenticated encryption faster than generic composition. *In Fast Software Encryption - FSE 2005*, volume 3557 de *Lecture Notes in Computer Science*, pages 284–298. Springer, 2005.

[Mat94] M. Matsui : Linear cryptanalysis method for DES cipher. *In Advances in Cryptology - EUROCRYPT 1993*, volume 765 de *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1994.

[Mau92] U. Maurer : Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, 1992. ftp://ftp.inf.ethz.ch/pub/crypto/publications/Maurer92b.pdf.

[MBPB11] A. Moise, E. Beroset, T. Phinney et M. Burns : EAX' Cipher Mode. NIST Modes Development, 2011. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/eax-prime/eax-prime-spec.pdf.

[McG05] D. A. McGrew : Efficient authentication of large, dynamic data sets using Galois/counter mode (GCM). *In IEEE Security in Storage Workshop - SISW 2005*, pages 89–94. IEEE Computer Society, 2005.

[Mer89] R. C. Merkle : A certified digital signature. *In Advances in cryptology - CRYPTO 1989*, volume 435 de *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[MF04] D. A. McGrew et S. Fluhrer : The extended codebook (XCB) mode of operation. http://eprint.iacr.org/2004/278.pdf, octobre 2004.

[MF07] D. A. McGrew et S. R. Fluhrer : The security of the extended codebook (XCB) mode of operation. *In Selected Areas in Cryptography - SAC 2007*, volume 4876 de *Lecture Notes in Computer Science*, pages 311–327. Springer, 2007.

[Mor07] F. Morain : *Cryptologie*. École polytechnique, 2007. Polycopié de cours.

[MV04] D. A. McGrew et J. Viega : The security and performance of the Galois/counter mode (GCM) of operation. *In Progress in Cryptology - INDOCRYPT 2004*, volume 3348 de *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[MvOV97] A.J. Menezes, P.C. van Oorshot et S.A. Vanstone : *Handbook of applied cryptography*. CRC Press, 1997. http://www.cacr.math.uwaterloo.a/hac/.

[NIS] NIST : Cryptographic hash algorithm competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[NIS01] NIST SP 800-38A : Recommendation for Block Cipher Modes of Operation: Methods and Technics. NIST Special Publication 800-38A, 2001. http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf.

[NIS04] NIST SP 800-38C : Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST Special Publication 800-38C, mai 2004. http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf.

[NIS05] NIST SP 800-38B : Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, mai 2005. http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf.

[NIS07] NIST SP 800-38D : Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, novembre 2007. http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[NIS10] NIST SP 800-38E : Recommendation for Block Cipher Modes of Operations: The XTS-AES Mode for Confidentiality on Storage Devices. NIST Special Publication 800-38E, janvier 2010. http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf.

[ÖSC03] G. Özsoyoglu, D. A. Singer et S. S. Chung : Anti-Tamper Databases: Querying Encrypted Databases. *In Data and Applications Security and Privacy - DBSec 2003*, pages 133–146. Kluwer, 2003.

[PGV93] B. Preneel, R. Govaerts et J. Vandewalle : Hash Functions Based on Block Ciphers: A Synthetic Approach. *In Advances in Cryptology - CRYPT0 1993*, volume 773 de *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.

[Poi02] D. Pointcheval : *Le chiffrement asymétrique et la sécurité prouvée*. Mémoire d'habilitation à diriger des recherches, Université Paris 7, juin 2002. http://www.di.ens.fr/users/pointche/Documents/Reports/2002_HDRThesis.pdf.

[PvO95] B. Preneel et P. C. van Oorschot : MDx-MAC and building fast MACs from hash functions. *In Advances in cryptology - CRYPTO 1995*, volume 963 de *Lecture Notes in Computer Science*, pages 1–14. Springer, 1995.

[Rab05] M.O. Rabin : Provably unbreakable hyper-encryption in the limited access model. *In Proceedings of the 2005 IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security*, pages 34–37. IEEE, 2005.

[RAD78] R. Rivest, L. Adleman et M. Dertouzos : On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169–177, 1978.

[RBB03] P. Rogaway, M. Bellare et J. Black : OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security*, 6(3):365–403, 2003.

[Riv91] R. Rivest : The MD4 message digest algorithm. *In Advances in Cryptology - CRYPT0 1990*, volume 537 de *Lecture Notes in Computer Science*, pages 303–311. Springer, 1991.

[Riv92] R. Rivest : The MD5 message digest algorithm. Internet RFC 1321, avril 1992.

[Riv99] R. L. Rivest : Permutation polynomials modulo 2^w. *In Finite Fields and their Applications*, volume 7, pages 287–292, 1999.

[Rog04] P. Rogaway : Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. *In Advances in Cryptology - ASIACRYPT 2004*, volume 3329 de *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[RS06] P. Rogaway et T. Shrimpton : A provable-security treatment of the key-wrap problem. *In Advances in Cryptology - EUROCRYPT 2006*, volume 4004 de *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[Rut09] J. Rutkowska : Evil Maid goes after TrueCrypt! Proof of concept, octobre 2009. http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html.

[Saa04] M.-J. O. Saarinen : Encrypted watermarks and linux laptop security. *In Information Security Applications, WISA 2004*, volume 3325 de *Lecture Notes in Computer Science*, pages 27–38. Springer, 2004.

[San09] ASIP Santé : Cahier des clauses techniques particulieres DMP1, octobre 2009. http://esante.gouv.fr/en/asip-sante/marches-publics/attributions-de-marche/aapc-dmp1-et-hebergement.

[Swe02] L. Sweeney : k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[SWP00] D. X. Song, D. Wagner et A. Perrig : Practical techniques for searches on encrypted data. *In IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[Vaz11] M. Vazirgiannis : INF 553 - data bases. Cours à l'École polytechnique, 2011.

[WY05] X. Wang et H. Yu : How to break MD5 and other hash functions. *In Advances in Cryptology - EUROCRYPT 2005*, volume 3494 de *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[WYY05] X. Wang, Y. L. Yin et H. Yu : Finding collisions in the full SHA-1. *In Advances in Cryptology - CRYPTO 2005*, volume 3621 de *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[YPM09] S. Yin, P. Pucheral et X. Meng : A sequential indexing scheme for flash-based embedded systems. *In Extending Databasec Technology - EDBT 2009*, volume 360, pages 588–599. ACM, 2009.

[ZLN08] Y. Zhang, W.-X. Li et X.-M. Niu : A secure cipher index over encrypted character data in database. *In Machine Learning and Applications - ICMLA 2008*, volume 2, pages 1111–1116. IEEE, juillet 2008.

[Zor10] C. Zorn : État des problèmes et pistes de résolution. Livrable 7 du projet DEMOTIS, juin 2010. Version 1.0.

Table des figures

Liste des algorithmes

Liste des tableaux

Table des matières