La cryptographie déchiffrée

Jean-Philippe Aumasson

La cryptographie déchiffrée

DUNOD

NOUS NOUS ENGAGEONS EN FAVEUR DE L'ENVIRONNEMENT :

Nos livres sont imprimés sur des papiers certifiés pour réduire notre impact sur l'environnement.

Le format de nos ouvrages est pensé afin d'optimiser l'utilisation du papier.

Depuis plus de 30 ans, nous imprimons 70 % de nos livres en France et 25 % en Europe et nous mettons tout en œuvre pour augmenter cet engagement auprès des imprimeurs français.

Nous limitons l'utilisation du plastique sur nos ouvrages (film sur les couvertures et les livres).

11 rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-084857-7

TABLE DES MATIÈRES

Avant-propos

J'ai écrit le livre que j'aurais aimé avoir quand j'ai commencé à apprendre la cryptographie. En 2005, j'étudiais pour un master près de Paris, et je me suis tout de suite inscrit au cours de cryptographie prévu pour le second semestre. Mais à mon grand désespoir, le cours fut annulé, trop peu d'étudiants l'ayant choisi. « La crypto, c'est compliqué », affirmaient les étudiants qui ont préféré suivre en nombre les classes de graphisme et de bases de données.

J'ai depuis souvent entendu cette affirmation. Mais la cryptographie est-elle *vraiment* si compliquée ? Que ce soit pour jouer d'un instrument de musique, maîtriser un langage de programmation, ou mettre en pratique les connaissances de tout domaine suffisamment complexe, il est nécessaire d'apprendre certains concepts de base et notations, mais cela ne nécessite pas un doctorat. Je pense qu'il en va de même pour devenir un cryptographe compétent. Je pense aussi que la cryptographie est perçue comme étant compliquée car les cryptographes ne l'ont pas suffisamment bien enseignée.

Une autre raison qui m'a motivé à écrire ce livre réside dans le fait que la crypto n'est plus seulement une affaire de crypto — elle est désormais un domaine multidisciplinaire. Pour faire quoi que ce soit d'utile et pertinent dans ce domaine, il vous faut comprendre l'environnement dans lequel elle est utilisée : comment les réseaux et systèmes informatiques fonctionnent, les besoins des utilisateurs, ou comment des adversaires peuvent exploiter les faiblesses d'algorithmes et de leurs implémentations. Autrement dit, vous devez être connecté à la réalité.

◆ *L'approche de ce livre*

Le titre initial du volume original était *Crypto for Real*, afin de souligner l'approche pratique, terre à terre, et sans fioriture que je souhaitais suivre. Pour rendre la cryptographie plus abordable sans pour autant trop la simplifier, j'ai choisi de m'appuyer sur des applications du monde réel. Cela se traduit notamment par des exemples de code et de vrais bugs catastrophiques.

Outre un lien clair avec la réalité, les fondements de ce livre sont sa simplicité et sa « modernité ». J'ai privilégié la simplicité dans la forme plutôt que dans le fond : je présente de nombreux concepts non triviaux, mais sans l'intimidant formalisme mathématique. J'essaie plutôt de transmettre les idées fondamentales de la cryptographie, qui sont plus importantes que la mémorisation d'une liste d'équations. Pour assurer la modernité du livre, je couvre de récentes techniques tels que TLS 1.3 et la cryptographie post-quantique. Je ne m'étale pas sur les algorithmes

obsolètes ou peu sûrs tels que DES ou MD5, bien que RC4 fasse exception à la règle, mais seulement pour discuter de ses défauts.

Serious Cryptography (*La Cryptographie déchiffrée*) n'est pas un guide d'utilisation des logiciels de cryptographie ni un recueil de spécifications techniques, lesquels sont aisément disponibles en ligne. L'objectif premier de ce livre est plutôt de vous enthousiasmer, de mettre fin au mythe selon lequel « la crypto, c'est compliqué », et d'enseigner les concepts fondamentaux en cours de route.

◆ *À qui s'adresse ce livre ?*

En écrivant, j'ai souvent imaginé le futur lecteur comme un développeur ayant été exposé à de la cryptographie, mais restant sur sa faim et frustré après avoir survolé de cryptiques ouvrages et articles scientifiques sur le sujet. Les développeurs ont en effet souvent besoin (et envie) de mieux comprendre la cryptographie afin de ne pas commettre de regrettables choix techniques ; j'espère que cet ouvrage les aidera.

Même si vous n'êtes pas développeur ou ingénieur, ne vous inquiétez pas ! Ce livre n'exige pas de connaissance en programmation, et est accessible à quiconque connaît les bases de l'informatique et possède des connaissances mathématiques du niveau secondaire (notions de probabilité et d'arithmétique modulaire, entre autres).

Cet ouvrage peut néanmoins être intimidant et, malgré son accessibilité, demandera un minimum d'attention pour en tirer le meilleur. Pour l'illustrer, j'aime à dresser une analogie entre les livres et l'alpinisme : tel un guide de montagne, l'auteur montre le chemin à suivre, fournit cordes et piolets pour faciliter l'ascension, mais le lecteur doit lui-même parcourir le chemin. Un effort pourra être nécessaire pour saisir certains des concepts de ce livre, mais vous en serez récompensé.

◆ *Comment ce livre est-il organisé ?*

Le livre contient seize chapitres, organisés en quatre parties. Les chapitres sont relativement indépendants les uns des autres, à l'exception du neuvième, qui décrit les bases des trois chapitres qui le suivent. Je vous conseille cependant de commencer par lire les trois premiers chapitres.

Les fondamentaux

- ✓ **Chapitre 1 : Le chiffrement** présente la notion de chiffrement sûr, depuis les simples schémas au crayon à papier jusqu'au solide chiffrement probabiliste.
- ✓ **Chapitre 2 : L'aléa** décrit comment les générateurs de nombres pseudo-aléatoires fonctionnent, quels sont leurs critères de sécurité et comment les utiliser de façon fiable.

- ✓ **Chapitre 3 : La sécurité cryptographique** couvre les notions théoriques et pratiques de sécurité cryptographique, comparant les approches « prouvable » et « probable ».

La cryptographie symétrique

- ✓ **Chapitre 4 : Les *blockciphers*** concerne les cryptosystèmes traitant les messages bloc par bloc, et en particulier le plus connu d'entre eux : l'*Advanced Encryption Standard* (AES).
- ✓ **Chapitre 5 : Les *streamciphers*** présente les cryptosystèmes produisant un flux de bits pseudo-aléatoires, qui sont XORés avec un message pour le (dé) chiffrer.
- ✓ **Chapitre 6 : Les fonctions de hachage** s'intéresse à des algorithmes qui fonctionnent sans clé secrète, et qu'on retrouve dans la plupart des protocoles cryptographiques.
- ✓ **Chapitre 7 : Hacher avec une clé** explique ce qui se passe quand on combine une clé secrète avec le concept de hachage, et comment s'en servir pour authentifier des messages.
- ✓ **Chapitre 8 : Le chiffrement authentifié** présente les algorithmes qui peuvent à la fois chiffrer et authentifier un message, avec comme exemple le standard AES-GCM.

La cryptographie asymétrique

- ✓ **Chapitre 9 : Les problèmes difficiles** décrit les concepts fondamentaux de la cryptographie à clé publique, en se reposant sur des notions d'informatique théorique du domaine de la théorie de la complexité.
- ✓ **Chapitre 10 : RSA** exploite le problème de la factorisation afin de construire des schémas de chiffrement et de signature sûrs, en se reposant sur une simple opération arithmétique.
- ✓ **Chapitre 11 : Diffie–Hellman** étend la cryptographie asymétrique à la notion d'échange de clé, grâce auquel deux parties établissent une valeur secrète commune en échangeant seulement des valeurs non secrètes.
- ✓ **Chapitre 12 : Les courbes elliptiques** fournit une introduction à la cryptographie basée sur les courbes elliptiques, qui offre des avantages notables en termes de performances.

Des applications

- ✓ **Chapitre 13** : **TLS** se concentre sur le protocole *Transport Layer Security* (TLS), sans doute le plus important protocole de sécurité réseau.
- ✓ **Chapitre 14** : **Quantique et post-quantique** présente les concepts de calcul quantique et de cryptographie post-quantique, un des sujets majeurs en cryptographie appliquée dans les années 2020.

- ✓ **Chapitre 15** : **Cryptomonnaies et cryptographie** conclut avec un aperçu des protocoles cryptographiques utilisés dans les applications blockchain ; ce chapitre est « le boss de fin » du livre.

◆ *Sur cette traduction*

Cette traduction de *Serious Cryptography* est publiée près de sept ans après la première édition. Depuis, la cryptographie et ses applications ont évolué. Aujourd'hui, le terme « crypto » évoque souvent les applications blockchain, le Bitcoin et autres cryptomonnaies, plutôt que la cryptographie elle-même. Malgré les avantages sociétaux discutables de ces technologies, leur influence positive indéniable sur l'avancement de la cryptographie ne peut être négligée. C'est pourquoi j'ai ajouté un nouveau chapitre intitulé « Cryptomonnaies et cryptographie », qui traite des techniques cryptographiques fascinantes employées dans les applications de la blockchain, représentant certaines des avancées les plus intéressantes dans le domaine de la cryptographie.

Réalisée en même temps que la mise à jour de *Serious Cryptography* pour sa seconde édition, cette traduction comporte des modifications substantielles par rapport à l'original. Son contenu a été mis à jour suite aux derniers développements en matière de recherche et application de la cryptographie. Parmi les changements significatifs : la discussion du chapitre 2 sur le générateur d'aléa du noyau Linux a été adaptée au nouveau comportement des interfaces */dev/random* et */dev/urandom* ; le chapitre 12 comporte une nouvelle section sur les schémas de signature EdDSA et Ed25519 ; et le chapitre 14 présente les standards de cryptographie post-quantique du NIST sélectionnés en 2022.

J'ai traduit le texte anglais original moi-même, souhaitant que *La Cryptographie déchiffrée* soit plus qu'une simple traduction, et connaissant la difficulté d'un tel exercice, notamment concernant la traduction des termes techniques. J'ai parfois conservé les termes anglais originaux, ma règle étant d'opter pour le terme le plus commun lors de discussions orales entre experts ; par exemple, je parle de *blockciphers* et non de chiffres par bloc. Lorsqu'un terme est traduit, je mentionne entre parenthèses le terme original, afin que les lecteurs puissent facilement rechercher le terme dans la littérature anglophone ; par exemple, je présente les signatures à seuil (*threshold signatures*). J'espère que cet ouvrage offrira aux lecteurs francophones, qu'ils soient étudiants ou professionnels aguerris, une compréhension approfondie des principes fondamentaux de la cryptographie, et qu'il s'avérera être un atout précieux dans leur parcours professionnel.

◆ *Remerciements (de l'édition originale)*

Je tiens à remercier Jan, Annie, ainsi que toute l'équipe de No Starch Press ayant contribué à ce livre, et tout particulièrement Bill pour avoir cru en ce projet dès le départ, et pour son laborieux travail d'édition ayant grandement contribué

à la qualité du texte. Je suis également reconnaissant à Laurel pour son attention à la mise en page.

Concernant la partie technique, un bon nombre d'erreurs et imprécisions ont été évitées grâce aux personnes suivantes : Jon Callas, Bill Cox, Niels Ferguson, Philipp Jovanovic, Samuel Neves, David Reid, Phillip Rogaway, Erik Tews, et les nombreux lecteurs qui ont signalé des fautes dans le texte. Je remercie Matthew Green de m'avoir fait l'honneur d'écrire la préface.

Je remercie de plus Kudelski Security, dont j'étais l'employé pendant l'écriture de ce livre, et qui m'a permis d'y dédier une partie de mon temps de travail. Ma profonde gratitude va enfin à Alexandra et Melina, pour leur soutien et leur patience.

Lausanne, le 17/05/2017 (trois nombres premiers)[1]

◆ *Remerciements (de la traduction)*

Je tiens tout d'abord à remercier les lecteurs de la première édition, en particulier ceux qui m'ont contacté pour me faire part de leurs commentaires et m'aider à améliorer le livre. Je suis reconnaissant envers Dunod, éditeur de l'ouvrage traduit, pour sa confiance et son professionnalisme (merci à Matthieu Daniel et Anne Temps). Merci à André Sintzoff pour sa relecture méticuleuse de l'ouvrage. Il va sans dire, les lacunes de ce livre sont les miennes.

Lausanne, le 28/02/2024

1. Date de l'écriture de la préface dans l'édition originale.

Préface

Si vous avez déjà lu un livre ou deux sur la sécurité informatique, vous aurez sûrement rencontré un jugement récurrent sur le domaine de la cryptographie. « La cryptographie », dit-on, « est le plus fort maillon de la chaîne ». Élogieux, certes, mais quelque peu simpliste. Si la cryptographie était en réalité le composant le plus fiable d'un système, pourquoi investir autant de temps à l'améliorer alors que tant d'autres parties du système pourraient bénéficier de notre attention ?

S'il y a une chose que j'espère que vous retiendrez de cet ouvrage, c'est que cette vision de la cryptographie est idéalisée ; c'est en grande partie un mythe. La cryptographie est solide *en théorie*, mais en pratique elle est aussi fragile que toute autre partie du système. C'est particulièrement vrai quand les implémentations sont développées sans l'expertise, l'expérience, et l'attention adéquate, comme c'est le cas de nombreux systèmes cryptographiques déployés aujourd'hui. Et plus grave encore : lorsque des implémentations cryptographiques ont des problèmes, les conséquences sont souvent remarquablement impressionnantes.

Mais pourquoi s'en inquiéter, et pourquoi ce livre ?

Quand j'ai commencé à travailler dans le domaine de la cryptographie appliquée il y a une vingtaine d'années, l'information disponible pour les développeurs de logiciels était souvent parcellaire et dépassée. Les cryptographes développaient des algorithmes et protocoles, les ingénieurs en cryptographie les implémentaient pour créer des bibliothèques impénétrables et mal documentées destinées principalement à d'autres experts. Il y avait — et il y a toujours — un grand clivage entre ceux qui connaissent et comprennent la cryptographie, et ceux qui l'utilisent (ou l'ignorent, à leurs risques et périls). Il y a peu de livres décents disponibles sur le sujet, et encore moins qui offrent des outils utiles aux professionnels.

Les résultats n'étaient pas beaux à voir. Je parle de problèmes impliquant les termes « CVE » et « Severity: High » et, dans certains cas alarmants, des présentations classées «TOP SECRET ». Vous connaissez peut-être certains des exemples les plus célèbres, ne serait-ce que parce qu'ils ont affecté des systèmes dont vous dépendez. Nombre de ces problèmes sont dus au fait que la cryptographie est subtile et mathématiquement élégante, et que les experts en cryptographie n'ont pas partagé leurs connaissances avec les ingénieurs qui écrivent les logiciels.

Heureusement, cela a commencé à changer et ce livre est un symptôme de ce changement.

Le présent ouvrage, *La Cryptographie déchiffrée*, a été écrit par l'un des plus grands experts en cryptographie appliquée mais il ne s'adresse pas seulement

aux experts. Il ne se limite toutefois pas à un aperçu superficiel du domaine. Au contraire, il offre une discussion approfondie et actualisée de l'ingénierie cryptographique, écrite pour aider les praticiens du domaine à mieux faire. Dans ces pages, vous apprendrez non seulement comment fonctionnent les algorithmes cryptographiques, mais aussi comment les utiliser dans de vrais systèmes.

Le livre commence par une exploration des principales primitives cryptographiques, dont les algorithmes de base tels que les chiffrements par bloc, schémas à clé publique, fonctions de hachage, ou générateurs de nombres aléatoires. Chaque chapitre présente des exemples concrets du fonctionnement des algorithmes, et ce que vous devez ou ne devez *pas* faire. Les derniers chapitres couvrent des sujets plus avancés, comme TLS, ainsi que l'avenir de la cryptographie — ou que faire lorsque des ordinateurs quantiques viendront nous compliquer la vie.

Bien qu'aucun livre ne puisse résoudre tous nos problèmes, un minimum de connaissances peut tout changer. C'est la promesse de cet ouvrage, qui en contient assez pour que la cryptographie réelle et déployée soit à la hauteur des grandes attentes qu'elle suscite chez beaucoup d'entre nous.

Bonne lecture.

Matthew D. Green

Professeur

Information Security Institute

Johns Hopkins University, USA

1 Le chiffrement

Le chiffrement est l'application principale de la cryptographie ; il rend les données incompréhensibles afin d'en assurer la *confidentialité*. Un système de chiffrement (*cipher* en anglais) utilise une valeur secrète appelée *clé* ; sans cette clé secrète, vous ne pouvez pas déchiffrer un message chiffré ni apprendre la moindre information sur son contenu, et aucun adversaire ne le peut non plus.

Note : On utilisera le verbe « chiffrer » plutôt que l'anglicisme « crypter ». De plus, on notera la différence entre « déchiffrer » (déterminer le message clair à partir du message chiffré à l'aide de la clé, en suivant la procédure de déchiffrement du cryptosystème), et « décrypter » (déterminer tout ou partie du message clair *sans connaître la clé de déchiffrement*, mais à l'aide de méthode de *cryptanalyse*, exploitant des imperfections de l'algorithme de chiffrement). On parlera de « chiffre » (*cipher*) pour désigner un « système de chiffrement » ou « cryptosystème ».

1. LES BASES

Lorsqu'on chiffre un message, *plaintext* fait référence au message en clair, et *ciphertext* au message chiffré (on utilisera souvent ces anglicismes, communément utilisés par les cryptographes francophones, qui équivalent respectivement à « texte clair » et « texte chiffré »). Un système de chiffrement est alors constitué de deux fonctions : le *chiffrement*, qui transforme un *plaintext* en un *ciphertext*, et le *déchiffrement*, qui calcule le *plaintext* correspondant à un *ciphertext* donné. La figure 1-1 illustre ces opérations, où le chiffrement correspond à l'opération notée **E** (pour *encryption*), qui reçoit comme entrées un *plaintext P*, une clé *K*, et retourne un *ciphertext C* en sortie. On note cette relation $C = \mathbf{E}(K, P)$. De même, cette figure représente le déchiffrement $\mathbf{D}(K, C)$.

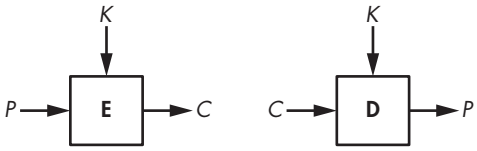

Figure 1-1 – Chiffrement et déchiffrement.

Note : Un *ciphertext* est souvent de la même taille que le *plaintext*, mais peut également être plus long ; il ne peut en revanche être plus court.

2. SYSTÈMES DE CHIFFREMENT CLASSIQUES

Les systèmes de chiffrement classiques existaient avant les ordinateurs et opèrent pour cette raison sur des lettres plutôt que sur des bits (1 et 0). De tels cryptosystèmes sont bien plus simples que leurs cousins plus modernes tels que DES. En effet, dans la Rome antique, ou même lors de la Première Guerre mondiale, on ne pouvait utiliser la puissance d'un microprocesseur pour chiffrer un message, les opérations devaient être faisables à l'aide d'un crayon et de papier. Parmi les nombreux cryptosystèmes classiques, les plus illustres sont certainement ceux de César et de Vigenère.

2.1 Le chiffre de César

Le chiffre de César (*Caesar cipher*) est ainsi nommé car l'historien romain Suétone a rapporté que Jules César l'utilisait. Cette méthode chiffre un message en décalant chacune de ses lettres de trois positions dans l'alphabet, en revenant à A si le décalage atteint Z. Par exemple, ZOO donnera CRR, et FDHDVU se déchiffrera en CAESAR, comme illustré sur la figure 1-2. La valeur 3 n'a rien de particulier ; elle est simplement plus facile à calculer de tête que 11 ou 23.

Le chiffre de César est très facile à casser : pour décrypter un texte chiffré donné, il suffit de reculer les lettres de trois positions pour retrouver le texte clair. Cela dit, le chiffre de César était peut-être suffisamment solide à l'époque de Crassus et de Cicéron. Étant donné qu'aucune clé secrète n'est utilisée (le décalage est toujours de trois positions), les utilisateurs devaient simplement supposer que les attaquants étaient analphabètes ou trop peu instruits pour comprendre l'astuce — une hypothèse beaucoup moins réaliste aujourd'hui. À noter qu'en 2006 la police italienne a arrêté un chef de Cosa nostra après avoir décrypté des messages écrits sur de petits bouts de papier, dits pizzini, qui étaient chiffrés à l'aide d'une variante de la méthode de César : ABC était chiffré en 456 au lieu de DEF, par exemple (D étant la quatrième lettre de l'alphabet, et ainsi de suite).

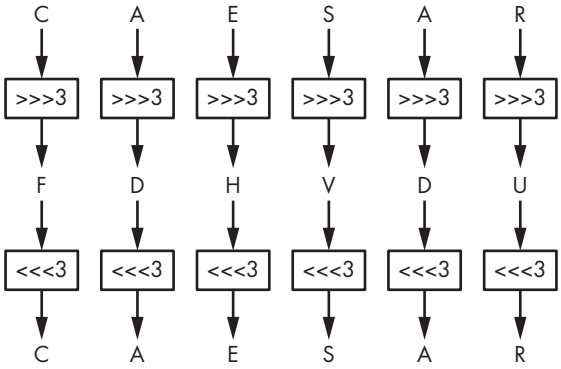

Figure 1-2 – Le chiffre de César.

Comment renforcer le chiffre de César ? On peut par exemple imaginer une variante qui utiliserait une valeur de décalage secrète au lieu de toujours 3 ; mais cela ne serait pas très utile, car un attaquant n'aurait qu'à essayer les 25 valeurs de décalage possibles jusqu'à ce que le message décrypté ait un sens.

2.2 Le chiffre de Vigenère

Il aura fallu attendre environ 1 500 ans pour voir une amélioration significative du chiffre de César, sous la forme du chiffre de Vigenère, créé au XVI^e^ siècle par un Italien nommé Giovan Battista Bellaso. Le nom « Vigenère » provient du Français Blaise de Vigenère qui a inventé un autre cryptosystème durant la même période ; mais en raison d'une mauvaise attribution historique, le nom de Vigenère est resté. Le chiffre de Vigenère a gagné en popularité et a été utilisé pendant la guerre de Sécession par les Sudistes, et pendant la Première Guerre mondiale par l'armée suisse, entre autres.

Le chiffre de Vigenère est similaire au chiffrement de César, à ceci près que les lettres ne sont pas décalées de trois positions, mais de valeurs distinctes déterminées par une *clé*, qui est une suite de lettres représentant chacune un nombre, selon leur position dans l'alphabet. Par exemple, si la clé est DUH, les lettres du *plaintext* sont décalées en utilisant les valeurs 3, 20, 7 car D est trois lettres après A, U est 20 lettres après A, et H est sept lettres après A.

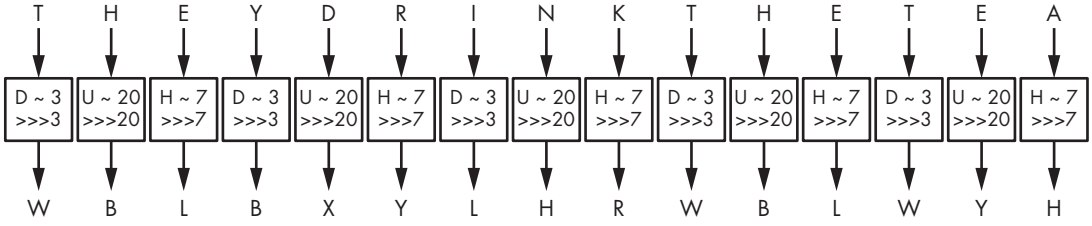

Figure 1-3 – Le chiffre de Vigenère.

Le schéma 3, 20, 7 se répète jusqu'à ce que vous ayez chiffré tout le *plaintext*. Par exemple, le mot CRYPTO sera chiffré en FLFSNV si DUH est la clé : C est déplacé de trois positions vers F, R est déplacé de 20 positions vers L, et ainsi de suite. La figure 1-3 illustre ce principe lors du chiffrement de la phrase THEY DRINK THE TEA.

Le chiffre de Vigenère est clairement plus sûr que le celui de César, mais il reste assez simple à casser. La première étape pour décrypter un message est de déterminer la longueur de la clé. Prenons l'exemple de la figure 1-3, où THEY DRINK THE TEA est chiffré en WBLBXYLHRWBLWYH avec la clé DUH. (Les espaces sont généralement supprimés pour masquer les limites des mots.) On remarque que dans WBLBXYLHRWBLWYH, le groupe de trois lettres WBL apparaît deux fois à des intervalles de neuf lettres. Cela suggère que le même mot de trois lettres a été chiffré en utilisant les mêmes valeurs de décalage, produisant WBL à chaque fois. Un cryptanalyste peut alors en déduire que la longueur de la clé est soit neuf, soit une valeur qui divise neuf (c'est-à-dire trois). De plus, si l'on sait que le texte est une phrase en anglais, on peut deviner que ce mot répété de trois lettres est THE et donc déterminer DUH comme potentielle clé de chiffrement.

La seconde étape pour casser le chiffre de Vigenère est l'*analyse de fréquence*, qui consiste à déterminer la clé réelle en exploitant la distribution non uniforme des lettres dans une langue. Par exemple, en français, E est la lettre la plus commune. Donc, si vous observez que X est la lettre la plus commune parmi les lettres chiffrées par une même valeur de clé, alors la lettre la plus probable du texte en clair à cette position est probablement E.

Malgré sa relative faiblesse, le chiffre de Vigenère a pu être assez solide pour protéger la confidentialité de messages. En effet, comme l'attaque que nous venons de décrire nécessite des messages d'au moins quelques phrases, elle ne fonctionne pas quand peu de messages courts sont chiffrés. De plus, à l'époque où Vigenère était utilisé, la plupart des messages n'avaient besoin d'être confidentiels que pendant de courtes périodes, de sorte qu'il importait peu que les messages chiffrés soient finalement décryptés par l'ennemi. (Le cryptographe du XIX^e^ siècle Auguste Kerckhoffs estimait qu'alors, en temps de guerre, la plupart des messages ne devaient rester confidentiels que pendant trois à quatre heures.)

3. COMMENT FONCTIONNE LE CHIFFREMENT

En se basant sur le chiffrement simpliste des techniques de César et de Vigenère, on peut essayer d'abstraire le fonctionnement d'un schéma de chiffrement, tout d'abord en identifiant ses deux composants principaux : une permutation et un mode opératoire.

Une *permutation* est une fonction qui transforme un élément (en cryptographie, une lettre ou un groupe de bits) de telle façon que chaque élément transformé ait un inverse unique. C'est par exemple le cas du décalage de trois lettres dans le chiffrement de César : pour inverser la transformation et retrouver la lettre originale, il suffit de décaler de trois lettres dans le sens inverse.

Un *mode opératoire* est un algorithme qui utilise une permutation pour traiter des messages de taille arbitraire. Le mode du chiffre de César est trivial : il répète simplement la même permutation pour chaque lettre, mais comme vous l'avez vu, le chiffre de Vigenère a un mode plus complexe où des permutations différentes sont appliquées à des lettres à différentes positions.

Dans les sections suivantes, je traite le fonctionnement et les propriétés de sécurité des permutations et modes opératoires. Je me repose sur ces deux composants pour démontrer que les systèmes de chiffrement classiques sont condamnés à être cassables, contrairement aux algorithmes modernes exécutés par des ordinateurs.

3.1 La permutation

La plupart des méthodes de chiffrement classiques fonctionnent en remplaçant chaque lettre du message par une autre lettre — autrement dit, en effectuant une *substitution*. Dans les chiffres de César et de Vigenère, la substitution est un déplacement de l'alphabet, typiquement les 26 lettres de A à Z. Toutefois l'alphabet peut varier et être plus général. Au lieu de l'alphabet latin, il peut s'agir de l'alphabet arabe ; au lieu de lettres, il peut s'agir de mots, de chiffres, d'idéogrammes, d'emojis, par exemple. Les éléments doivent juste pouvoir être ordonnés. Ils doivent aussi avoir une représentation, ou un encodage par ordinateur, mais ce dernier point ne concerne pas directement la sécurité.

La substitution d'un cryptosystème ne peut cependant pas être n'importe quelle substitution. Il doit s'agir d'une permutation, c'est-à-dire d'un réarrangement des lettres de A à Z de telle façon que chaque lettre ait un inverse unique. Par exemple, une substitution qui transforme les lettres A, B, C et D, respectivement en C, A, D et B est une permutation, car chaque lettre se transforme en une autre lettre unique. Mais une substitution qui transforme A, B, C, D en D, A, A, C n'est pas une permutation, car B et C se transforment tous les deux en A. En observant la lettre A, on ne peut donc pas déterminer avec certitude son antécédent ; avec une permutation, chaque lettre a exactement un inverse.

Cependant, toutes les permutations ne sont pas fiables cryptographiquement. Pour s'assurer de sa fiabilité, la permutation d'un chiffrement doit répondre à trois critères :

1. **La permutation doit être déterminée par la clé**, afin que la permutation soit secrète tant que la clé est secrète. Dans le chiffre de Vigenère, si l'on ne connaît pas la clé, on ne sait pas laquelle des 26 permutations a été utilisée

pour chiffrer une lettre donnée, et l'on ne peut alors pas facilement décrypter les messages.

2. **Des clés différentes doivent donner lieu à des permutations différentes**. Le cas échéant, il est plus facile de décrypter sans connaître la clé : si différentes clés correspondent à des permutations identiques, donc à une même transformation *plaintext* en *ciphertext*, on parle de *clés équivalentes*. Cela implique qu'il y a moins de permutations réalisables que de clés, et donc moins de clés à essayer si l'on cherche à décrypter par recherche exhaustive (ou *brute force*). Dans le chiffrement de Vigenère, chaque lettre de la clé détermine une substitution ; il y a 26 lettres distinctes, et autant de permutations.
3. **La permutation doit avoir l'air aléatoire**. Il ne doit pas y avoir de structure ou de relation particulière entre le *plaintext* et *ciphertext*, car cela rendrait une permutation en partie prévisible pour un attaquant, et donc moins sûre. Par exemple, la substitution du chiffrement de Vigenère est assez prévisible : si l'on sait que A est chiffré en F, on en déduit que le décalage est de 5 positions, et donc que B est chiffré en G, C en H, et ainsi de suite. Dans le cas d'une permutation choisie au hasard, le fait de savoir que A est transformé en F indiquerait seulement que B (ou toute autre lettre) n'est pas chiffré en F.

Nous appellerons une permutation qui satisfait à ces critères une permutation *sûre*.

Comme nous allons le voir, une permutation sûre est nécessaire mais pas suffisante pour construire un chiffrement sûr. Celui-ci aura également besoin d'un mode opératoire lui permettant de supporter de façon sûre des messages de n'importe quelle longueur.

3.2 Le mode opératoire

Supposons que nous ayons une permutation sécurisée qui transforme la lettre A en X, B en M, et N en L, par exemple. Le mot BANANA est donc chiffré en MXLXLX, où chaque occurrence de A est remplacée par un X, et chaque N par L. L'utilisation de la même permutation pour toutes les lettres du message révèle donc toute lettre en double. En analysant les doublons, vous n'apprendrez peut-être pas le message entier, mais vous apprendrez *quelque chose* sur le message. Dans notre exemple de MXLXLX, vous n'avez pas besoin de la clé pour deviner que le texte en clair comporte la même lettre aux trois positions X et une autre lettre identique aux deux positions L. Si vous savez en plus que, par exemple, le message est le nom d'un fruit, vous pouvez déterminer qu'il s'agit de BANANA plutôt que de CHERRY, LYCHEE, ou un autre fruit à six lettres.

Le mode opératoire (ou simplement le *mode*) d'un schéma de chiffrement peut éliminer l'exposition des lettres en double, en utilisant différentes permutations

pour différentes lettres. Le mode du chiffre de Vigenère répond partiellement à ce problème : si la clé a une longueur de N lettres, alors N permutations différentes seront utilisées pour chaque séquence de N lettres consécutives. Cependant, cela peut encore révéler des doublons dans le *ciphertext*, car chaque Nième lettre du message utilisera la même permutation. C'est pourquoi l'analyse de fréquence fonctionne pour casser le chiffrement de Vigenère, comme nous l'avons vu.

On peut imaginer que l'analyse de fréquence sera mise en échec si le chiffre de Vigenère ne traite que les messages qui ont la même longueur que la clé. Mais dans ce cas, il y a un autre problème : la réutilisation de la même clé plusieurs fois expose les similitudes entre les messages. Par exemple, avec la clé KYN, les mots TIE et PIE sont chiffrés en DGR et ZGR, respectivement. Les deux mots se terminent par les deux mêmes lettres (GR), ce qui révèle que les deux messages en clair partagent également leurs deux dernières lettres. Trouver ces modèles ne devrait pas être possible avec un chiffrement sécurisé. Il faudrait alors changer de clé avec chaque message, et chaque clé devrait avoir la même taille que le message ; comment transmettre alors la clé de façon sûre ? Et pourquoi ne pas directement transmettre le message ? Nous avons donc un problème de gestion de clés (*key management*).

Pour construire un chiffrement sécurisé, vous devez combiner une permutation sûre avec un mode opératoire sûr. Idéalement, cette combinaison empêchera les attaquants d'apprendre quoi que ce soit sur un message autre que sa longueur.

3.3 Pourquoi les chiffres classiques sont cassables

Contre des adversaires munis d'ordinateurs, les méthodes de chiffrement classiques ne peuvent être sûres et sont condamnées à être cassables. En effet, les chiffres classiques se limitent à des opérations simples que l'on peut faire de tête ou sur un bout de papier, et qui n'exploitent pas la puissance de calcul d'un microprocesseur. Cette limitation rend leur chiffrement trivial à casser avec de simples programmes informatiques. Voyons maintenant la raison fondamentale pour laquelle cette simplicité limite leur sécurité dans le monde d'aujourd'hui.

Rappelez-vous qu'une permutation doit se comporter de façon aléatoire afin d'être sûre. Évidemment, la meilleure façon de paraître aléatoire est d'*être* aléatoire, c'est-à-dire de sélectionner chaque permutation au hasard parmi l'ensemble de toutes les permutations, et il existe de nombreuses permutations parmi lesquelles choisir. Dans le cas de l'alphabet latin à 26 lettres, il existe environ 2^{88} permutations :

$$26! = 403\,291\,461\,126\,605\,635\,584\,000\,000 \approx 2^{88}$$

Ici, le point d'exclamation « ! » désigne la factorielle d'un nombre, définie ainsi :

$$n! = n \times (n - 1) \times (n - 2) \times \ldots \times 3 \times 2$$

Pour comprendre pourquoi nous obtenons ce nombre gigantesque ($2^{88} \approx 10^{26}$), comptez les permutations comme des listes de lettres réordonnées : il y a 26 choix pour la première lettre possible, puis 25 choix pour la deuxième, 24 pour la troisième, et ainsi de suite. Ce nombre est énorme : il est du même ordre de grandeur que le nombre d'atomes dans le corps humain. Mais les chiffrements classiques ne peuvent utiliser qu'une petite fraction de ces permutations, à savoir celles qui ne nécessitent que des opérations simples (comme les décalages) et qui ont une description succincte, comme un algorithme élémentaire et/ou une petite table de correspondance (*lookup table*). Le problème est qu'une permutation obéissant à ces deux limitations ne peut être cryptographiquement sûre.

Une façon d'obtenir une permutation sûre et algorithmiquement simple consisterait à choisir une permutation aléatoire en tirant une séquence de lettres au hasard (sans doublons), en la représentant sous la forme d'un tableau de 25 lettres (suffisant pour représenter une permutation de 26 lettres, la 26e étant manquante), et en l'appliquant en recherchant des lettres dans ce tableau. Mais alors vous n'auriez pas une description courte. Par exemple, il faudrait 250 lettres pour décrire 10 permutations différentes, plutôt que les 10 lettres utilisées dans le chiffrement de Vigenère. Le secret nécessaire pour (dé)chiffrer risquerait alors d'être plus long que le message, ce qui est absurde.

On peut toutefois créer des permutations sûres ayant une description courte. Au lieu de simplement déplacer l'alphabet, on utilisera une combinaison d'opérations telles que l'addition, la multiplication, les opérateurs logiques, etc. C'est ainsi que fonctionnent les chiffrements modernes : avec une clé de seulement 128 ou 256 bits, ils effectuent (au moins) des centaines d'opérations arithmétiques pour chiffrer une seule lettre. Ce processus est rapide sur un ordinateur capable d'effectuer des milliards d'opérations binaires par seconde, mais il faudrait des heures pour le faire à la main.

4. LE CHIFFREMENT PARFAIT : LE *ONE-TIME PAD*

Nous avons vu qu'un chiffre classique ne peut être sûr à moins que sa clé (en tant que « chose secrète à connaître pour (dé)chiffrer ») ne soit gigantesque, ce qui est rarement réaliste en pratique. La méthode du *one-time pad* (ou « masque jetable ») est une solution reposant sur un système simpliste, avec une longue clé, mais qui reste le plus sûr en garantissant le *secret parfait* : même si un attaquant dispose d'une puissance de calcul illimitée, il lui est impossible d'apprendre quoi que ce soit sur le texte en clair à partir du texte chiffré, à l'exception de sa longueur.

Dans les prochaines sections, je vous montrerai comment fonctionne le *one-time pad*, puis je vous présenterai une ébauche de sa preuve de sécurité.

4.1 Chiffrer et déchiffrer avec le *one-time pad*

Le *one-time pad* transforme un *plaintext P*, en utilisant une clé *K* de même longueur que *P*, et produit un *ciphertext C* défini ainsi :

$$C = P \oplus K$$

Où *C*, *P*, et *K* sont des chaînes de bits de même taille, et $\oplus$ est l'opération logique de OU exclusif (XOR), qui transforme les bits selon la règle suivante :

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0.$$

Note : Je décris le *one-time pad* dans sa forme habituelle, opérant sur des bits 1 et 0, mais il peut être adapté à tout autre alphabet. Avec les lettres latines, par exemple, on obtiendrait une variante du chiffre de César, dont la valeur de décalage serait aléatoire pour chaque lettre.

Avec le *one-time pad*, l'opération de déchiffrement est identique au chiffrement, c'est juste un XOR : $P = C \oplus K$. On peut vérifier qu'on obtient bien le *P* de départ, en remplaçant *C* par sa valeur calculée plus haut :

$$C \oplus K = (P \oplus K) \oplus K = P \oplus (K \oplus K) = P$$

Nous avons simplement appliqué la propriété d'associativité du XOR, et observé que $K \oplus K$ est toujours égal à une chaîne de zéros. Chiffrer et déchiffrer sont donc encore plus simples qu'avec le chiffre de César.

Par exemple, si $P = 01101101$ et $K = 10110100$, alors on obtiendra :

$$C = P \oplus K = 01101101 \oplus 10110100 = 11011001$$

Pour déchiffrer, on retrouve *P* en calculant :

$$P = C \oplus K = 11011001 \oplus 10110100 = 01101101$$

4.2 De l'importance du « *one-time* »

Comme son nom l'indique, la clé utilisée comme masque par le *one-time pad* ne doit être utilisée qu'*une seule fois*. Le cas échéant, si une même clé K est utilisée pour chiffrer deux messages distincts P_1 et P_2, respectivement en C_1 et C_2, alors un adversaire ayant observé ces deux *ciphertexts* pourra calculer :

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2 \oplus (K \oplus K) = P_1 \oplus P_2$$

L'adversaire pourra donc découvrir la différence entre P_1 et P_2 selon XOR, une information qui ne devrait pas être exposée. Concrètement, on saura à quelles

positions les deux *plaintexts* ont des bits identiques. De surcroît, si l'un des deux messages est connu ou deviné, alors le second message sera directement révélé — il suffirait de calculer $C_1 \oplus C_2 \oplus P_1$ pour déterminer P_2 si P_1 est connu.

Cependant, le *one-time pad* n'est clairement pas pratique à utiliser, car il nécessite une clé aussi longue que le *plaintext*, ainsi qu'une nouvelle clé aléatoire pour chaque nouveau message chiffré. Pour chiffrer un disque dur d'un téraoctet, il faudrait un autre disque d'un téraoctet pour stocker la clé ! Néanmoins, le *one-time pad* a souvent été utilisé dans le monde réel, par exemple par le Special Operations Executive britannique pendant la Seconde Guerre mondiale, par les espions soviétiques du KGB ou par la NSA américaine, et reste à ce jour utilisé dans certains contextes spécifiques. (Il y a quelques années, j'ai entendu l'histoire de banques suisses qui ne parvenaient pas à se mettre d'accord sur un algorithme de chiffrement fiable, et qui ont fini par utiliser des *one-time pads* — d'un point de vue cryptographique, je n'approuve pas.)

4.3 Pourquoi le *one-time pad* est-il sûr ?

Bien que le *one-time pad* ne soit pas très pratique, il est important de comprendre l'essence de sa sécurité. Dans les années 1940, le mathématicien américain Claude Shannon a prouvé que la clé d'un *one-time pad* doit être au moins aussi longue que le message afin d'obtenir une sécurité parfaite, en termes de confidentialité du message. L'idée de la preuve est assez simple : on suppose que l'attaquant a un pouvoir de calcul illimité, et qu'il peut donc essayer toutes les clés une à une.

◆ *Objectif de sécurité*

Le chiffrement doit donc être conçu tel que l'attaquant ne puisse pas identifier le « bon » message, et plus généralement tel qu'il n'ait aucun moyen d'exclure un des messages possibles. En d'autres termes, si le *ciphertext* est de 128 bits, alors toutes les valeurs de 128 bits devraient être des messages potentiels, du point de vue de l'attaquant — intuitivement, aucune information ne devrait *fuiter* depuis le message.

◆ *Intuition de la preuve*

L'idée de la preuve de *sécurité absolue* du *one-time pad* est la suivante : si la clé K est aléatoire — et donc totalement inconnue de l'attaquant — alors le *ciphertext* $C = P \oplus K$ sera aussi une valeur aléatoire du point de vue de l'attaquant. En effet, le XOR entre une chaîne de bits aléatoire et toute chaîne fixe donne une chaîne aléatoire.

Pour s'en convaincre, considérons la probabilité d'obtenir 0 comme premier bit de K, une chaîne aléatoire (à savoir, une probabilité de 1/2). Une fois XORé