Sériousse Cryptographie

Chapitre 1 : Le Chiffrement

v20221016

Sériousse Cryptographie est une traduction de l'ouvrage **Serious Cryptography**, de Jean-Philippe Aumasson, publié en 2017 par No Starch Press, voir https://nostarch.com/seriouscrypto. Le présent chapitre est publié sur le site de l'auteur : https://www.aumasson.jp/.

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 ».

Les Bases

Quand 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, équivalent 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 = **E**(*K*, *P*). De même, cette figure représente le déchiffrement **D**(*K*, *C*).

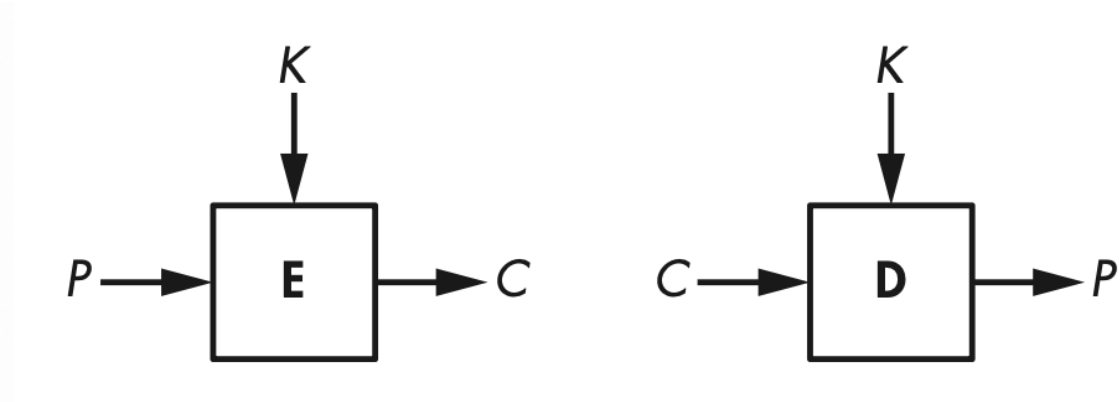

Figure 1-1 : Chiffrement et déchiffrement.

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.

Systèmes de Chiffrement Classiques

Les systèmes de chiffrement classiques prédatent les ordinateurs, et pour cette raison opèrent 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 processeur électronique pour chiffrer un message, les opérations devaient être faisable 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.

Le Chiffre de César

Le *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 3 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 boss de la 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).

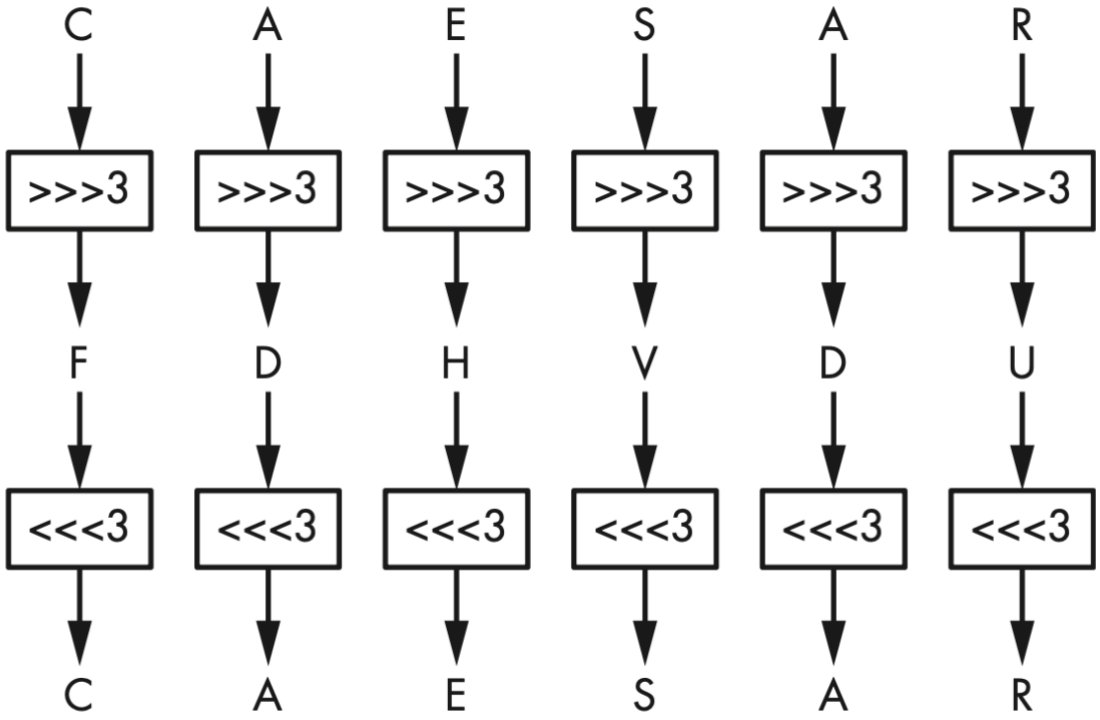

Figure 1-2 : Le chiffre de César.

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.

Le Chiffre de Vigenère

Il aura fallu attendre environ 1500 ans pour voir une amélioration significative du chiffre de César, sous la forme du chiffre de Vigenère, créé au XVIème 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 civile américaine par les forces confédérées, et pendant la première guerre mondiale par l'armée suisse, entre autres.

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

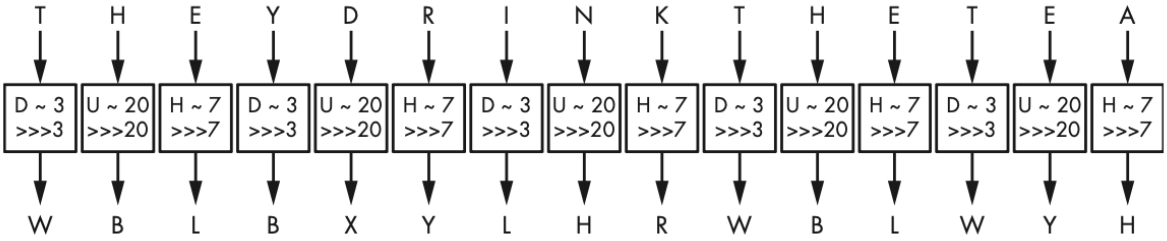

Figure 1-3 : Le chiffre de Vigenère.

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 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 langues. 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 déchiffrés par l'ennemi. (Le cryptographe du XIXème siècle Auguste Kerckhoffs estimait qu'alors, en temps de guerre, la plupart des messages ne devaient être que pendant trois à quatre heures).

Comment Fonctionne le Chiffrement

En se basant sur le chiffrement simpliste des techniques 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) tel 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 inverse 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 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 couvre le fonctionnement et les propriétés sécuritaires des permutations et modes opératoires. Je me repose sur ces deux composants pour démontrer que les systèmes de chiffrement classiques sont condamnés à cassables, contrairement aux algorithmes modernes exécutés par des ordinateurs.

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 tel 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.

Mais toutes les permutations ne sont pas fiables cryptographiquement. Pour être sûre, la permutation d'un chiffrement doit répondre à trois critères :

- **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 on ne connaît pas la clé, on ne sait pas laquelle des 26 permutations a été utilisée pour chiffer une lettre donnée, et on ne peut alors pas facilement décrypter les messages.
- **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 a 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 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.
- **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 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.

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 donner exposer 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 alors transmettre la clé de façon sûre ? Et pourquoi ne pas directement transmettre le message ? Nous avons donc un problème de *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.

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 cassables. En effet, les chiffres classiques se limitent à des opérations simples que l'on peut faire dans sa tête ou sur un bout de papier, et qui n'exploitent pas la puissance de calcul d'un processeur électronique. 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 basique et/ou une petite *look-up 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), et 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.

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 n'est rarement réaliste en pratique. La méthode du one-time pad (ou « masque jetable ») est un tel système : simpliste, longue clé, mais sûr. En fait, le plus sûr, et qui garantit 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 sa sécurité.

Chiffrer et Déchiffrer Avec le One-Time Pad

Le one-time pad transforme un plaintext *P*, and 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 : $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, et $1 \oplus 1 = 0$.

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 :

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

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 K est utilisée pour chiffrer deux messages distincts P_1 et P_2, respectivement en C_1 et C_2, alors un adversaire ayant observés ces deux ciphertexts pourra calculer

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

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 position les deux plaintexts ont des bits identiques. De surcroît, si 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 été déjà utilisé dans le monde réel, par exemple par le Special Operations Executive britannique pendant la Seconde Guerre mondiale, par les espions russes du KGB, 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 e chiffrement fiable, et qui ont fini par utiliser des one-time pads—je n'approuve pas, d'un point de vue cryptographique).

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 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 message 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* sur 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 bit 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 avec le premier bit de *P*, quelle est la probabilité d'obtenir 0 comme premier bit de $C = P \oplus K$? Exact, 1/2 à nouveau ! Le même argument peut être itéré sur tous les bits de K, quelle que soit sa longueur. Le ciphertext *C* apparaît donc aléatoire pour un attaquant qui ne connaît pas *K*. Il lui est donc littéralement impossible d'apprendre quoi que ce soit sur *P* à partir de *C*, en raison du masquage aléatoire de *K*. Une puissance de calcul et une mémoire illimités n'aident aucunement.

Nous pouvons en conclure que la connaissance du texte chiffré ne donne aucune information sur le texte en clair, à l'exception de sa longueur. C'est essentiellement la définition moderne d'un chiffrement sûr.

Les Probabilités en Cryptographie

Une *probabilité* est un nombre qui exprime la chance qu'un certain événement se produise. C'est un nombre décimal compris entre 0 et 1, où 0 signifie « jamais » et 1 signifie « toujours ». Plus la probabilité est élevée, plus la chance est grande. On trouve de nombreuses explications sur les probabilités, souvent en termes de boules de deux couleurs dans un sac et de la probabilité de choisir une boule de l'une ou l'autre couleur.

La cryptographie utilise souvent les probabilités pour mesurer les chances de succès d'une attaque, en comptant

1. Le nombre d'événements réussis (par exemple, l'événement "trouver la clé secrète correcte"), et
2. Le nombre total d'événements possibles (par exemple, le nombre total de clés est de 2^n si nous traitons des clés de *n* bits).

Dans cet exemple, la probabilité qu'une clé choisie au hasard soit la bonne est de $1/2^n$, soit le nombre d'événements recherchés (trouver 1 clé secrète) divisé le compte des événements possibles (2^n clés possibles). Le nombre $1/2^n$ est négligeable pour les longueurs de clé courantes telles que 128 et 256. Autrement dit, une probabilité de $1/2^{128}$ est tellement faible qu'en pratique elle est synonyme de « jamais ».

Si un évènement a une probabilité de *p*, alors la probabilité que l'événement ne se *produise pas* est $1 - p$. La probabilité d'obtenir une mauvaise clé dans notre exemple précédent est donc $1 - 1/2^n$, un nombre très proche de 1, ce qui signifie une quasi-certitude quand *n* est suffisamment grand.

Sur la Taille de la Clé

Par exemple, pour un ciphertext donné de 128 bits (et donc un plaintext sous-jacent de 128 bits), il existe 2^{128} plaintexts possibles, du point de vue de l'attaquant qui ne connaît pas la clé. S'il y avait moins de 2^{128} clés possibles, l'attaquant pourrait exclure certains de ces plaintexts possibles, de la façon suivante : Il n'aurait qu'à essayer toutes les clés possibles et collecter les plaintexts obtenus pour chaque clé—rappelez-vous que l'on suppose, dans le case de cette analyse du one-time pad, que l'attaquant a une puissance de calcul illimitée.

Si la clé ne comporte que 64 bits, par exemple, l'attaquant peut déterminer les 2^{64} plaintexts possibles et exclure la grande majorité des chaînes de 128 bits. L'attaquant n'apprendrait alors pas quel est *le* plaintext, mais il apprendrait ce que plaintext *n'est pas*, ce qui rend le secret du cryptage imparfait, car ceci constitue une *information* sur le message. Pour avoir une idée de l'ordre de grandeur de ces nombres, comparez

$$2^{64} = 18\ 446\ 744\ 073\ 709\ 551\ 616 \approx 10^{19}$$

et

$$2^{128} = 340\ 282\ 366\ 920\ 938\ 463\ 463\ 374\ 607\ 431\ 768\ 211\ 456 \approx 10^{38}$$

En résumé, la clé doit être aussi longue que le plaintext si on veut obtenir une sécurité parfaite. Mais c'est en pratique inutilisable (il faudrait, pour chaque message envoyer, transmettre en premier lieu, de façon sûre, une clé de la même taille). Dans la suite, nous allons voir des méthode de chiffrement avec une clé plus courte, et aussi sûres en pratique.

Définir la Sécurité du Chiffrement

Nous avons vu que les méthodes de chiffrements classiques n'étaient pas sûres, et qu'un chiffrement parfaitement sûr comme le one-time pad n'était pas utilisable en pratique. Nous devons donc faire certaines concessions si nous voulons chiffrer de façon sécurisée dans nos applications. Mais que signifie réellement « sécurisée » en dehors de l'informelle définition « on ne doit pas pouvoir trouver le message sans la clé » ?

Intuitivement, un chiffrement est sûr si, même en connaissant un grand nombre de paires

$$[P, C] = [P, C = \mathbf{E}(K, P)], \text{ pour une clé } K \text{ fixe et inconnue,}$$

alors *rien ne peut être appris* sur le plaintext P correspondant à un nouveau $C = \mathbf{E}(K, P)$. Cela amène de nouvelles questions :

- Comment un attaquant obtiendrait-il de telles paires $[P, C]$? Qu'entendons-nous par « un grand nombre » ? Tout cela est défini par les *modèles d'attaque*, les hypothèses sur ce que l'attaquant peut et ne peut pas faire.
- Qu'est-ce qui pourrait être « appris » sur un plaintext, et comment définir cette notion mathématiquement ? Ceci est défini par les *objectifs de sécurité*, les descriptions de ce qui est considéré comme une attaque réussie.

Les modèles d'attaque et les objectifs de sécurité doivent aller de pair ; on ne peut pas juste prétendre qu'un système est sûr sans décrire contre quel type d'attaque. Une *notion de sécurité* est donc la combinaison d'un certain objectif de sécurité avec un certain modèle d'attaque. Nous

dirons qu'un chiffrement atteint une certaine notion de sécurité si tout attaquant obéissant aux contraintes d'un modèle donné ne peut atteindre l'objectif de sécurité.

Les Modèles d'Attaques

Un modèle d'attaque est un ensemble d'hypothèses sur ce qu'un attaquant pourrait et ne pourrait pas faire. Les buts d'un tel modèle sont les suivantes :

- Définir des exigences techniques précises pour les cryptographes qui créent des cryptosystèmes (et plus généralement, des *protocoles* cryptographiques), afin qu'ils sachent contre quel type d'attaques l'algorithme doit être protégé.
- Donner aux utilisateurs les conditions sous lesquelles il est sûr (ou pas) d'utiliser un cryptosystème donné.
- Donner aux cryptanalystes essayant de « casser » un cryptosystème une définition claire de « casser », et d'éviter les ambiguïtés sur ce qui constitue une attaque valide. Par exemple, si une technique d'attaque nécessite des moyens ou des informations allant au-delà de celles définies par le modèle, alors l'attaque n'est pas valide (et pouvait par exemple être connue des créateurs du système).

Les modèles d'attaques ne sont pas tenus de correspondre précisément à la réalité, d'autant plus qu'il n'existe pas une unique réalité—en termes de capacité d'attaques d'un cryptosystème—mais souvent autant de réalités que de cas d'utilisation. Par exemple, dans certains cas un attaquant pourra capturer quelques paires [plaintext, ciphertext] au hasard, alors que dans d'autres cas il n'aura que des ciphertext, etc. Le cryptographe doit donc se baser sur un modèle qui *capture* ces réalités, sachant qu'un modèle, dans tout domaine, reste une approximation simplifiée d'une réalité inévitablement complexe.

Comme l'a dit le statisticien George E. P. Box, « tous les modèles sont faux ; la question pertinente est de savoir jusqu'à quel point ils doivent être faux pour ne pas être utiles ».

Pour être utiles en cryptographie, les modèles d'attaque doivent au moins englober ce que des attaquants réels pourraient faire pour attaquer un cryptosystème. Ce n'est pas grave, et même une bonne chose, si un modèle surestime les capacités des attaquant, car cela permet d'anticiper les futures techniques d'attaque—seuls les cryptographes paranoïaques survivent.

Un modèle sous-estimant le pouvoir des attaquants est un mauvais modèle, et donne une fausse confiance dans un cryptosystème en le faisant paraître sûr en théorie alors qu'il ne l'est pas en pratique

Le Principe de Kerckhoffs

Une hypothèse commune à tous les modèles est le principe dit de Kerckhoffs, qui stipule que la sécurité d'un système de chiffrement doit uniquement reposer sur le secret de la clé, et non sur le secret de la méthode. Cela peut sembler évident aujourd'hui, où les algorithmes de chiffrement et les protocoles que nous utilisons sont publiquement spécifiés et standardisés, et où leurs implémentations sont souvent open-source (mais pas toujours). Historiquement, cependant, le linguiste néerlandais Auguste Kerckhoffs faisait référence aux machines de chiffrement militaires spécifiquement conçues pour une armée ou une division donnée.
Citons l'essai de Kerckhoffs daté de 1883, « La Cryptographie Militaire », où il énumère six exigences d'un système de chiffrement militaire : « Le système ne doit pas exiger le secret et peut être volé par l'ennemi sans causer de problèmes ».

Les Modèles Black-Box

Découvrons maintenant quelques modèles d'attaque, exprimés en termes de ce que l'attaquant peut observer et des requêtes qu'il peut faire au système chiffrement. Dans notre cas, une *requête* est l'opération qui envoie une ou plusieurs valeurs d'entrée (*input*) à un système, et en obtient la ou les valeurs de retour (*output*), sans exposer les détails du système. Une requête de chiffrement, par exemple, prend un texte en clair et renvoie un texte chiffré correspondant, sans révéler la clé secrète.

Cette opacité du système lors d'une requête, cachant notamment la clé secrète, est à l'origine du terme *black-box* (« boîte noire »), où l'attaquant a juste accès aux entrées et aux sorties du système. S'il pouvait voir tout ce qui se passe dans la boîte—soit, le calcul du (dé)chiffrement, dans notre case—il pourrait simplement en extraire la clé. On parle en effet de *modèles black-box*.

Une fois de plus, ce modèle de requête et réponse est une abstraction visant à capturer les capacités *éventuelles* d'un attaquant, bien que dans la plupart des cas réels on n'ait pas la possibilité de faire de telle requête, du moins simplement.

Voici un exemple où les requêtes de déchiffrement sont possibles : Certains microprocesseurs de cartes à puce protègent de manière sécurisée les éléments internes d'un chiffrement ainsi que ses clés, mais acceptent de déchiffrer n'importe quel ciphertext qu'on leur envoie (par exemple dans certains systèmes de TV payante). Dans un tel cas, l'attaquant peut recevoir les plaintexts d'un nombre arbitraire de ciphertexts, ce qu'il pourrait exploiter pour déterminer la clé.

Il existe quatre modèles d'attaque principaux pour le chiffrement. Je les décris ici du plus faible au plus fort, en conservant la terminologie établie en anglais :

- **Ciphertext-only attackers (COA),** où l'attaquant observe les ciphertexts sans connaître les plaintexts associés, et n'a pas la capacité d'influencer leur choix. Il reçoit des $C = \mathbf{E}(K, P)$ sans connaître P, ni évidemment K. Ce type d'attaquant est totalement passif et ne peut faire aucune requête.
- **Known-plaintext attackers (KPA***),* où l'attaquant observe des ciphertexts et connaît les plaintexts associés, mais sans pouvoir les choisir. Il reçoit des paires $[P, C]$, avec $C = \mathbf{E}(K, P)$. En théorie, les P sont aléatoires, mais en pratique ils le sont rarement totalement. Encore une fois, KPA est un modèle passif.
- **Chosen-plaintext attackers (CPA),** où l'attaquant peut faire des requête de chiffrement pour les plaintexts de son choix, et obtenir les ciphertexts correspondants. Il reçoit des $C = \mathbf{E}(K, P)$ pour les P de son choix. Ce modèle capture les situations où les attaquants peut choisir tout ou partie des plaintexts. Contrairement aux modèles COA et KPA, qui sont passifs, un attaquant *CPA est actif*, car il peut influencer le processus de chiffrement.
- **Chosen-ciphertext attackers (CCA),** le modèle le plus fort, où l'attaquant peut à la fois chiffrer et déchiffrer. Un tel attaquant peut faire tout ce qu'un CPA peut faire, et également envoyer des ciphertexts C afin de recevoir le résultat de $\mathbf{D}(K, C)$, le déchiffrement de C avec la clé K, dont le résultat est soit un plaintext P, soit une erreur (au cas où le C n'est pas un ciphertext valide).

Parmi ces modèles, le CCA peut sembler absurde de prime abord—si un attaquant peut déchiffrer, alors que veut-il de plus ? Cependant, le modèle CCA vise à englober des situations où les attaquant peuvent exercer une certaine influence sur le texte chiffré, et ensuite découvrir tout ou partie du plaintext correspondant, ce qui est moins improbable que la capacité de

complètement déchiffrer n'importe quel ciphertext choisi. De plus, pour compromettre un système de sécurité utilisant le chiffrement, il n'est pas toujours suffisant de pouvoir déchiffrer. Par exemple, dans le cas du piratage de vidéo, il est plus simple de partager une clé de 64 octets que les gigaoctets de données déchiffrées. Dans ce cas, le pouvoir de CCA est insuffisant.

NOTE : Dans ces modèles d'attaque, les ciphertexts observés ainsi que ceux transmis dans des requêtes ne sont pas « gratuits ». En général, un ciphertext provient du calcul de la fonction de chiffrement. Cela signifie que la génération de 2^N pairs [*P*, *C*] par le biais de requêtes dans un modèle CPA ou CCA coûte à peu près autant en calcul qu'essayer 2^N clés de déchiffrement.

Les Modèles Gray-Box

Le modèle gray-box (« boîte grise ») est une version souvent plus réaliste du modèle black-box, où l'attaquant a accès à *l'implémentation physique* d'un cryptosystème. Notez que l'on parle de l'appareil réel, par exemple le circuit imprimé, la carte à puce, le HSM (*Hardware Security Module*), ou autre module cryptographique. L'attaquant peut aussi connaître l'implémentation logique (typiquement, via le code source), une hypothèse que l'on fait aussi pour le modèle black-box.

Un attaquant gray-box peut donc employer divers moyens pour exploiter les composants physiques du système, afin d'en extraire des informations ou en modifier le comportement—la confidentialité et l'intégrité de l'implémentation n'est plus garantie. Cela rend les modèles gray-box plus réalistes que les modèles black-box pour les applications telles que les systèmes embarqués (aussi dits « IoT »), les cartes à puce, et même les systèmes cloud où on utilise souvent des machines virtuelles sur des infrastructures physiques contrôlées par d'autres organisations.

Il est toutefois souvent difficile de définir des modèles gray-box pertinents, car ils dépendent de propriétés de composants physiques d'un système, des propriétés rarement clairement définies ni formalisées ; on se trouve alors dans le domaine de l'ingénierie plutôt que celui, plus structuré, des mathématiques—à l'inverse de modèles black-box où on a une vue abstraite d'un système comme une fonction mathématique, avec ces entrées et sorties. Mais malgré cette complexité intrinsèque à la réalité, il est crucial de dépasser les limites de l'analyse théorique, mais si tout modèle gray-box ne sera jamais une représentation parfaite du vrai système.

Parmi les attaques prises en compte par les modèles gray-box, en plus de celles couvertes par une vue black-box, on distinguera :

- **Les attaques side-channel,** dites « par canal auxiliaire » en français. Un side channel[1] est une source d'information qui dépend de l'implémentation du cryptosystème, que ce soit en software ou hardware. Les attaquants side-channel observent ou mesurent les caractéristiques analogues d'une implémentation—au sens général de phénomène physique—mais n'en n'altèrent pas son intégrité ; ils sont *non-invasifs*. Certains side channels sont mesurable sans un accès physique direct à la plateforme, et sont potentiellement exploitable à distance. Par exemple, le temps d'exécution et le comportement du système *autour* de la cryptographie, comme les messages d'erreur, les valeurs de retour des fonctions, ainsi que le comportement d'autres programmes qui pourraient être exécutés en même temps sur la même plateforme, partageant les ressources de calcul et de mémoire. Parmi les side channels exigeant une proximité

[1] On écrira « side-channel » avec un tiret quand il est employé comme adjectif, mais « side channel » pour la forme nominal, où « side » est l'adjectif.

physique, les plus courants sont la consommation électrique, les émanations électromagnétiques, ou les ondes acoustiques.

- **Les attaques invasives**, qui sont une famille d'attaques plus puissantes que les attaques side channel, mais souvent plus complexes, et plus coûteuses, car elles nécessitent un équipement sophistiqué. Alors qu'on peut exécuter des attaques side-channel de base avec un PC standard, les attaques invasives nécessitent des outils tels qu'un microscope à haute résolution et un laboratoire chimique. Ces attaques englobent un ensemble de techniques et de procédures, allant de l'utilisation d'acide nitrique pour éliminer les couches superficielles d'un microprocesseur, à l'acquisition d'images avec un microscopique, en passant par le *reverse-engineering* de circuits imprimé et l'injection de fautes par laser.

Les Objectifs de Sécurité

J'ai défini de manière informelle l'objectif de sécurité du chiffrement comme « on ne doit pas pouvoir trouver le message sans la clé ». Plus généralement, on voudra empêcher un attaquant d'apprendre quoi que ce soit sur le comportement du chiffrement. Pour transformer cette idée en une définition rigoureuse, les cryptographes définissent deux objectifs de sécurité principaux, qui décrivent mathématiquement l'intuition de « chiffrement sécurisé » :

- **L'indistinguabilité (IND)** : On ne doit pas pouvoir différencier un ciphertext d'une chaîne de bits aléatoire. C'est en général illustré par le « jeu » suivant : un attaquant choisit deux plaintexts, ensuite un des deux est choisi au hasard et chiffré, et l'attaquant reçoit le ciphertext ; il doit ensuite deviner lequel des deux messages a été chiffré.
- **La non-malléabilité (NM)** : Donné un ciphertext $C_1 = \mathbf{E}(K, P_1)$, un attaquant ne doit pas pouvoir construire un autre ciphertext C_2 tel que $\mathbf{D}(K, C_2) = P_2$ avec P_2 un message ayant une relation particulière avec P_1. Par exemple, tel que $P_2 = P_1 \oplus 1$, ou $P_2 = P_1 \oplus X$ pour une valeur X fixe, ou encore $P_2 = f(P_1)$ pour une fonction f() simple. On remarquera que le one-time pad est trivialement malléable via un XOR avec le ciphertext.

Dans la suite, nous allons combiner ces objectifs de sécurité avec les modèles d'attaque discutés plus haut.

Les Notions de Sécurité

Un objectif de sécurité n'est rien sans des hypothèses sur le pouvoir d'un attaquant, c'est-à-dire, sans un modèle d'attaque. On suivra la convention établie en notant une notion de sécurité OBJECTIF-MODÈLE. Par exemple, IND-CPA désigne l'indistinguabilité contre des attaquants chosen-plaintext ; NM-CCA désigne la non-malléabilité contre des attaquants chosen-ciphertext, et ainsi de suite. Commençons par une notion essentielle :

Sécurité Sémantique et Chiffrement Probabiliste : IND-CPA

La notion de sécurité la plus importante est IND-CPA, également connue sous le nom équivalent de *sécurité sémantique*. Elle capture l'intuition selon laquelle les ciphertexts ne devraient pas divulguer d'information sur les plaintexts tant que la clé est secrète. Pour atteindre la sécurité IND-CPA, observons que le chiffrement doit retourner des ciphertexts différents s'il est appelé deux fois sur le même plaintext ; sinon, un attaquant qui observe deux ciphertexts identiques

pourrait en déduire que les plaintexts sous-jacents sont identique, ce qui contredit la définition selon laquelle les ciphertexts ne doivent révéler aucune information.

Une façon d'atteindre la sécurité IND-CPA est d'utiliser le *chiffrement probabiliste* (*randomized encryption*). Comme son nom l'indique, il randomise le processus de chiffrement et permet de retourner des ciphertexts différents lorsque le même texte en clair est chiffré plusieurs fois de suite. Le chiffrement peut alors être noté $C = \mathbf{E}(K, R, P)$, où R est une chaîne de bits aléatoires—c'est-à-dire, choisis aléatoirement à chaque appel à $\mathbf{E}()$, et donc différents à chaque fois, sauf avec une probabilité négligeable, si R est assez grand, par exemple 128 bits. Le déchiffrement reste déterministe, car si on a calculé $C = \mathbf{E}(K, R, P)$ alors $\mathbf{D}(K, R, C)$ doit toujours retourner P, quelle que soit la valeur de R.

Si on ne dispose pas de générateur aléatoire, et qu'on ne peut donc réaliser de chiffrement probabiliste, il reste possible d'atteindre la sécurité IND-CPA en utilisant un cryptosystème qui a juste besoin d'une valeur unique pour chaque chiffrement, une valeur qu'on appelle *nonce* (pour *number used only once*). Un compteur 1, 2, 3, etc. peut donc faire office de nonce. Par exemple, AES-CTR (le blockcipher AES en mode de chiffrement compteur, dit CTR) est IND-CPA si on lui fournit, en plus du message, un *nonce* unique. À la différence du chiffrement probabiliste, le nonce est *nécessaire pour déchiffrer*.

If you don 't have a pseudorandom generator, you may still achieve IND-CPA security by using an encryption scheme that only requires a *nonce* (or number used only once), rather than a random, unpredictable value. A nonce just needs to be unique for every new encryption call. A mere counter 1, 2, 3, etc. would do the trick. For example, AES-CTR (or the AES block cipher used in CTR mode, a.k.a. counter mode) is IND-CPA if its additional input, the nonce, is unique. Unlike randomized encryption, the nonce is *required to decrypt*.

Comment violer la sécurité IND-CPA si le chiffrement d'un même message donne toujours le même ciphertext ? Reprenons le jeu IND-CPA décrit dans la section « Objectifs de Sécurité », où l'attaquant doit choisir deux plaintext P_1 et P_2, reçoit le chiffrement d'un des deux, et il doit trouver lequel des deux messages a été chiffrés. Grâce au modèle CPA, chosen-plaintext, cet attaquant peut faire des requêtes de chiffrement et donc demander la valeur de $\mathbf{E}(K, P_1)$. Si le chiffrement est déterministe, alors P_1 donnera toujours le même ciphertext. Il devient donc trivial de gagner le jeu, en comparant la valeur de challenge avec le résultat de la requête $\mathbf{E}(K, P_1)$: si les valeurs sont identiques, alors P_1 a été choisi, et sinon, P_2.

NOTE : Avec le chiffrement probabiliste, les ciphertexts doivent être un peu plus longs que leur plaintexts respectifs, car à un plaintext donné correspondent plusieurs ciphertexts différents. Par exemple, s'il y a 2^{64} ciphertexts possible par plaintext (correspondant à chacun des 2^{64} valeurs aléatoires possibles pour R), alors les ciphertexts devront être au moins 64 bits plus longs que les plaintexts.

Réaliser la Sécurité Sémantique

Une des plus simples constructions sémantiquement sûres utilise un générateur aléatoire déterministe[2], ou *deterministic random bit generator* (DRBG), un algorithme qui retourne des bits ayant l'air aléatoire à partir d'une valeur secrète, souvent appelée *seed* (« graine »). Notre construction d'un système de chiffrement est la suivante :

---

[2] « Aléatoire déterministe » sonne comme un oxymore, car c'est une simplification : plus précisément, les bits générés (de façon déterministe, à partie d'une valeur imprévisible) *auront l'air* aléatoire, et sont aussi dits *pseudo-aléatoires*.

$$\mathbf{E}(K, R, P) = (\mathbf{DRBG}(K \parallel R) \oplus P, R)$$

où R est une valeur choisie aléatoirement pour chaque nouvel appel à **E**(), et où $K \parallel R$ désigne la concaténation de K et R, soit la chaîne de bits constituée de K suivie de R. Cette approche rappelle clairement le one-time pad : au lieu d'utiliser une clé secrète de la même taille que le message, on utilise un DRBG pour générer une chaîne pseudo-aléatoire aussi longue que le message, à partir seulement d'une courte valeur secrète.

Preuve de Sécurité

La preuve que le cryptosystème ci-dessus est IND-CPA est relativement simple, si nous supposons que le DRBG produit des bits indistinguable des « vrais » bits aléatoires. La preuve fonctionne par l'absurde : si le chiffrement n'était pas IND-CPA, alors cela signifie que l'on pourrait distinguer les ciphertexts des chaînes aléatoires, ce qui implique que l'on pourrait distinguer les valeurs de

$$\mathbf{DRBG}(K \parallel R) \oplus P$$

de bits purement aléatoires. Mais ceci implique que l'on pourrait distinguer $\mathbf{DRBG}(K \parallel R)$ de bits aléatoire, car dans notre modèle les plaintexts possibles sont connus, ce qui nous permettrais de calculer

$$(\mathbf{DRBG}(K \parallel R) \oplus P) \oplus P = \mathbf{DRBG}(K \parallel R)$$

Nous avons alors atteint une contradiction, car nous avons postulé que les résultats de $\mathbf{DRBG}(K \parallel R)$ ne pouvaient pas être distingués de bits aléatoires. Nous concluons que les ciphertexts ne peuvent pas être distingués des chaînes aléatoires, et donc que le chiffrement est bien IND-CPA.

NOTE : Comme exercice, essayer de déterminer si ce cryptosystème satisfait d'autres notions que IND-CPA. Est-il NM-CPA ? IND-CCA ? Vous trouverez des réponses dans la prochaine section.

Comparer les Notions de Sécurité

Nous avons appris que les modèles d'attaque tels que CPA et CCA sont combinés avec des objectifs de sécurité tels que NM et IND pour construire les notions de sécurité NM-CPA, NM-CCA, IND-CPA et IND-CCA. Y-a-t-il des liens entre ces notions ? Pouvons-nous prouver que satisfaire la notion X implique la notion Y ?

Certaines relations sont évidentes : IND-CCA implique IND-CPA, et NM-CCA implique NM-CPA, car tout ce qu'un attaquant CPA peut faire, un attaquant CCA peut le faire aussi. En d'autres termes, si on ne peut pas casser un chiffrement en faisant des requêtes chosen-plaintext *et* chosen-ciphertext, alors on ne pourra pas non plus le casser en effectuant uniquement des requêtes chosen-plaintext.

Une relation moins évidente est que IND-CPA n'implique pas NM-CPA. Pour comprendre cela, observons que la construction précédente, qui est IND-CPA, n'est pas NM-CPA. Pour simplificer, notons $X = \mathbf{DRBG}(K, R) \oplus P$ la première partie d'un ciphertext, soit

$$(X, R) = (\mathbf{DRBG}(K, R) \oplus P, R)$$

Étant donné un texte chiffré (X, R), on peut créer un second ciphertext $(X \oplus 1, R)$, qui est un texte chiffré valide de $P \oplus 1$, ce qui contredit la notion de non-malléabilité. Cette astuce fonctionne évidemment pour toute autre valeur que 1.

Mais la relation inverse existe : NM-CPA implique IND-CPA. L'intuition est que le chiffrement IND-CPA est comme mettre des objets dans un sac opaque : on ne les voit plus, mais on peut réarranger leurs positions dans le sac en le secouant le sac dans tous les sens. Le chiffrement NM-CPA ressemble davantage à un coffre-fort fixé à un mur: une fois des objets à l'intérieur, on ne peut plus interagir avec son contenu. Cette analogie ne fonctionne cependant pas pour IND-CCA et le NM-CCA, qui sont des notions fondamentalement équivalentes. Je vous épargne la preuve, qui est assez technique.

Il existe deux principaux types d'applications de cryptage. Le cryptage en transit protège les données envoyées d'une machine à une autre : les données sont cryptées avant d'être envoyées et décryptées après avoir été reçues, comme dans les connexions cryptées aux sites Web de commerce électronique. Le chiffrement au repos protège les données stockées sur un système d'information. Les données sont cryptées avant d'être écrites en mémoire et décryptées avant d'être lues. Les exemples incluent les systèmes de cryptage de disque sur les ordinateurs portables ainsi que le cryptage de machine virtuelle pour les instances virtuelles du cloud. Les notions de sécurité que nous avons vues s'appliquent aux deux types d'applications, mais la bonne notion à prendre en compte peut dépendre de l'application.

Deux Classes d'Application du Chiffrement

On peut distinguer deux classes d'applications principales du chiffrement. Le chiffrement *en transit* (*in-transit*), qui protège les données transmises d'un point à un autre (d'un navigateur à un serveur, d'un smartphone à un autre, etc.) : les données sont chiffrées avant d'être envoyées, et déchiffrer après être reçues, par un autre parti. Le chiffrement *au repos* (*at-rest*) protège les données stockées sur un système donné, typiquement une base de données ou un système de fichier, et sont en général déchiffrées par le même parti qui les a chiffrées. Les données sont chiffrées avant d'être écrite en mémoire, et sont déchiffrées pour être lues. Le chiffrement de risque ou de machine virtuelle est une forme de chiffrement au repos. Les notions de sécurité que nous avons vues s'appliquent à ces deux classes, mais la notion nécessaire dépend de l'application et du type d'attaques possibles.

Le Chiffrement Asymétrique

Jusqu'à présent, nous n'avons considéré que le chiffrement symétrique, où deux parties partagent une même clé. Dans le cas du *chiffrement asymétrique*, on détient chacun *une paire de clés* : une pour chiffrer, et l'autre pour déchiffrer. La clé de chiffrement est dite *clé publique* et est généralement considérée comme publiquement disponible pour quiconque souhaite vous envoyer des

messages chiffrés. La clé de déchiffrement, en revanche, doit rester secrète et est appelée *clé privée*, ou clé secrète.

La clé publique peut être calculée à partir de la clé privée, mais la clé privée ne peut bien sûr pas être calculée à partir de la clé publique. Autrement dit, il est facile de calculer dans un sens, mais pas dans l'autre. C'est là l'essence de la *cryptographie à clé publique* : des fonctions faciles à calculer dans un sens, mais pratiquement impossibles à inverser.

Les modèles d'attaque et les objectifs de sécurité pour le chiffrement asymétrique sont à peu près les mêmes que pour le cryptage symétrique, sauf que, comme la clé de chiffrement est publique, tout attaquant peut chiffrer n'importe quel message, sans avoir à faire de requêtes de chiffrement à un système tiers. Le modèle d'attaque par défaut est donc le chosen-plaintext (CPA).

Les méthodes de chiffrement symétrique et asymétrique sont les deux grandes familles de chiffrement, et sont généralement combinées pour construire des systèmes de communication sécurisées. Ils sont également utilisés pour former construire des systèmes plus sophistiqués, comme nous allons le voir.

Au-Delà du Chiffrement

Le chiffrement de base transforme les plaintexts en ciphertexts, et inversement, sans autre exigence que la confidentialité. Cependant, on a parfois besoin de plus que cela, qu'il s'agisse de notions de sécurité ou bien de fonctionnalités additionnelles. C'est pourquoi les cryptographes ont créé une myriades de variantes du chiffrement symétrique et asymétrique, inspirés par la variété de risques auxquels s'exposent les applications technologiques modernes. Certaines variantes sont bien comprises, fiables, et largement déployées, tandis que d'autres en sont encore expérimentales, avec des temps de calculs prohibitifs, ou parfois uniquement destinées à l'analyse théorique.

Le Chiffrement Authentifié

Le chiffrement authentifié ne se limite pas à créer un ciphertext, il retourne également un *tag d'authentification*, tel qu'illustré en Figure 1-4 : on notera **AE**(), pour *authenticated encryption*, le chiffrement authentifié, tel que **AE**(*K*, *P*) = (*C*, *T*), avec T le tag, une courte chaîne de bit (par exemple 128 bits) impossible à deviner si on ne connaît pas la clé.

Le déchiffrement prend en entrée une clé *K*, un ciphertext *C*, et un tag *T*, et retourne un plaintext *P* uniquement si *T* est valide pour cette paire plaintext–ciphertext. Le cas échéant, il retourne une erreur. À l'inverse, le déchiffrement basique (non-authentifié) pouvait accepter n'importe quel chaîne de bits comme ciphertext.

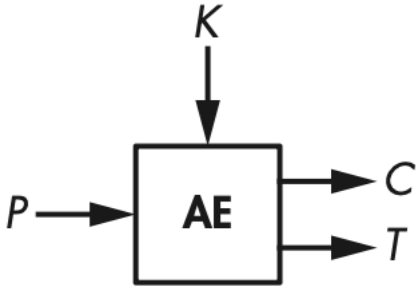

Figure 1-4 : Le chiffrement authentifié.

Le tag garantit l'intégrité du message, dans le sens que, si un attaquant essaye de modifier le ciphertext pour donner un plaintext différent, la nécessité d'avoir un tag valide l'en empêchera.

Le tag sert donc de preuve que le ciphertext reçu est identique à celui envoyé par un parti qui connaît la clé *K*. Lorsque *K* est partagée avec un seul autre système, le tag garantit que le message a été envoyé par ce système; c'est-à-dire qu'il *authentifie* implicitement l'expéditeur attendu comme le *créateur* réel du ciphertext.

NOTE : Je parle ci-dessus de « créateur » plutôt qu'expéditeur, car un attaquant peut parfois capturer des paires (*C*, *T*) envoyées par un système A à un système B à un moment donné, pour plus tard les renvoyer à B en se faisant passer pour A. C'est ce qu'on appelle un *replay* (ou « rejeu »), et peut être évité en liant le message à des données temporelles ou propres à la session de communication. Beaucoup de protocoles utilisent pour cela un compteur de message, tel que le compteur $i = 1, 2, 3, \ldots$ est incrémenté à $i + 1$ pour chaque message. Si le destinataire reçoit un message avec un compteur inférieur au dernier *i*, alors il en déduira qu'un ancien message a été renvoyé. Ceci permet également de détecter les messages perdus ou bloqués par un attaquant.

Une extension du chiffrement authentifié est le chiffrement authentifié avec données associées, ou *authenticated encryption with associated data* (AEAD). Celui-ci prend en entrée des données qui ne seront pas chiffrées, mais seront néanmoins prises en compte lors du calcul du tag. On aura alors

$$\mathbf{AEAD}(K, P, A) = (C, T),$$

Le déchiffrement recevra *C*, *A*, et *T* en entrée, avec donc *A* en clair. Une application typique de l'AEAD est la protection de datagrammes des protocoles dont l'en-tête (*header*) est en clair et le contenu (*payload*) est chiffrée. Dans de tels cas, au moins certaines données d'en-tête doivent rester en clair ; par exemple, les adresses de destination doivent pouvoir être lues afin d'acheminer les paquets à travers un réseau. Pour en savoir plus sur le chiffrement authentifié, passez au Chapitre 8.

Le Chiffrement de Format

Le chiffrement de base reçoit des bits et renvoie des bits ; il ne se soucie pas de savoir si les bits représentent du texte, une image, ou un document PDF. Le texte chiffré peut à son tour être encodé sous forme d'octets bruts, de caractères hexadécimaux, de base64, ou tout autre format de sérialisation. Mais que faire si vous avez besoin que le ciphertext ait le même format que le

texte en clair, comme l'exigent certaines base de données qui ne peuvent stocker les données que dans un format prescrit ?

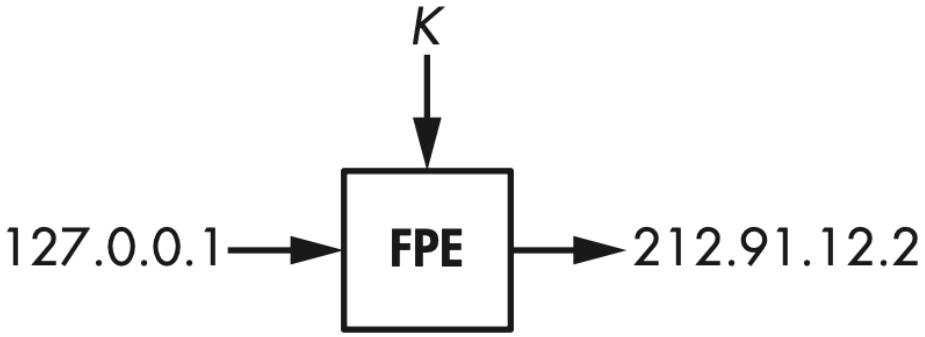

Figure 1-5 : Chiffrement de format pour des adresses IP.

Le chiffrement de format, ou *format-preserving encryption* (FPE) résout ce problème. Il peut créer des ciphertexts qui ont le même format que le texte en clair. Par exemple, le FPE peut chiffrer des adresses IP valides en d'autres adresses IP valides (comme illustré sur la Figure 1-5), des codes postaux en codes postaux, des numéros de carte de crédit en numéros de carte de crédit valides, etc.

Le Chiffrement Homomorphe

Le chiffrement (totalement) homomorphe, ou *fully homomorphic encryption* (FHE) est le Saint Graal des cryptographes : c'est une forme de chiffrement asymétrique qui permet de calculer un ciphertext *C'* (prononcé « C prime ») directement à partir d'un ciphertext $C = \mathbf{FHE}(K, P)$, tel que *C'* puisse être déchiffré en $\mathbf{F}(P)$, où $\mathbf{F}(P)$ peut être n'importe quelle fonction de *P*, et sans jamais exposer le plaintext initial *P*.

Sans le chiffrement homomorphe, il faudrait

1. Obtenir *P* en déchiffrant *C* : $P = \mathbf{D}(K, C)$
2. Appliquer la fonction $\mathbf{F}()$ à *P* : $P = \mathbf{F}(P)$
3. Chiffrer le nouveau *P* : $C' = \mathbf{E}(K, P)$

Imaginez l'application suivante du FHE : un traitement de texte *cloud-based*, comme Google Docs ou Microsoft 365, mais qui n'aurait jamais accès aux documents en clair. Le plaintext *P* serait seulement calculé par les utilisateurs, dans leur navigateur, et chaque changement effectué côté client serait transmis par une *requête de modification chiffrée* au serveur, qui calculerait une nouvelle version du document chiffré, sans pouvoir apprendre la modification faite au plaintext sous-jacent. Cela semble impossible, et pourtant ça marche—la magie de la cryptographie.

Mais il y a un revers à la médaille : le chiffrement homomorphe est lent, très lent, et nous sommes loin de pouvoir implémenter l'application décrite ci-dessus. Cependant, depuis la première construction de FHE publiée 2009, de nombreuses variantes et implémentations ont vu le jour, et on commence à voir des applications émerger, principalement pour des opérations simple (et ce qu'on appelle *partially homomorphic encryption*, une version plus rapide restreinte à certaines fonctions basiques).

Le Chiffrement Cherchable

Le chiffrement cherchable, ou *searchable symmetric encryption* (SSE), permet d'effectuer des recherches dans une base de données chiffrée sans divulguer le contenu de la requête de recherche, ni les valeurs retournées. Comme le chiffrement homomorphe, un SSE efficace et rapide pourrait massivement améliorer la confidentialité des services cloud stockant d'énormes quantités de données. Mais la technologie n'est pas encore prête pour un déploiement simple et universel sur tout type de donnée et pour tout type de requête.

Néanmoins, en 2022 un fournisseur majeur de base de données a intégré le SSE à sa technologie, en se basant sur des protocoles provenant de la recherche académiques. On trouve également diverses solutions commerciales de SSE, plus ou moins fiables.

Le Chiffrement Paramétrable

Le chiffrement paramétrable, ou *tweakable encryption*, est une forme de chiffrement symétrique très proche du chiffrement de base, à l'exception d'un paramètre supplémentaire appelé le *tweak*, qui vise à simuler différentes versions du cryptosystème (voir Figure 1-6).

La principale application du chiffrement paramétrable est le chiffrement de disque (*disk encryption*), pour chiffrer les données au niveau matériel ou au niveau du système de fichier. Dans ce cas, on ne peut pas utiliser de chiffrement probabiliste, car la taille des données doit rester constante. Un attaquant ne doit pas pouvoir copier les données d'un endroit à l'autre du disque, afin de préserver l'intégrité de son contenu. Les *tweakable ciphers* permettent alors de résoudre ces problème, en définissant un tweak propre à la position des données (par exemple, le numéro de secteur du disque ou numéro de bloc).

Dans d'autres cas d'utilisation, le tweak peut aussi, par exemple, être une valeur unique par environnement, par type de système, ou par client d'une société, pour s'assurer que les données chiffrées pour l'un des destinataires ne seront pas directement utilisables par les autres.

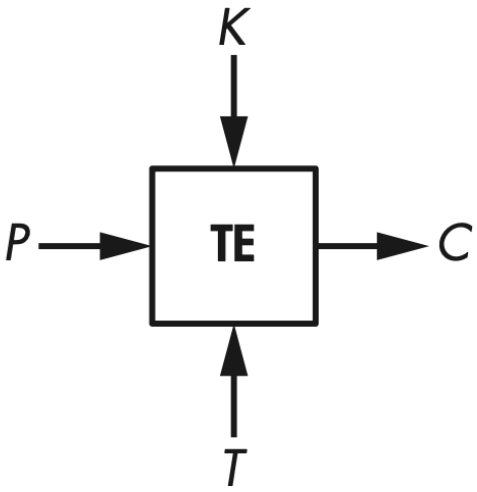

Figure 1-6 : Chiffrement paramétrable, avec un tweak T.

Exemples de Problèmes

Les algorithmes de chiffrement ou leurs implémentations peuvent échouer à protéger la confidentialité des données de plusieurs façons. Cela peut être dû à un manquement aux niveau de sécurité nécessaire (telles que « être IND-CPA ») ou à l'inadéquation des notions de sécurité à la réalité (par exemple, si vous ciblez uniquement la sécurité IND-CPA alors que les attaquants peuvent effectuer des requêtes *chosen-ciphertext*). Hélas, de nombreux ingénieurs ne réfléchissent pas en termes abstraits de notions de sécurité cryptographique, et veulent simplement être « sécurisés » sans comprendre ce que cela signifie réellement. Cela mène généralement droit au désastre. Examinons deux exemples.

Chiffrement Faible

Notre premier exemple concerne les cryptosystèmes vulnérable à la cryptanalyse, comme cela s'est produit avec la norme de communication mobile 2G. Le chiffrement des communications téléphones mobiles 2G utilisait un algorithme de chiffrement appelé A5/1 qui s'est avéré plus faible que prévu, permettant l'interception des appels par toute personne disposant des compétences et des outils appropriés. Les opérateurs de télécommunication ont dû modifier leur système pour empêcher l'attaque.

NOTE : La norme 2G a également défini A5/2, un algorithme pour les zones géographiques autres que l'UE et les États-Unis. A5/2 était volontairement cassable afin d'empêcher l'utilisation de chiffrement fort dans toutes les régions du monde.

Cela dit, l'attaque de A5/1 n'est pas triviale, et il a fallu plus de dix ans aux chercheurs pour trouver une méthode de cryptanalyse efficace. En outre, l'attaque est ce qu'on appelle un compromis temps-mémoire, ou *time-memory trade-off* (TMTO), un type de méthode qui exécute une longue phase préalable de pré-calcul, pendant des jours ou semaines, afin de construire de grande tables de données qui pourront ensuite être utiliser pour l'attaque effective. Pour A5/1, ces tables représentent plus de 1 To. Les normes ultérieures de chiffrement mobile, telles que 3G, LTE, ou 5G, n'utilisent plus A5/1, mais des algorithmes plus fiables.

Mauvais Modèle

L'exemple suivant concerne un modèle d'attaque inadéquat, ayant négligé certains side channels.

De nombreux protocoles utilisant le chiffrement s'assurent qu'ils utilisent des constructions offrants des garanties de sécurité dans le modèle CPA ou CCA. Cependant, certaines attaques ne nécessitent pas de requêtes de chiffrement ni de déchiffrement. Elles ont simplement besoin de *requêtes de validité*, afin de savoir si un ciphertext est valide ou non. Cette information peut généralement être obtenue en envoyant des ciphertexts au système responsable du déchiffrement, et en observant sa réaction. On nomme ce type de source d'information un *oracle*. Et certains types d'oracles peuvent être léthaux à certaines méthodes de chiffrement IND-CPA.

Contre le chiffrement symétrique, on parle surtout de *padding oracle*, un type d'oracle qui, en révélant la validité du ciphertext, révèle la validité de son padding, ou *remplissage* : les octets que l'on rajoute à la fin d'un plaintext pour que sa longueur soit un multiple de la taille de bloc du de

l'algorithme de chiffrement utilisé. Plus précisément, dans le cas des *padding oracle attacks*, un ciphertext n'est valide que si son plaintext possède le padding approprié, conformément à la spécification de l'algorithme. Le décryptage échoue si le padding est incorrect, et les attaquants peuvent souvent détecter les échecs de déchiffrement (via leurs messages d'erreur, par exemple), et chercher à créer des ciphertexts valide, ce qui peut révéler de l'information sur le plaintext. Si cela vous semble encore très abstrait, c'est normal.

Bien que ce type d'attaque était connu par les chercheurs académiques depuis 2002, de peu de systèmes ont été corrigés. En 2010, lorsque ces attaques ont été redécouvertes, implémentées, et appliquées serveurs d'applications web, de nombreux systèmes vulnérables ont finalement été patchés—mais pas tous. Les requêtes de validité consistaient à envoyer des ciphertexts aux serveurs, jusqu'à ce qu'aucune erreur soit observée, et répéter l'opération pour différents ciphertexts soigneusement choisis, afin finalement de retrouver le plaintext (sans jamais connaître la clé).

Les cryptographes négligent souvent les attaques qui exploitent des propriétés des implémentations et du déploiement des algorithmes dans des applications réelles. Mais si vous n'anticipez pas ces risques et ne les incluez pas dans votre modèle lors de la conception et du déploiement de la cryptographie, vous pourriez avoir de mauvaises surprises.

Pour Aller Plus Loin

Nous discuterons du chiffrement sous ses différentes formes de manière plus détaillée tout au long de ce livre, notamment le fonctionnement des algorithmes modernes. Néanmoins, nous ne pouvons pas tout couvrir, et de nombreux sujets fascinants ne seront pas abordés. Par exemple, pour connaître les fondements théoriques du chiffrement et mieux comprendre la notion d'indiscernabilité (IND), je recommande l'article de 1982 qui a introduit l'idée de sécurité sémantique, « Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information » par Goldwasser et Micali. Si vous vous intéressez aux attaques physiques et à la sécurité de la cryptographie en hardware, les actes de la conférence CHES sont la référence.

Il existe également bien d'autres formes de chiffrement que celles présentées dans ce chapitre, notamment le *attribute-based encryption*, le *broadcast encryption*, le *functional encryption*, le *identity-based encryption*, le *time-lock encryption*, ou le *proxy re-encryption*, pour n'en citer que quelques-uns. Pour connaître les dernières recherches sur ces sujets, vous pouvez consulter le site https://eprint.iacr.org/, l'archive publique d'articles de recherche en cryptographie.