0x700

Cryptologie

La *cryptologie* est une science qui englobe la cryptographie et la cryptanalyse. La *cryptographie* sous-tend le processus de communication secrète à l'aide de codes. La *cryptanalyse* correspond au processus de décodage, ou de déchiffrement, de ces communications secrètes. L'intérêt de la cryptologie est apparu historiquement au cours des guerres, pendant lesquelles les pays utilisaient des codes secrets pour communiquer avec leurs troupes tout en essayant de casser les codes de l'ennemi afin d'infiltrer ses communications.

Les applications en temps de guerre existent toujours, mais l'utilisation de la cryptographie dans la vie civile se répand de plus en plus avec l'augmentation des transactions critiques sur Internet. Le reniflage du réseau est si courant qu'imaginer que quelqu'un est toujours à l'écoute du trafic réseau n'est sans doute pas aussi paranoïaque qu'on pourrait le croire. Les mots de passe, les numéros de cartes bancaires et d'autres informations privées peuvent être détectés et volés lorsqu'ils sont transmis à l'aide de protocoles sans chiffrement. Les protocoles de communication chiffrée apportent une solution à cette atteinte à la vie privée et sont au cœur de l'économie sur Internet. Sans le chiffrement SSL (*Secure Sockets Layer*), les transactions bancaires sur les sites Web seraient peu commodes ou non sécurisées.

Toutes ces données privées sont protégées par des algorithmes cryptographiques qui sont probablement sûrs. Aujourd'hui, les cryptosystèmes véritablement sécurisés sont beaucoup trop lourds pour être employés en pratique. Ainsi, à la place d'une preuve mathématique de leur sûreté, nous utilisons des cryptosystèmes qui sont *pratiquement* sûrs. Autrement dit, il existe certainement des méthodes pour prendre en défaut ces codes, mais personne n'a encore été en mesure de les mettre en œuvre. Bien entendu, il existe également des cryptosystèmes qui ne sont absolument pas sûrs. Cela peut venir de leur implémentation, de la taille de la clé ou simplement d'une faiblesse cryptanalytique du code lui-même. En 1997, conformément aux lois des États-Unis, les logiciels exportés ne devaient pas utiliser des clés de chiffrement de taille supérieure à 40 bits. À cause de cette contrainte, le code correspondant n'était pas sûr, comme l'ont montré la société RSA Data Security et Ian Goldberg, un étudiant diplômé de l'université de Californie, à Berkeley. RSA a proposé un défi qui consistait à décoder un message chiffré avec une clé de 40 bits. Trois heures et demie plus tard, Ian avait réussi. Il était alors évident que les clés sur 40 bits ne permettaient pas d'obtenir un cryptosystème sûr.

Les liens entre la cryptologie et le hacking sont variés. D'un point de vue idéaliste, trouver la solution à un casse-tête est extrêmement séduisant pour les curieux. D'un point de vue moins glorieux, les données secrètes protégées par ce casse-tête sont sans doute plus attrayantes. Casser ou contourner les protections cryptographiques de données secrètes peut apporter une certaine satisfaction personnelle, sans faire mention du contenu des données protégées. Par ailleurs, un chiffrement fort est très utile pour éviter la détection. Les systèmes de détection des intrusions coûteux, conçus pour renifler le trafic réseau à la recherche des signatures des attaques, ne servent à rien si l'assaillant emploie un canal de communication chiffrée. L'accès Web chiffré fourni au client pour assurer sa sécurité constitue en réalité un vecteur d'attaque pour les assaillants, car il est difficile à surveiller.

0x710 Théorie de l'information

De nombreux concepts de la sécurité cryptographique sont issus des recherches menées par Claude Shannon. Ses idées ont énormément influencé ce domaine, en particulier les concepts de *diffusion* et de *confusion*. Même si les concepts de sécurité inconditionnelle, de masques jetables, de cryptographie quantique et de sécurité informatique, décrits dans les sections suivantes, ne sont pas dus à Shannon, ses idées sur le secret parfait et la théorie de l'information ont largement influencé les définitions de la sécurité.

0x711 Sécurité inconditionnelle

Un système cryptographique est inconditionnellement sûr s'il ne peut pas être cassé, même avec des ressources de calcul infinies. Autrement dit, la cryptanalyse est impossible et, même si chaque clé possible pouvait être essayée dans une attaque exhaustive par force brute, il serait impossible de déterminer la clé correcte.

0x712 Masques jetables

Les masques jetables font partie des cryptosystèmes inconditionnellement sûrs. Un *masque jetable* est un système très simple qui utilise des blocs de données aléatoires appelés *masques*. Le masque doit être au moins aussi long que le message en clair afin qu'il puisse être codé et les données aléatoires du masque doivent être réellement aléatoires, au sens le plus littéral du terme. Deux masques identiques sont créés : un pour le destinataire et un pour l'émetteur. Pour encoder un message, l'émetteur réalise simplement un OU exclusif entre chaque bit du message en clair et ceux du masque. Après l'encodage du message, le masque est détruit afin d'être certain qu'il ne sera pas réutilisé. Le message chiffré est ensuite envoyé au destinataire, sans craindre la cryptanalyse puisqu'il ne peut pas être décodé sans le masque. Lorsque le destinataire reçoit le message chiffré, il applique également un OU exclusif entre les bits du message reçu et ceux de son masque afin de retrouver le message en clair originel.

Si les masques jetables sont théoriquement impossibles à casser, ils se révèlent peu pratiques à utiliser. La sécurité de ce système repose sur celle des masques. La distribution des masques au destinataire et à l'émetteur doit se faire au travers d'un canal sécurisé. Pour être réellement sûre, elle devrait se faire par un échange en mains propres, mais, pour des raisons pratiques, la transmission du masque est effectuée à l'aide d'un autre code. Par conséquent, la résistance de l'ensemble du système est équivalente à celle du lien le plus faible, c'est-à-dire le code employé pour transmettre les masques. Puisque le masque est constitué de données aléatoires de la même taille que le message en clair et puisque la sécurité du système est identique à celle de la transmission du masque, il suffit généralement d'envoyer le message en clair chiffré avec le code qui aurait été utilisé pour transmettre le masque.

0x713 Cryptographie quantique

L'avènement de l'informatique quantique a eu des conséquences intéressantes sur la cryptologie. En particulier, la distribution quantique des clés a permis l'implémentation pratique des masques jetables. Les mystères de l'intrication quantique peuvent apporter une méthode fiable et secrète pour envoyer une chaîne de bits aléatoire servant de clé. Tout cela s'appuie sur les états quantiques non orthogonaux des photons.

Sans trop entrer dans les détails, la polarisation d'un photon représente la direction d'oscillation de son champ électrique, qui, dans ce cas, peut être horizontale, verticale ou selon l'une des deux diagonales. Les états sont *non orthogonaux* lorsqu'ils sont séparés par un angle différent de 90°. Assez curieusement, il est impossible de déterminer avec certitude la polarisation, parmi les quatre existantes, d'un photon. La base rectiligne des polarisations horizontale et verticale est incompatible avec la base diagonale des deux autres polarisations. Par conséquent, du fait du principe d'incertitude de Heisenberg, il est impossible de mesurer à la fois ces deux formes de polarisation. Des filtres permettent de mesurer les polarisations – un pour la base rectiligne et un autre pour la base diagonale. Lorsqu'un photon passe au travers du filtre adéquat, sa polarisation n'est pas modifiée. En revanche, s'il traverse un filtre inadapté, sa polarisation est modifiée de manière aléatoire. Autrement dit, toute tentative indiscrète de mesurer la polarisation d'un photon a de fortes chances de perturber les données et donc d'indiquer que le canal n'est pas sûr.

Ces aspects étranges de la mécanique quantique ont été exploités par Charles Bennett et Gilles Brassard dans le premier modèle, et certainement le plus connu, de distribution quantique des clés, appelé *BB84*. Tout d'abord, l'émetteur et le destinataire s'accordent sur la représentation binaire des quatre polarisations afin que chaque base ait un 1 et un 0. Dans ce modèle, 1 peut représenter la polarisation verticale et l'une des polarisations diagonales (+45°), tandis que 0 peut représenter la polarisation horizontale et l'autre polarisation diagonale (–45°). Ainsi, la mesure des polarisations rectilignes ou diagonales peut générer des 1 et des 0.

Ensuite, l'émetteur envoie un flux de photons aléatoires, chacun issu d'une base choisie aléatoirement (soit rectiligne, soit diagonale), et ils sont mémorisés. Lorsque

le destinataire reçoit un photon, il choisit également de manière aléatoire sa mesure, par une base rectiligne ou diagonale, et enregistre le résultat. Ensuite, les deux parties comparent publiquement les bases employées pour chaque photon et ne conservent que les données associées aux photons qu'elles ont mesurés avec la même base. Les valeurs binaires des photons ne sont pas révélées, puisque chaque base possède des 1et des 0. La clé du masque jetable est ainsi déterminée.

Puisqu'une écoute indiscrète modifiera la polarisation de certains de ces photons et brouillera les données, elle peut être détectée en calculant le taux d'erreur d'un sous-ensemble aléatoire de la clé. Si les erreurs sont trop nombreuses, une écoute indiscrète est probablement en cours et la clé doit être jetée. Sinon la transmission des données de la clé s'est faite de manière sécurisée et privée.

0x714 Sécurité informatique

Un cryptosystème est considéré *informatiquement sûr* si le meilleur algorithme connu pour le casser nécessite des ressources informatiques et du temps en quantité excessive. Autrement dit, il est théoriquement possible qu'une écoute indiscrète casse le chiffrement, mais elle ne peut pas être mise en pratique, car le temps et les ressources nécessaires dépasseraient de loin la valeur de l'information chiffrée. En général, le temps nécessaire à casser un cryptosystème informatiquement sûr se mesure en dizaines de milliers d'années, même sous l'hypothèse d'une vaste grille de ressources de calcul. La plupart des cryptosystèmes modernes entrent dans cette catégorie.

Il est important de savoir que les meilleurs algorithmes connus pour casser des cryptosystèmes ont toujours évolué et se sont améliorés. Attention à la définition idéale d'un cryptosystème informatiquement sûr, car il n'existe aujourd'hui aucune manière de prouver qu'un algorithme donné de cassage d'un chiffrement est et sera toujours le *meilleur* ! Par conséquent, le meilleur algorithme connu *actuellement* est utilisé à la place pour mesurer le niveau de sécurité d'un cryptosystème.

0x720 Temps d'exécution d'un algorithme

Le *temps d'exécution d'un algorithme* est un peu différent du temps d'exécution d'un programme. Puisqu'un algorithme ne représente qu'une idée, il n'existe aucune limite à la vitesse de traitement pour l'évaluation de l'algorithme. Autrement dit, on ne peut pas exprimer le temps d'exécution d'un algorithme en minutes ou en secondes.

Sans les éléments tels que la rapidité du processeur et l'architecture, l'inconnue principale d'un algorithme est la *taille de l'entrée*. Un algorithme de tri appliqué à 1 000 éléments prendra plus de temps que le même algorithme de tri appliqué à 10 éléments. La taille de l'entrée est généralement notée *n* et chaque étape atomique

est exprimée sous forme d'un nombre. Le temps d'exécution d'un algorithme simple, comme le suivant, est donné en fonction de n :

```
for(i = 1 to n) {
   Faire quelque chose;
   Faire autre chose;
}
Faire une dernière chose;
```

Cet algorithme effectue n boucles, en réalisant à chaque fois deux actions, puis il procède à une dernière opération. Par conséquent, la *complexité temporelle* de cet algorithme est $2n + 1$. Un algorithme plus complexe, avec des boucles imbriquées, comme celui ci-après, a une complexité égale à $n^2 + 2n + 1$, puisque la nouvelle opération est exécutée n^2 fois.

```
for(x = 1 to n) {
    for(y = 1 to n) {
        Faire la nouvelle action;
    }
}
for(i = 1 to n) {
   Faire quelque chose;
   Faire autre chose;
}
Faire une dernière chose;
```

Mais le niveau de détail de cette complexité temporelle est trop fin. Par exemple, lorsque n augmente, la différence relative entre $2n + 5$ et $2n + 365$ diminue. Cependant, lorsque n augmente, la différence relative entre $2n^2 + 5$ et $2n + 5$ augmente. Cette tendance générale est plus importante que le temps d'exécution d'un algorithme.

Prenons deux algorithmes, le premier avec une complexité temporelle égale à $2n + 365$, le second avec la complexité $2n^2 + 5$. L'algorithme en $2n^2 + 5$ sera plus rapide que l'algorithme en $2n + 365$ pour les petites valeurs de n. En revanche, pour $n = 30$, les deux algorithmes affichent la même rapidité, tandis que pour n supérieur à 30 l'algorithme en $2n + 365$ sera plus véloce que l'algorithme en $2n^2 + 5$. Puisqu'il n'existe que 30 valeurs de n pour lesquelles l'algorithme en $2n^2 + 5$ est plus performant, et une infinité de valeurs de n pour lesquelles l'algorithme en $2n + 365$ est plus rapide, ce dernier est, de manière générale, plus efficace.

Autrement dit, le taux de croissance de la complexité temporelle d'un algorithme en fonction de la taille de son entrée est plus important que sa complexité temporelle pour une entrée de taille figée. Même si cela n'est pas toujours vrai pour certaines applications réelles, cette mesure de l'efficacité d'un algorithme se révèle en moyenne juste pour toutes les applications possibles.

0x721 Notation asymptotique

La *notation asymptotique* est une manière d'exprimer l'efficacité d'un algorithme. Elle est dite asymptotique car elle concerne le comportement de l'algorithme lorsque la taille de l'entrée tend vers l'infini.

En revenant aux exemples des algorithmes en $2n + 365$ et en $2n^2 + 5$, nous avons déterminé que le premier est généralement plus efficace car il suit l'évolution de n, tandis que le second suit la courbe générale de n^2. Autrement dit, $2n + 365$ possède une borne supérieure fixée par un multiple positif de n pour toutes les valeurs suffisamment grandes de n, alors que $2n^2 + 5$ possède une borne supérieure fixée par un multiple positif de n^2 pour n suffisamment grand.

Cela peut sembler un peu confus, mais cela signifie qu'il existe en réalité une constante positive pour la valeur de tendance et une borne inférieure sur n, telles que la multiplication de la valeur de tendance par la constante sera toujours supérieure à la complexité temporelle pour tous les n supérieurs à la borne inférieure. Autrement dit, $2n^2 + 5$ est de l'ordre de n^2 et $2n + 365$ est de l'ordre de n. Il existe une notation mathématique pour cela, appelée *grand O*, qui permet d'écrire $O(n^2)$ pour indiquer un algorithme qui est de l'ordre de n^2.

Pour convertir la complexité temporelle d'un algorithme en notation grand O, il suffit d'examiner les termes d'ordre supérieur, puisqu'ils comptent le plus lorsque n devient suffisamment grand. Ainsi, un algorithme dont la complexité temporelle est $3n^4 + 43n^3 + 763n + \log n + 37$ est de l'ordre de $O(n^4)$, tandis qu'un algorithme en $54n^7 + 23n^4 + 4325$ est en $O(n^7)$.

0x730 Chiffrement symétrique

Les codes *symétriques* sont des cryptosystèmes qui utilisent la même clé pour chiffrer et déchiffrer les messages. Le processus de chiffrement et de déchiffrement est généralement plus rapide que dans un modèle asymétrique, mais la distribution des clés peut se révéler difficile.

Le chiffrement se fait soit par bloc, soit par flot. Un *chiffrement par bloc* opère sur des blocs de taille fixe, en général 64 ou 128 bits. Le chiffrement du même bloc de texte en clair produira toujours le même bloc de texte chiffré, si la même clé est employée. DES, Blowfish et AES (Rijndael) sont des méthodes de chiffrement par bloc. Le *chiffrement par flot* génère une séquence de bits pseudo-aléatoires, en général un bit ou un octet à la fois. Cette séquence, appelée *keystream*[1], est combinée au texte en clair par un OU exclusif. Ce mode de fonctionnement est adapté au chiffrement des flux continus de données. RC4 et LSFR sont des algorithmes de

1. NdT : ce terme a été traduit codon dans le livre *Cryptographie appliquée*, de Bruce Schneier (Vuibert).

chiffrement par flot très répandus. Nous détaillerons RC4 à la section "Chiffrement dans les réseaux sans fil 802.11b", page 454.

DES et AES sont tous deux des chiffrements par bloc très utilisés. De nombreux travaux sont menés pour la construction de systèmes de chiffrement par bloc afin qu'ils résistent aux attaques cryptanalytiques connues. La confusion et la diffusion sont deux concepts que l'on retrouve dans le chiffrement par bloc. La *confusion* fait référence aux méthodes employées pour masquer le lien entre le texte en clair, le texte chiffré et la clé. Cela signifie que les bits en sortie doivent provenir d'une transformation complexe fondée sur la clé et le texte en clair. La *diffusion* est utilisée pour étaler au maximum l'influence des bits du texte en clair et ceux de la clé sur l'ensemble du texte chiffré. Le *chiffrement par produit* combine ces deux concepts en répétant différentes opérations simples. DES et AES font partie des algorithmes de chiffrement par produit.

DES s'appuie également sur un réseau de Feistel. Il est utilisé dans plusieurs chiffrements par bloc afin de garantir que l'algorithme est symétrique. Chaque bloc est divisé en deux moitiés, gauche (*L*, *left*) et droite (*R*, *right*). Ensuite, au cours d'une ronde, la nouvelle moitié gauche (L_i) devient l'ancienne moitié droite (R_{i-1}), et la nouvelle moitié droite (R_i) devient l'ancienne moitié gauche (L_{i-1}) combinée par un OU exclusif avec la sortie d'une fonction qui utilise l'ancienne moitié droite (R_{i-1}) et la sous-clé de la ronde (K_i). En général, chaque ronde d'opération dispose d'une sous-clé distincte qui a été calculée à l'avance. Les valeurs de L_i et de R_i sont les suivantes (le symbole $\oplus$ dénote le OU exclusif) :

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

DES utilise 16 rondes. Ce nombre a été choisi de manière à empêcher la cryptanalyse différentielle. La seule faiblesse connue de DES est la taille de sa clé. Puisqu'elle est limitée à 56 bits, l'ensemble des clés peut être essayé dans une attaque par force brute en quelques semaines sur du matériel spécialisé.

Triple-DES corrige ce problème en utilisant deux clés DES réunies pour former une clé sur 112 bits. Le chiffrement du bloc de texte en clair se fait avec la première clé, puis il est déchiffré avec la deuxième clé, et à nouveau chiffré avec la première clé. Le déchiffrement se passe de manière analogue, mais en inversant les opérations de chiffrement et de déchiffrement. Grâce à la clé plus longue, une attaque par force brute devient plus difficile.

La plupart des chiffrements par bloc industriels résistent à toutes les formes connues de cryptanalyse. Par ailleurs, les clés ont des tailles trop importantes pour tenter une attaque par force brute. Cependant, l'informatique quantique offre des possibilités intéressantes, mais elle fait généralement l'occasion d'un battage trop important.

0x731 Algorithme de recherche quantique de Lov Grover

L'informatique quantique promet un parallélisme massif. Un ordinateur quantique peut enregistrer de nombreux états différents dans une superposition (que vous pouvez voir comme un tableau) et réaliser des calculs sur l'ensemble en une fois. Cette possibilité est parfaitement adaptée à une approche par force brute, y compris sur le chiffrement par bloc. Toutes les clés possibles peuvent être chargées dans la superposition, puis l'opération de chiffrement est effectuée sur toutes les clés en même temps. Le point délicat est d'obtenir la bonne valeur depuis la superposition. Les ordinateurs quantiques sont étranges, car, lorsque l'on examine la superposition, l'ensemble se "décohére" en un seul état. Malheureusement, cette décohérence est initialement aléatoire et les chances de se décohérer dans chaque état de la superposition sont égales.

Sans possibilité de manipuler les chances des états de la superposition, nous pouvons obtenir le même effet en devinant simplement les clés. Fortuitement, Lov Grover est arrivé à un algorithme qui permet ces manipulations. Il permet d'augmenter les chances d'un certain état, tout en diminuant celle des autres. Ce processus est répété plusieurs fois jusqu'à ce que la décohérence de la superposition dans l'état souhaité soit quasiment assurée. Cela nécessite environ $O\sqrt{n}$ étapes.

Avec nos quelques connaissances mathématiques, nous comprenons que cela divise par deux la taille de la clé pour une attaque par force brute. Par conséquent, le vrai paranoïaque doublera la taille de la clé d'un chiffrement par bloc afin de le rendre résistant aux possibilités théoriques d'une attaque par force brute à l'aide d'un ordinateur quantique.

0x740 Chiffrement asymétrique

Le chiffrement asymétrique repose sur une clé publique et une clé privée. Comme leur nom l'indique, la *clé publique* est rendue publique, tandis que la *clé privée* n'est pas dévoilée. Un message chiffré à l'aide de la clé publique peut uniquement être déchiffré avec la clé privée. Cela supprime le problème de la distribution de la clé – les clés publiques sont publiques et, en utilisant la clé publique, un message peut être chiffré pour la clé privée correspondante. Contrairement au chiffrement symétrique, il est inutile de disposer d'un canal de communication séparé pour transmettre la clé secrète. En revanche, le chiffrement asymétrique tend à être plus lent.

0x741 RSA

RSA est l'un des algorithmes asymétriques les plus répandus. Sa sécurité se fonde sur la difficulté à factoriser les grands nombres. Tout d'abord, deux nombres premiers sont choisis, P et Q, puis leur produit, N, est calculé :

$$N = P \times Q$$

Ensuite, il est nécessaire de calculer la quantité de nombres qui sont premiers relativement à N et qui se trouvent entre 1 et $N - 1$ (deux nombres sont premiers entre eux si leur plus grand diviseur commun est 1). Il s'agit de l'indicatrice d'Euler, que l'on note généralement à l'aide de la lettre phi minuscule (φ).

Par exemple, $\varphi(9) = 6$, puisque 1, 2, 4, 5, 7 et 8 sont premiers relativement à 9. Vous remarquerez aisément que si N est premier $\varphi(N)$ est égal à $N - 1$. En revanche, il est moins évident de noter que si N est le produit de deux nombres premiers, P et Q, alors, $\varphi(P \times Q) = (P - 1) \times (Q - 1)$. Cette égalité est très intéressante, puisqu'il faut calculer $\varphi(N)$ dans la mise en œuvre de RSA.

Il faut également choisir une clé de chiffrement, E, qui est un nombre premier relativement à $\varphi(N)$. Ensuite, une clé de déchiffrement qui satisfait l'équation suivante, où S est un entier, doit être déterminée :

$$E \times D = S \times \varphi(N) + 1$$

Cette équation peut être résolue à l'aide de l'algorithme d'Euclide étendu. L'*algorithme d'Euclide* est très ancien, mais il se révèle très rapide pour le calcul du plus grand diviseur commun (PGCD) de deux nombres. Le plus grand des deux nombres est divisé par le plus petit, en prêtant attention au reste. Ensuite, le plus petit nombre est divisé par le reste et le processus se répète jusqu'à ce que le reste soit égal à zéro. La dernière valeur avant que le reste soit égal à zéro correspond au plus grand diviseur commun des deux nombres initiaux. Cet algorithme est assez rapide, avec un temps d'exécution en $O(\log_{10}N)$. Autrement dit, il faut autant d'étapes pour trouver la réponse qu'il y a de chiffres dans le plus grand nombre.

Le tableau ci-après illustre le calcul du PGCD de 7253 et de 120, écrit sous la forme pgcd(7253, 120). Les deux nombres sont tout d'abord placés dans les colonnes A et B, le plus grand étant dans la colonne A. Ensuite, A est divisé par B et le reste est inscrit dans la colonne R. À la ligne suivante, l'ancien B devient le nouveau A et l'ancien R devient le nouveau B. R est à nouveau calculé et la procédure se répète jusqu'à ce que le reste soit égal à zéro. La dernière valeur de R, avant zéro, est le plus grand diviseur commun.

pgcd(7253, 120)

A	*B*	*R*
7253	120	53
120	53	14
53	14	11
14	11	3
11	3	2
3	2	1
2	1	0

Le PGCD de 7253 et de 120 est donc 1. Autrement dit, 7253 et 120 sont des nombres premiers entre eux.

L'*algorithme d'Euclide étendu* se charge de trouver deux entiers, J et K, tels que

$$J \times A + K \times B = R$$

lorsque pgcd(A, B) = R.

Pour cela, l'algorithme d'Euclide est déroulé à reculons. Cependant, dans ce cas, les quotients sont importants. Voici la procédure mathématique de l'exemple précédent, avec les quotients :

$$7253 = 60 \times 120 + \mathbf{53}$$

$$120 = 2 \times 53 + \mathbf{14}$$

$$53 = 3 \times 14 + \mathbf{11}$$

$$14 = 1 \times 11 + \mathbf{3}$$

$$11 = 3 \times 3 + \mathbf{2}$$

$$3 = 1 \times 2 + \mathbf{1}$$

Grâce à une petite opération algébrique, nous pouvons déplacer les termes sur chaque ligne de manière que le reste (en gras) se trouve seul à gauche du signe égal :

$$\mathbf{53} = 7253 - 60 \times 120$$

$$\mathbf{14} = 120 - 2 \times 53$$

$$\mathbf{11} = 53 - 3 \times 14$$

$$\mathbf{3} = 14 - 1 \times 11$$

$$\mathbf{2} = 11 - 3 \times 3$$

$$\mathbf{1} = 3 - 1 \times 2$$

En partant de la fin, il est clair que

$$1 = 3 - 1 \times \mathbf{2}$$

La ligne au-dessus, qui est $2 = 11 - 3 \times 3$, nous permet de remplacer 2 :

$$1 = 3 - 1 \times (11 - 3 \times 3)$$

$$1 = 4 \times \mathbf{3} - 1 \times 11$$

La ligne au-dessus, qui est $3 = 14 - 1 \times 11$, nous permet de remplacer 3 :

$$1 = 4 \times (14 - 1 \times 11) - 1 \times 11$$

$$1 = 4 \times 14 - 5 \times \mathbf{11}$$

Bien entendu, nous trouvons la substitution suivante dans la ligne au-dessus, $11 = 53 - 3 \times 14$:

$$1 = 4 \times 14 - 5 \times (53 - 3 \times 14)$$

$$1 = 19 \times \mathbf{14} - 5 \times 53$$

Nous reproduisons la même opération avec la ligne au-dessus, $14 = 120 - 2 \times 53$:

$$1 = 19 \times (120 - 2 \times 53) - 5 \times 53$$

$$1 = 19 \times 120 - 43 \times \mathbf{53}$$

Enfin, la dernière substitution nous est donnée par la première ligne, $53 = 7253 - 60 \times 120$:

$$1 = 19 \times 120 - 43 \times (7253 - 60 \times 120)$$

$$1 = 2599 \times 120 - 43 \times 7253$$

$$2599 \times 120 + -43 \times 7253 = 1$$

Nous déterminons ainsi que $J = 2599$ et que $K = -43$.

Les nombres de l'exemple précédent ont été choisis en raison de leur adéquation avec RSA. Si nous prenons les valeurs 11 et 13 pour P et Q, N est égal à 143. Par conséquent, $\varphi(N) = 120 = (11 - 1) \times (13 - 1)$. Puisque 7253 est premier relativement à 120, ce nombre constitue une très bonne valeur pour E.

Notre objectif était de trouver une valeur de D qui satisfait l'équation suivante :

$$E \times D = S \times \varphi(N) + 1$$

Un peu d'algèbre et la voici sous une forme plus familière :

$$D \times E + S \times \varphi(N) = 1$$

$$D \times 7253 \pm S \times 120 = 1$$

En utilisant les valeurs obtenues avec l'algorithme d'Euclide étendu, il apparaît que $D = -43$. La valeur de S a peu d'importance. Autrement dit, l'opération mathématique est effectuée modulo $\varphi(N)$, ou modulo 120. Cela signifie donc que 77 est une valeur positive équivalente pour D, puisque $120 - 43 = 77$. Nous pouvons replacer tout cela dans l'équation précédente :

$$E \times D = S \times \varphi(N) + 1$$

$$7253 \times 77 = 4654 \times 120 + 1$$

Les valeurs de N et de E constituent la clé publique, tandis que D est gardée secrète. P et Q sont abandonnés. Les fonctions de chiffrement et de déchiffrement sont relativement simples.

Chiffrement : $C = M^E (\text{mod } N)$

Déchiffrement : $M = C^D (\text{mod } N)$

Par exemple, si le message, *M*, est 98, son chiffrement donne :

$98^{7253} = 76 (\text{mod } 143)$

Le texte chiffré est 76. Ensuite, si quelqu'un connaît la valeur de *D*, il peut déchiffrer le message et retrouver le nombre 98 à partir du nombre 76 :

$76^{77} = 98 (\text{mod } 143)$

Évidemment, si le message *M* est plus grand que *N*, il doit être décomposé en tronçons plus petits que *N*.

Cette procédure est possible grâce au théorème de l'indicatrice d'Euler. Il stipule que si *M* et *N* sont premiers entre eux, avec *M* étant le plus petit nombre, alors, lorsque *M* est multiplié par lui-même $\varphi(N)$ fois et divisé par N le reste sera toujours 1 :

Si pgcd$(M, N) = 1$ et si $M < N$ alors $M^{\varphi(N)} = 1 (\text{mod } N)$

Puisque toute cette opération se fait modulo *N*, les équations suivantes sont également vraies (du fait des propriétés de la multiplication en arithmétique modulo) :

$M^{\varphi(N)} \times M^{\varphi(N)} = 1 \times 1 (\text{mod } N)$

$M^{2 \times \varphi(N)} = 1 (\text{mod } N)$

Ce processus peut être répété *S* fois afin d'obtenir l'équation suivante :

$M^{S \times \varphi(N)} = 1 (\text{mod } N)$

Si nous multiplions les deux côtés par *M*, nous obtenons :

$M^{S \times \varphi(N)} \times M = 1 \times M (\text{mod } N)$

$M^{S \times \varphi(N) + 1} = M (\text{mod } N)$

Cette équation est au cœur de RSA. Un nombre *M*, élevé à une puissance modulo *N*, produit à nouveau le nombre *M* initial. Autrement dit, nous avons une fonction qui retourne sa propre entrée, ce qui, en soi, n'est pas très intéressant. En revanche, si cette équation peut être décomposée en deux parties distinctes, la première peut être employée pour chiffrer et la seconde, pour déchiffrer, en produisant le message d'origine. Pour cela, il suffit de trouver deux nombres, *E* et *D*, dont la multiplication est égale à *S* fois $\varphi(N)$ plus 1. Ensuite, ces valeurs peuvent être insérées dans l'équation précédente :

$E \times D = S \times \varphi(N) + 1$

$M^{E \times D} = M (\text{mod } N)$

Ce qui équivaut à :

$$M^{E^D} = M(\text{mod } N)$$

Nous pouvons alors la décomposer en deux étapes :

$$M^E = C(\text{mod } N)$$

$$C^D = M(\text{mod } N)$$

Et voilà pour les fondamentaux de RSA. La sécurité de l'algorithme repose sur le secret de *D*. Mais, puisque *N* et *E* sont des valeurs publiques, si nous pouvons factoriser *N* afin de retrouver les valeurs *P* et *Q* originelles, alors, il est facile de calculer $\varphi(N)$ avec $(P-1) \times (Q-1)$, puis de déterminer *D* avec l'algorithme d'Euclide étendu. Par conséquent, la taille des clés de RSA doit être choisie en ayant à l'esprit le meilleur algorithme de factorisation connu afin de garantir la sécurité informatique. Aujourd'hui, le meilleur algorithme de factorisation connu pour les grands nombres se nomme NFS (*number field sieve*). Cet algorithme possède un temps d'exécution sous-exponentiel, ce qui n'est pas mal, mais pas assez rapide pour craquer une clé RSA sur 2 048 bits en un temps raisonnable.

0x742 Algorithme de factorisation quantique de Peter Shor

À nouveau, l'informatique quantique promet des augmentations stupéfiantes en potentiel de calcul. Peter Shor a été capable d'exploiter le parallélisme massif des ordinateurs quantiques pour factoriser efficacement des nombres en utilisant une ancienne astuce de la théorie des nombres.

L'algorithme est assez simple. Soit le nombre *N* à factoriser. Choisissons une valeur *A* inférieure à *N*. Cette valeur doit également être première relativement à *N*, mais, en supposant que *N* est le produit de deux nombres premiers (ce qui est toujours le cas lorsqu'on tente de factoriser des nombres pour casser RSA), si *A* n'est pas premier relativement à *N*, alors, *A* est l'un des facteurs de *N*.

Ensuite, nous chargeons la superposition avec des nombres séquentiels en partant de 1 et en passant chacune de ces valeurs à la fonction $f(x) = A^x(\text{mod } N)$. Grâce à la magie de l'informatique quantique, tout cela se fait en même temps. Un motif de répétition émergera dans les résultats et la période de répétition doit être déterminée. Heureusement, elle peut être obtenue rapidement sur un ordinateur quantique à l'aide de la transformée de Fourier. Nous appelons *R* cette période.

Puis nous calculons simplement $\text{pgcd}(A^{R/2} + 1, N)$ et $\text{pgcd}(A^{R/2} - 1, N)$. Au moins l'une de ces valeurs doit être un facteur de *N*. En effet, $A^R = 1(\text{mod } N)$, ce que nous expliquons ci-après :

$$A^R = 1(\text{mod } N)$$

$$(A^{R/2})^2 = 1(\text{mod } N)$$

$(A^{R/2})^2 - 1 = 0(\text{mod } N)$

$(A^{R/2} - 1) \times (A^{R/2} + 1) = 0(\text{mod } N)$

Cela signifie que $(A^{R/2} - 1) \times (A^{R/2} + 1)$ est un entier multiple de *N*. Tant que ces valeurs ne s'annulent pas, l'une d'elles aura un facteur en commun avec *N*.

Pour craquer l'exemple RSA précédent, nous devons factoriser la valeur publique *N*. Dans ce cas, *N* est égal à 143. Nous choisissons ensuite une valeur pour *A*, de manière qu'elle soit première relativement à *N* et inférieure à *N*. Nous obtenons donc $A = 21$. La fonction devient alors $f(x)= 21^x(\text{mod } 143)$. Toutes les valeurs à suivre, à partir de 1 jusqu'à celle autorisée par l'ordinateur quantique, sont passées à cette fonction.

Pour faire court, nous supposons que l'ordinateur quantique traite des valeurs sur 3 bits et que la superposition peut donc contenir huit valeurs :

$x = 1 \mid 211(\text{mod } 143) = 21$

$x = 2 \mid 212(\text{mod } 143) = 12$

$x = 3 \mid 213(\text{mod } 143) = 109$

$x = 4 \mid 214(\text{mod } 143) = 1$

$x = 5 \mid 215(\text{mod } 143) = 21$

$x = 6 \mid 216(\text{mod } 143) = 12$

$x = 7 \mid 217(\text{mod } 143) = 109$

$x = 8 \mid 218(\text{mod } 143) = 1$

Dans cet exemple, un simple coup d'œil permet de déterminer facilement la période : *R* vaut 4. À partir de cette information, $\text{pgcd}(21^2 - 1143)$ et $\text{pgcd}(21^2 + 1143)$ doit produire au moins l'un des facteurs. Cette fois-ci, les deux facteurs apparaissent, car $\text{pgcd}(440, 143) = 11$ et $\text{pgcd}(442, 142) = 13$. Ces deux facteurs permettent de recalculer la clé privée utilisée dans notre exemple RSA précédent.

0x750 Chiffrement hybride

Un cryptosystème *hybride* bénéficie du meilleur des deux mondes. Un chiffrement asymétrique permet d'échanger une clé générée aléatoirement. Elle est ensuite utilisée pour les communications au travers d'un chiffrement symétrique. Cette approche apporte la rapidité et l'efficacité du chiffrement symétrique, tout en résolvant le problème de l'échange sécurisé de la clé. Le chiffrement hybride est employé dans les applications cryptographiques, comme SSL, SSH et PGP.

Puisque la plupart des applications s'appuient sur un code résistant à la cryptanalyse, une attaque sur ce code ne réussira pratiquement jamais. En revanche, si un