Table des matières

Chapitre 1

Introduction à la cryptographie

La cryptographie est l'art de protéger l'information (par exemple en la rendant secrète via un chiffrement, ou en la signant) en utilisant un secret (on dit aussi une clé). La *cryptanalyse* regroupe l'ensemble des procédés mis au point pour outrepasser (ou *attaquer*) les procédés de protection mis au point en cryptographie.
La *cryptologie* est la conjonction de ces deux disciplines, cryptographie et cryptanalyse.
Historiquement, elle a longtemps été l'apanage des empereurs, des diplomates ou des militaires. La cryptographie a d'ailleurs, en France, été considérée comme une arme de guerre jusqu'en 1999. De nos jours, même si les applications militaires existent toujours, la cryptographie est peu à peu passée d'une *science du secret* à une *science de la confiance*, au fur et à mesure que de nouvelles applications apparaissaient ; en particulier, le développement du réseau Internet (et du commerce électronique) a ouvert des champs d'applications en posant implicitement le défi d'être capable de trouver des outils qui permettent d'interagir à distance et de manière sécurisée. Ainsi, au-delà de la problématique historique de la confidentialité des communications, de nouvelles questions ont vu le jour, essentiellement à partir de l'introduction de la cryptographie à clé publique en 1976 par Whitfield Diffie et Martin Hellman dans leur article fondateur « New directions in cryptography ». Cet article a posé les bases de la cryptographie moderne et de la cryptographie à clé publique.

Traditionnellement, la cryptographie permettait à deux personnes partageant un secret commun de pouvoir échanger des messages chiffrés qu'elles seules étaient censées pouvoir déchiffrer à partir de leur secret (ou clé) commun ; la clé pouvait être utilisée à la fois pour chiffrer et déchiffrer un message donné. Cette approche répond maintenant au nom de *cryptographie symétrique*. Les limites de ce type de protocole sont d'une part la nécessité pour deux individus voulant communiquer entre eux de partager un secret avant de communiquer et, d'autre part, la nécessité de partager autant de secrets communs qu'il y a de personnes avec qui ils voudraient communiquer.

La *cryptographie à clé publique* casse la symétrie des systèmes de chiffrement à clé symétrique en partant du principe de l'existence non pas d'une même clé commune aux deux individus (que l'on appelle traditionnellement Alice et Bob ou, plus prosaïquement, A et B), mais plutôt d'un couple de clés (ou biclé) : une clé publique accessible à tous qui permet de chiffrer un message, et une clé privée (ou secrète) qui permet de déchiffrer les messages chiffrés par la clé publique associée. Ce point de vue permet alors de ne publier qu'une même clé publique, accessible à tous dans un annuaire public par exemple, et réduit à la fois le nombre de clés globalement nécessaires ainsi que le fait d'avoir à partager un secret commun au préalable. Ce type d'idée est aussi valable pour la signature où la clé privée est utilisée pour signer un message et la clé publique pour vérifier qu'un message a bien été signé par la clé privée associée.

En pratique, pour le chiffrement, les systèmes à clé symétrique restent beaucoup plus rapides que les systèmes à clé publique. On utilisera donc la cryptographie à clé publique pour que deux entités puissent partager une clé commune, puis on utilisera alors les algorithmes symétriques pour les échanges ultérieurs entre les deux parties. Pour la plupart des autres types de problèmes (par exemple la signature), la cryptographie à clé publique est incontournable.

Les mécanismes de sécurité dont on souhaite disposer pour la sécurité des systèmes d'information sont les suivants :

1) *confidentialité* : la possibilité pour Alice et Bob de communiquer de manière confidentielle, de telle sorte qu'un importun ayant accès à un message échangé ne puisse en connaître le contenu ;

2) *intégrité des données* : ou comment être certain qu'un document n'a pas été modifié ;

3) *authentification* : Bob veut être sûr que le message a bien été envoyé par Alice ;

4) *non-répudiation* : Alice ne doit pas être capable de nier qu'elle a fait un type d'action donné (une signature d'un message par exemple) ;

5) *contrôle d'accès* : être capable de contrôler qui accède aux données.

En plus de ces mécanismes de base pour la sécurité des systèmes d'information, la cryptologie propose des protocoles plus spécifiques, par exemple (parmi d'autres) :

1) *la signature électronique :* comment signer un document de manière électronique en étant sûr de l'identité de la personne qui a signé et que cette personne ne puisse pas nier avoir signé le document ;

2) *le partage de secret :* comment partager un secret à plusieurs, par exemple si l'on veut qu'un document ne soit reconstructible que si plusieurs personnes interagissent ;

3) *l'échange de clés :* comment deux personnes, qui n'ont pas de secret préalable en commun, peuvent être capables de partager un secret ;

4) *le vote électronique :* comment voter de manière anonyme ;

5) *l'argent électronique :* comment manipuler de l'argent électronique sans pouvoir faire de la fausse monnaie.

La cryptographie est un domaine en évolution rapide et de nouvelles problématiques apparaissent régulièrement. Nous allons décrire ici des solutions à ces problèmes qui seront majoritairement fondées sur des problèmes de théorie algorithmique des nombres (factorisation, logarithme discret, etc.). Cette présentation ne doit pas occulter le fait que la cryptographie ne se réduit pas à sa composante mathématique : les systèmes utilisés concrètement (par exemple industriellement) sont un entrelacement souvent sophistiqué de techniques mathématiques comme celles qui suivent, mais aussi d'aspects informatiques (matériels et logiciels) poussés.

On dira avoir *cassé* un système cryptographique si l'on est capable de déchiffrer un message (ou de signer frauduleusement le cas écheant) sans connaître la clé (ou, a fortiori, si l'on arrive à retrouver la clé). On parle d'*attaque par force brute* lorsque l'on essaie exhaustivement tous les paramètres d'un système (par exemple en essayant toutes les clés de chiffrement possibles).

Un système est considéré comme cryptanalysé si l'on a mis en évidence une attaque qui a une complexité inférieure au niveau de sécurité annoncé pour le système (même quand, en pratique, l'attaque ne permettrait pas de casser le système).

Pour attaquer un procédé cryptographique, il faut faire des hypothèses sur la quantité d'information dont on peut disposer pour travailler. Par exemple, pour un algorithme de chiffrement à clé symétrique, on a :

1) l'attaque à chiffré seul : on ne connaît que des messages chiffrés ;

2) l'attaque à clair connu : on dispose de messages clairs et de chiffrés associés ;

3) l'attaque à chiffré choisi : on peut choisir le message clair et obtenir un chiffré associé.

En général, on exige d'un système qu'il soit résistant au type d'attaque qui donne le plus d'information et qui puisse être menée en pratique : l'attaque à chiffré choisi. Lorsque l'on choisit des tailles de paramètres, on choisit des valeurs qui sont résistantes aux attaques par force brute.

On évalue la sécurité d'un système en fonction du nombre d'opérations élémentaires (une somme de 2 bits par exemple) nécessaire a priori pour cryptanalyser (ou casser) le système par les meilleures attaques connues. On compare ce nombre à ce qu'il est envisageable de faire avec des ordinateurs existants ou probables.

Considérons un ordinateur avec une architecture à 64 bits ; à chaque top d'horloge, l'ordinateur peut faire 64 opérations élémentaires. Pour se donner une idée, on peut calculer le nombre d'opérations élémentaires pouvant être faites par 10 000 ordinateurs en réseaux à une

fréquence de 2 gigahertz pendant une année, soit $10\,000.64.2.10^9.3600.24.365 \approx 2^{75}$ opérations élémentaires. En 2008, on considère donc que 2^{80} opérations élémentaires représentent ce qu'il est calculatoirement infaisable de faire. Mais ce nombre évolue en fonction de la puissance des ordinateurs.

La loi heuristique de Moore (suivie depuis plus de quarante ans) affirme que la vitesse des ordinateurs double tous les dix-huit mois, ce qui donne un rapport d'à peu près 100 tous les dix ans. Ainsi, dans les années 1980, on considérait que 2^{60} opérations étaient infaisables ; maintenant, on prend 2^{80} et, d'ici quelques années, on passera à 2^{100}.

I. Un rapide historique de la cryptographie

I.1. Transposition : la scytale grecque

Les méthodes historiques de la cryptographie commencent par la scytale grecque au quatrième siècle avant J.-C. où, pour communiquer de manière chiffrée, deux individus avaient en commun le diamètre d'une scytale (un long baton de diamètre constant). Pour chiffrer, Alice entourait la scytale comme une bande autour de la scytale puis écrivait de manière verticale sur la bande. Le message chiffré était alors la bande écrite. Le message chiffré, la bande lue horizontalement, apparaissait donc comme un anagramme du message. Pour déchiffrer, Bob entourait le message autour de sa propre scytale avec le même diamètre et lisait le message verticalement. Dans ce cas, la notion de secret partagé est donc le diamètre de la scytale.

On appelle ce type de technique une *transposition* qui consiste à mélanger les lettres du message au milieu d'autres.

I.2. Le chiffrement de César (substitution)

César, pour communiquer avec ses généraux, partait d'un texte en latin puis décalait toutes les lettres de 3 : A $\to$ D, B $\to$ E, C $\to$ F, ..., Z $\to$ C.
Par ce procédé, le message *VENI VIDI VICI* devient *YHQL YLGL YLFL*. Pour déchiffrer, il suffit de décaler les lettres du message de trois dans l'autre sens. Le secret partagé est la valeur du décalage. Dans ce cas, on parle de *substitution* car chaque lettre est substituée par une autre.

Ces deux exemples de chiffrement par transposition et par substitution forment la base théorique des systèmes de chiffrement. La substitution *engendre* de la confusion pour retrouver le message et la transposition *diffuse* cette confusion.

I.3. Substitutions monoalphabétiques

On parle de chiffrement monoalphabétique lorsqu'une lettre est toujours remplacée par une même lettre.
Le système de chiffrement de César peut être vu comme comme un chiffrement décalé :

$$x \in \{0, \dots, 25\} \to y = x + 3 \pmod{26}.$$

Ce chiffrement peut être généralisé en un chiffrement affine de la forme

$$x \in \{0, \dots, 25\} \to y = ax + b \pmod{26},$$

avec $\mathtt{pgcd}(a, 26) = 1$. Pour déchiffrer, on pose $y \in \{0, \cdots, 25\} \to x = a^{-1}(y - b) \pmod{26}$, la clé partagée étant dans ce cas les valeurs de a et b.

Test 1.1.

Montrer que si l'on connaît un message d'au moins 2 lettres et son chiffré par substitution monoalphabétique affine, alors on peut retrouver les paramètres a et b.

Dans ce cas, l'ensemble des clés est l'ensemble des couples (a, b). Il y a 13 possibilités pour a car a doit être premier avec 26 et peut aussi être nul ; pour b, il y a 26 possibilités. On obtient un total de $13 \times 26 = 338$ manières différentes de chiffrer. On peut donc, de manière raisonnable, tester toutes les clés possibles et voir celles qui amènent à un texte compréhensible : c'est un exemple d'attaque par force brute réussie. Cet exemple montre que l'ensemble des clés doit être de très grande taille pour qu'un système de chiffrement soit résistant.

Le cas du chiffrement affine est un cas particulier de substitution monoalphabétique. On peut imaginer que la clé secrète soit la liste complète des substitutions : on se donne une permutation (fixée) sur l'ensemble des lettres. Par exemple, si l'on choisit HURFABOPL ... K comme référence, cela signifiera que $A \to H, B \to U, C \to R, \ldots, Z \to K$. Dans ce cas, le nombre de clés possibles est l'ensemble des permutations possibles soit $26! \approx 2^{88}$. Il n'est donc pas possible de faire ici une recherche exhaustive sur l'ensemble de toutes les clés possibles.

On peut, en revanche, attaquer par la fréquence des lettres de l'alphabet. En effet, si l'on prend un corpus important (comme par exemple un texte de 10 000 caractères), on s'aperçoit que certaines lettres apparaissent en moyenne plus que d'autres. Par exemple, en français, les lettres apparaissent en moyenne dans les proportions suivantes :

$$E : 17,75\,\% \ - \ A : 8,25\,\% \ - \ S : 8,25\,\% \ - \ I : 7,25\,\% \ - \ R : 7,25\,\%$$
$$N : 7,25\,\% \ - \ U : 6,25\,\% \ - \ O : 5,75\,\% \ - \ L : 5,75\,\% \ - \ \ldots$$

On peut utiliser cette observation pour donner une attaque sur un texte chiffré de manière monoalphabétique. Si l'on a un texte long chiffré, la fréquence de chaque lettre reste invariante par substitution. En menant une étude des fréquences statistiques à partir du texte chiffré, on peut donc (si le texte est assez long) retrouver des fréquences et essayer de les comparer aux fréquences originelles des lettres pour retrouver la valeur de chaque lettre.

Cet exemple montre qu'avoir un ensemble de clés important est une condition nécessaire de sécurité, mais qu'elle n'est pas suffisante.

Test 1.2.

Déchiffrer le texte suivant : ABT BRA UOPM HGP FONXQPA UOBWMOB, chiffré par substitution monoalphabétique.

I.4. Chiffrement de Vigenère

Le chiffrement de Vigenère est une variation sur le chiffrement monoalphabétique classique. Introduit au XVI^{e} siècle, il procède à partir d'une clé secrète commune, comme une phrase qui va être placée juste en-dessous du message à chiffrer. On décale les lettres pour les chiffrer en fonction de la valeur de la lettre associée à la clé $A \to 0, B \to 1, \ldots, Z \to 25$.
Par exemple, prenons la clé *CRYPTO* et le texte *CHIFFREMENT*. On écrit

message	:	CHIFFREMENTS
clé de chiffrement	:	CRYPTOCRYPTO
message chiffré	:	EYGUYFGDCCMG.

On voit que, par cette méthode, le chiffrement d'une lettre donnée dépendra de la lettre associée dans la clé. Par exemple, le premier F n'est pas chiffré comme le deuxième. On peut attaquer ce système en deux temps. Tout d'abord, il convient de trouver la longueur de la clé ; dans un

deuxième temps, on procède à nouveau par analyse statistique. Une fois que l'on a la longueur de la clé, on sait que, si l'on prend comme période dans le texte la longueur de la clé, toutes les lettres associées en prenant des lettres du chiffré espacées de la taille de la clé seront décalées par la même valeur. On peut alors faire une analyse statistique (si l'on a un long texte).

I.5. Chiffrement produit (substitutions mêlées aux transpositions)

Pendant la Première Guerre mondiale, les allemands utilisaient un chiffrement produit, en ce sens que ce chiffrement alternait des substitutions et des transpositions. Ce chiffrement, dit ADFGX, procédait de la manière suivante. On considère le carré :

	A	D	F	G	X
A	p	g	c	e	n
D	b	q	o	z	r
F	s	l	a	f	t
G	m	d	v	i	w
X	k	u	y	x	h

Chaque lettre de l'alphabet (à part le j) peut être lue comme un élément du tableau ayant une coordonnée verticale et horizontale en $\mathsf{A}, \mathsf{D}, \mathsf{F}, \mathsf{G}$ et X. Ainsi, à chaque lettre à chiffrer, on fera correspondre ses deux coordonnées en commençant par la coordonnée verticale : au mot *deutsch* on fera correspondre *GD AG XD FX FA AF XX*. On écrit alors les diverses lettres sous forme de lignes de longueur un mot clé donné, on réordonne les colonnes en fonction de l'ordre alphabétique des lettres du mot clé et on lit le message chiffré en lignes. Par exemple, pour décomposer notre mot *deutsch* avec la clé *NICE*, on écrit

NICE		CEIN
GDAG		AGDG
XDFX	qui devient	FXDX
FAAF		AFAF
XX		XX

et le chiffré est *AGDGFXDXAFAFXX*.

Ce système a été cassé avant la fin de la guerre par le cryptanalyste français Georges Painvin qui travaillait au bureau du chiffre.

I.6. Chiffrement par blocs

Dans les exemples que nous avons vus jusqu'à présent, les lettres sont essentiellement traitées de manière séquentielle, à l'origine. En 1929, L. Hill a introduit un système où le chiffré n'était pas construit séquentiellement à partir de lettres, mais à partir de blocs de lettres. Le système est très facile à casser, mais il a été le premier système à utiliser une structure de blocs et à faire un usage clair de l'algèbre et de l'algèbre linéaire. De nos jours, la plupart des systèmes de chiffrement utilisés, comme le D.E.S. ou l'A.E.S., sont des chiffrements par blocs.

Le système fonctionne de la manière suivante. On considère des blocs $\mathsf{m} = (\mathsf{x}, \mathsf{y}, \mathsf{z})$ de trois lettres considérées comme des entiers modulo 26 ; on se donne une matrice M de taille 3×3 à coefficients dans $\mathbb{Z}$ telle que son déterminant soit premier avec 26. Le chiffré c est le résultat du produit $\mathsf{c} = \mathsf{m}\mathsf{M}$. Le fait que M ait un déterminant premier avec 26 assure qu'il existe une matrice 3×3 N, à valeur dans les entiers modulo 26, telle que $\mathsf{M}\mathsf{N} = \mathrm{I}$ la matrice identité (modulo 26). Pour déchiffrer, on multiplie juste le chiffré par N.

Test 1.3.

On considère le système de Hill sur lequel on va faire une attaque à clair connu. Montrer que si l'on connaît quelques clairs (différents du vecteur nul) et leurs chiffrés, on peut retrouver M, N et donc casser le système.

I.7. Chiffrement de Vernam (chiffrement à flot)

En 1918, G. Vernam a introduit un système dont le principe est encore très utilisé aujourd'hui. On commence par écrire toutes les lettres de l'alphabet, les chiffres, la ponctuation et les majuscules sous forme d'une suite de 8 bits (c'est par exemple le cas du code ASCII). Sur 8 bits, on peut écrire 256 nombres différents (de 0 à 255) qui vont tous représenter un symbole. On peut voir ce système comme un dictionnaire qui marche dans les deux sens : à une suite de 8 bits, on peut aussi associer un symbole donné.

Le système de Vernam (aussi appelé *masque jetable* ou *chiffrement à flot*) fonctionne de la manière suivante. On suppose que les deux parties ont en commun une séquence de bits aléatoire s_i qu'elles peuvent synchroniser ; par exemple, historiquement, ces séquences étaient définies au préalable en fonction d'un jour donné.

Le chiffrement s'effectue bit à bit. Considérons les bits m_i du message clair m. Les bits c_i du message chiffré c seront $c_i = m_i \oplus s_i$ où la somme s'effectue modulo 2 ; en informatique et dans la suite du texte, on parle de XOR (un *ou exclusif*) sur 2 bits. Pour déchiffrer, on n'aura qu'à faire la somme modulo 2 de c_i et s_i :

$$c_i \oplus s_i = m_i \oplus s_i \oplus s_i = m_i \pmod 2.$$

Par exemple avec un message 01001101 et une suite d'aléas 10101110, on obtient

m	01001101
s	10101110
c	11100011

et, pour le déchiffrement,

c	11100011
s	10101110
m	01001101

.

À première vue, le système paraît intriguant puisqu'en moyenne un bit sur deux n'est pas modifié. Mais le fait de ne pas savoir quels bits ne sont pas modifiés rend le système incassable théoriquement. En fait, n'importe quelle suite de bits de chiffré peut correspondre à n'importe quel clair possible lorsque l'on suppose que toutes les clés (les suites de bits aléatoires) sont possibles. On peut donc montrer que ce chiffrement est cryptographiquement sûr (c'est même le seul dont la sûreté soit prouvée théoriquement). En pratique, la difficulté va être de construire un générateur d'aléas non prévisibles (et la synchronisation). Nous reviendrons plus en détail sur la manière de construire des générateurs aléatoires dans la section sur le chiffrement à flot.

I.8. Chiffrement mécanique

La fin de la Première Guerre mondiale a vu le début de la mécanisation du chiffrement où l'on utilisait des machines à rotors pour effectuer le chiffrement. Les premières machines de l'après-guerre ne faisaient qu'une alternance de un ou deux chiffrements produits. Pendant la Seconde Guerre mondiale, les machines ENIGMA basées sur ces idées de chiffrement pouvaient atteindre des séquences de 6 ou 7 alternances de chiffrement produit. Les chiffrements symétriques actuels peuvent être vus comme les descendants de ces systèmes avec une plus grande complexité des outils utilisés pour faire du chiffrement produit.

II. Chiffrement à clé secrète

Deux grands types de chiffrement à clé secrète (ou symétrique) sont utilisés aujourd'hui : le chiffrement par blocs (par exemple les standards D.E.S. ou A.E.S.) et le chiffrement à flot (en généralisant le procédé de Vernam vu plus haut).

Les chiffrements par blocs que nous présentons ci-dessous sont généralement plus lents (mais plus sûrs) que les chiffrements à flot. Les chiffrements à flot sont utilisés dans des protocoles demandant un procédé léger, par exemple dans le téléphone portable (chiffrement A5/1) ou dans la norme Bluetooth (chiffrement E_0).

II.1. Chiffrement par blocs

Dans cette partie, nous travaillons systématiquement sur le corps fini à deux éléments $\mathbb{F}_2$ (et ses extensions). Les messages seront donc considérés comme étant des suites de bits. L'intérêt est d'utiliser la propriété $1 \oplus 1 = 0$ dans $\mathbb{F}_2$, qui permettra les déchiffrements.

II.1.1. Définitions générales

Étant donné un message M à chiffrer, le principe du chiffrement par blocs sera de le couper en blocs de taille n et de chiffrer ces blocs successivement. On se donne un sous-ensemble $K \subset \mathbb{F}_2^l$ qui constituera l'ensemble des clés secrètes.

Définition 1.1. (Chiffrement par blocs) *Soit $K \subset \mathbb{F}_2^l$. Un chiffrement par bloc est une fonction $E : \mathbb{F}_2^n \times K \to \mathbb{F}_2^n$ telle que, pour tout $k \in K$, la fonction $E_k() : \mathbb{F}_2^n \to \mathbb{F}_2^n, m \mapsto E(m,k)$ est inversible. On dit que k est la clé du chiffrement, et K est l'ensemble des clés.*
La fonction inverse de $E_k()$ est notée $D_k()$: pour chaque bloc $m \in \mathbb{F}^n$ du message, on a

$$D_k(E(m,k)) = m.$$

Remarque. La fonction E (E comme *Encryption* en anglais) est une fonction de chiffrement. La fonction D (D comme *Decipher* en anglais) est une fonction de déchiffrement.

Comme nous l'avons vu, on appelle *chiffrement produit* un chiffrement par blocs qui combine plusieurs transformations élémentaires. Ces transformations peuvent être des substitutions, des transpositions ou des opérations arithmétiques.

Définition 1.2. (Chiffrement itératif par blocs) *Soit F une fonction de chiffrement produit.*
Un chiffrement itératif par blocs est un chiffrement résultant de l'application itérée de F à chaque bloc de clair. Chaque application de F est appelée un tour.

Remarque. Dans un chiffrement itératif par blocs, on peut aussi utiliser, pour chaque tour, des sous-clés K_i engendrées à partir de la clé de chiffrement K initiale.
Pour rendre le chiffrement inversible (donc déchiffrable), F doit être inversible.

Nous allons expliquer les deux types de schémas référants pour le chiffrement par blocs : le schéma de Feistel (sur lequel est basé le standard D.E.S.) et le schéma de chiffrement substitutions/permutations de Shannon.

II.1.2. Schéma de Feistel

La classe de schéma introduite par H. Feistel en 1970 permet de construire facilement des algorithmes de chiffrement par blocs.

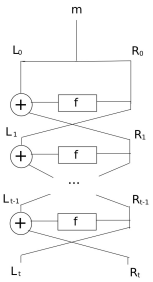

Fig. 1.1. Schéma de Feistel

Définition 1.3. (Schéma de Feistel) *Un schéma de Feistel est un chiffrement itératif à* t *tours qui prend en entrée un bloc de clair de* $2n$ *bits* (L_0, R_0) *et renvoie en sortie un bloc chiffré* (R_t, L_t) *de taille* $2n$.
Soient f *un chiffrement produit de* $\mathbb{F}_2^n \to \mathbb{F}_2^n$ *et* $K_1, \ldots, K_t$ *des sous-clés dérivées d'une clé secrète* K.
À chaque tour, le schéma transforme (L_{i-1}, R_{i-1}) *en* (L_i, R_i) *tels que*

$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus f(R_{i-1}, K_i). \end{cases}$$

Le schéma de Feistel est un chiffrement par blocs : la fonction F est celle qui transforme le couple (L_i, R_i) en le couple (L_{i+1}, R_{i+1}).

Proposition 1.4. *Pour tout couple* (L_i, R_i)*, on peut retrouver le couple* (L_{i-1}, R_{i-1}) *par les opérations* $R_{i-1} = L_i, L_{i-1} = R_i \oplus f(L_i, K_i)$. *Le schéma de Feistel est inversible quel que soit le nombre de tours.*

Preuve. Supposons que l'on connaisse (L_i, R_i), alors on a bien, par définition, $L_i = R_{i-1}$, qui permet de retrouver R_{i-1} et $R_i \oplus f(L_i, K_i) = L_{i-1} \oplus f(R_{i-1}, K_i) \oplus f(L_i, K_i) = L_{i-1} \oplus f(L_i, K_i) \oplus f(L_i, K_i) = L_{i-1}$, puisque $1 \oplus 1 = 0$. Par ce procédé, à partir de (L_t, R_t), on est capable de retrouver, tour après tour, les différents (L_i, R_i) jusqu'au clair (L_0, R_0).
On peut remarquer que l'on peut réutiliser le circuit utilisé pour le chiffrement, mais en partant de (R_r, L_r) au lieu de (L_0, R_0) et en inversant l'ordre des sous-clés K_i. ■

Exemple 1.5. Considérons un exemple simple (et académique). Nous prenons $2n = 8$. Pour calculer $f(x, k)$, on commence par faire une rotation de x (tous les bits sont décalés circulairement d'une position vers la droite), puis on additionne bit à bit la clé de tour K_i ; enfin, on applique l'opération $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ définie par

$$S(x_1, x_2, x_3, x_4) = (x_1x_2 \oplus x_3, x_1x_2x_3 \oplus x_4, x_2x_3 + 1, x_1x_3x_4 + x_2).$$

On a schématiquement

$$f_{K_i} : x \xrightarrow{Rot} \xrightarrow{\oplus K_i} \xrightarrow{S} f(x, K_i).$$

On crée les clés de tour K_i par l'opération suivante : à partir de la clé de chiffrement $K = K_0$, on dérive K_{i+1} à partir de K_i en faisant une rotation de K_i d'une position vers la gauche.

On peut remarquer que l'opération f est bien un chiffrement produit puisqu'il rassemble une opération de transposition (la permutation circulaire) qui amène de la diffusion et des

opérations de confusion (comme ajouter la clé de tour K_i), et surtout l'action de S qui est non linéaire.

On va maintenant faire tourner l'algorithme de chiffrement pour le clair $M = (10101001)$ sur deux tours, en prenant comme clé de chiffrement $K = (1011)$.

On sépare M en deux morceaux : $L_0 = (1010)$ et $R_0 = (1001)$. La clé de tour K_1 est le décalage de $K_0 = K$ d'une position vers la gauche, soit $K_1 = (0111)$. On calcule alors l'action de f. On commence par décaler R_0 d'une position vers la droite, soit (1100) ; on ajoute K_1 pour obtenir (1011), puis on applique la fonction S et l'on obtient $(1.0 \oplus 1, 1.0.1 \oplus 1, 0.1+1, 1.1.1 \oplus 0) = (1, 1, 1, 1)$. Finalement, $f(R_0, K_1) = (1111)$.

On calcule $R_1 = L_0 \oplus f(R_0, K_1) = (1010) \oplus (1111) = (0101)$, puis $L_1 = R_0 = (1001)$. À partir de $(L_1, R_1) = (1001, 0101)$, on peut alors recommencer pour calculer (L_2, R_2). La clé K_2 est le décalage de K_1 d'une position vers la gauche, soit $K_2 = (1110)$; l'application f donne $f(R_1, K_2) = (0011)$, puis on obtient $(L_2, R_2) = (0101, 1010)$.

Par cet algorithme de chiffrement sur deux tours, par la clé $K = (1011)$, le chiffré de $M = (10101001)$ est donc $C = (01011010)$.

Test 1.4.

Déchiffrer le chiffré de l'exemple avec la clé de chiffrement.

Test 1.5.

Considérons un schéma de Feistel où f est composé uniquement d'opérations linéaires et de l'addition d'une sous-clé de tour. Montrer que, dans ce cas, il est très facile de déchiffrer, sans connaître la clé de chiffrement, à partir de quelques couples clairs/chiffrés.

La sécurité d'un schéma de type Feistel repose dans la partie non linéaire (dans l'exemple, la transformation S) de la fonction f.

II.1.3. Le cas du D.E.S.

Le D.E.S. est un algorithme de chiffrement symétrique basé sur le schéma de Feistel. Historiquement, le bureau national des standards américains (N.B.S.), qui allait devenir le National Institute of Standards and Technologies (N.I.S.T.), a lancé un appel à propositions pour un standard de schéma de chiffrement. La compagnie I.B.M. a posé sa candidature en 1974 avec un système basé sur un schéma de Feistel appelé LUCIFER. Ce système de chiffrement a été envoyé pour analyse à la N.S.A. qui l'a modifié légèrement en 1975 ; c'est ce schéma qui a été standardisé en 1977 sous le nom de D.E.S. comme libre de droits.

Le D.E.S. est un chiffrement par blocs avec $n = 64$, 16 tours et une clé de 64 bits formée de 8 blocs de 8 bits dont 1 bit de chaque bloc est utilisé pour faire un contrôle de parité (pour détecter une erreur potentielle), ce qui donne en fait seulement 56 bits utiles pour la clé.

Il repose sur un schéma de type Feistel sur 16 tours (légèrement modifié) qui commence par une permutation sur les 64 bits du message et finit (par symétrie) sur la permutation inverse. La fonction f du D.E.S. de $\mathbb{F}_2^{32}$ dans $\mathbb{F}_2^{32}$ est composée d'une fonction E qui étend les 32 bits de R_i en 48 bits d'une manière linéaire en les permutant, de l'addition d'une sous-clé de tour et de 8 boîtes S non-linéaires (les fameuses *boîtes S* du D.E.S.) qui sont chacunes de $\mathbb{F}_2^6$ dans $\mathbb{F}_2^4$ et transforment les 48 bits en 32 bits. La dérivation des sous-clés se fait par des décalages successifs de une ou deux positions à chaque tour à partir de la clé de chiffrement.

Pour pallier la faiblesse relative du D.E.S. due à la taille de la clé, qui a été très vite pointée du doigt, il a été proposé une variation utilisant le D.E.S. comme brique de base : le triple D.E.S. ou TD.E.S. Le triple D.E.S. utilise deux clés D.E.S. au lieu d'une ; il se définit par

$$TD.E.S._{K_1,K_2}(m) = D.E.S._{K_1}(D.E.S._{K_2}^{-1}(D.E.S._{K_1}(m))),$$

le déchiffrement se faisant par

$$TD.E.S._{K_1,K_2}^{-1}(c) = D.E.S._{K_1}^{-1}(D.E.S._{K_2}(D.E.S._{K_1}^{-1}(C))).$$

La taille de la clé passe alors de 56 bits à 112 bits.

Pourquoi utiliser un triple D.E.S. plutôt qu'un double D.E.S. ? Car le double D.E.S. est vulnérable à une attaque très classique, l'attaque par le milieu. Supposons que l'on définisse le double D.E.S. par $DD.E.S._{K_1,K_2}(M) = D.E.S._{K_2}(D.E.S._{K_1}(M)) = C$. On introduit alors une valeur intermédiaire R de la façon suivante :

$$M \overset{D.E.S._{K_1}}{\longrightarrow} R \to \overset{D.E.S._{K_2}}{\longrightarrow} C.$$

L'attaque est une attaque à clair connu où l'on suppose que l'on connaît quelques couples clair/chiffré (M_i, C_i) pour la même clé (K_1, K_2). Pour le couple (M_1, C_1), on va créer une table de toutes les clés possibles K_1' indicée par toutes les valeurs obtenues de $D.E.S._{K_1}(M_1)$. Cette table a donc 2^{56} entrées et peut être parcourue très facilement puisqu'elle est indicée (par exemple par la valeur de la séquence de 64 bits $D.E.S._{K_1}(M_1)$, considérée comme un entier entre 0 et $2^{64}-1$ classés par ordre croissant).

On va alors considérer pour C_1 toutes les valeurs possibles de K_2 et calculer $D.E.S._{K_2}^{-1}(C_1)$; pour chaque clé K_2', on trouve un $R_1 = D.E.S._{K_2'}^{-1}(C_1)$ auquel on va faire correspondre une clé K_1' telle que $D.E.S._{K_1'}(M_1) = R_1$:

$$M_1 \overset{D.E.S._{K_1'}}{\longrightarrow} R_1 \overset{D.E.S._{K_2'}^{-1}}{\longleftarrow} C_1'.$$

On peut donc alors créer une liste L_1 des couples de clés (K_1', K_2') indicée par exemple sur la valeur de K_1 telle que pour tous ces couples, on a $D.E.S._{K_2}(D.E.S._{K_1}(M_1)) = C_1$. Il est clair que la bonne valeur du couple recherché (K_1, K_2) fait partie de l'ensemble des couples de la liste L_1. On procède de même pour le couple (M_2, C_2) et l'on obtient une liste L_2. On calcule alors l'intersection des listes L_1 et L_2 ; on sait que le couple de la clé recherchée appartient à l'intersection. De plus, comme la probabilité qu'un K_1' dans la liste L_1 soit associé au même K_2' est très faible (puisque le chiffrement de M_i par K_1' est indépendant pour la construction du déchiffrement de C_i par K_2'), l'intersection est de taille très petite et l'on peut alors tester tous les couples de clés obtenus dont le nombre est très petit.

La complexité en nombre d'opérations élémentaires de l'attaque est de l'ordre de 2^{56} opérations fois une constante pas trop grande. En effet, la mise en place des listes est de l'ordre de $56 \cdot 2^{56}$ (où 56 est le log en base 2 de 56 et le coût d'un parcours en arbre de la liste). La création de la liste L_1 est aussi de cet ordre-là puisqu'il faut parcourir une table de taille 2^{56} et aller prendre une valeur dans une table indicée de taille 2^{56}. L'intersection des listes L_1 et L_2 se fait en parcourant les deux listes indicées par K_1' et en regardant si les K_2' associées sont identiques. Le coût de l'opération en mémoire (le coût de stockage des listes) est aussi de l'ordre de $2^{56} \cdot 56$ (à un petit facteur près). On obtient donc une attaque en $K.2^{56}$ en mémoire et en nombre d'opérations élémentaires, soit bien mieux que 2^{112}, auquel on pouvait s'attendre a priori. Cela donne une sécurité pour le double D.E.S. trop faible.

Test 1.6.

Montrer que le coût de cette attaque, où l'on enchaîne deux chiffrements avec des clés de taille n bits, est en $\mathcal{O}(n2^{n/2})$.

Test 1.7.

On considère le cas du $D.E.SV_{K_1,K_2}(M) = K_1 \oplus D.E.S._{K_2}(M)$ et $D.E.SW_{K_1,K_2}(M) = D.E.S._{K_1}(M \oplus K_2)$. Montrer que l'on peut mener une attaque par le milieu pour ces deux systèmes. Est-ce que le $D.E.SX_{K_1,K_2,K_3}(M) = K_1 \oplus D.E.S._{K_2}(M \oplus K_3)$ est sensible à cette attaque ?

Test 1.8.

On considère le schéma utilisant trois D.E.S., mais deux clés k_1 et k_2 : $D.E.S._{k_1}(D.E.S._{k_2}(D.E.S._{k_2}(M)))$, (l'ordre des clés a été modifié par rapport au TD.E.S. classique). Montrer comment on peut attaquer ce schéma avec une attaque de type attaque par le milieu.

II.1.4. Schémas de type substitution/permutation de Shannon

Pour ce type de schéma, un tour est composé d'une opération de type substitution inversible qui assure la confusion et d'une permutation qui permet la diffusion.

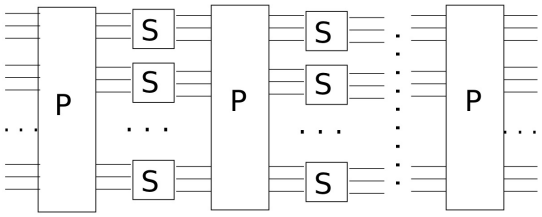

Fig. 1.2. Schéma de substitution/permutation de Shannon

Exemple 1.6. Plutôt qu'une définition formelle, considérons un exemple (académique) simple. Considérons le corps $\mathbb{F}_{16}$ à 16 éléments construit comme $\mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4+X+1)$. Un élément $x = (x_1, x_2, x_3, x_4)$ de $\mathbb{F}_2^4$ est associé à l'élément $p_x \overset{\text{déf}}{=} x_1 + x_2X + x_3X^2 + x_4X^3 \in \mathbb{F}_{16}$. Un tour est composé :
- d'une rotation P (permutation circulaire d'une position vers la droite) ;
- d'une application inversible $F(x, K_i)$ qui assure la confusion de la façon suivante : à $p_x \in \mathbb{F}_{16}$ on associe $p_x^{-1} \in \mathbb{F}_{16}$ (avec la convention que l'inverse de 0 est 0), puis on prend l'image de p_x^{-1} dans $\mathbb{F}_2^4$ et l'on ajoute (par addition XOR modulo 2) la clé de tour K_i (construite comme dans notre exemple de schéma de Feistel).

Calculons le chiffré du message $M = (1110)$ pour la clé $K = (1010)$ par ce procédé.
On commence par appliquer P à M pour obtenir (0111), puis on calcule (voir la section III) l'inverse de $X + X^2 + X^3$ modulo $X^4 + X + 1$ qui donne $1 + X$, soit (1100). On ajoute alors la clé $K_1 = (0101)$ et l'on obtient (1001) comme valeur du chiffré après un tour. On réitère pour un second tour avec $K_2 = (1010)$ en appliquant P et $F(K_2, x)$ pour obtenir comme chiffré $C = 1101$.

Pour le déchiffrement, on considère les opérations inverses en additionnant la clé K_2, puis en calculant l'inverse et enfin en appliquant la permutation inverse P^{-1}, une rotation d'une position vers la gauche. Et on recommence pour le premier tour avec clé de tour K_1.

Test 1.9. Déchiffrer le chiffré de l'exemple avec la clé de chiffrement.

II.1.5. Le standard A.E.S.

En 1996, le N.I.S.T. a lancé une compétition pour remplacer le standard D.E.S. Douze algorithmes ont été proposés. Une première sélection en a retenu 5 ; l'algorithme Rijndael proposé par John Daemen et Vincent Rijmen a été sélectionné et est devenu en 2000 le nouveau standard A.E.S.
Ce schéma est un schéma de type réseau de substitution/permutation de Shannon. Il a comme taille de bloc pour le chiffré 128 et une taille de clé variable (128, 192 ou 256) : on parle alors d'A.E.S.-128, A.E.S.-192 ou A.E.S.-256. Un tour d'A.E.S. est constitué de quatre transformations élémentaires : **SubBytes, ShiftRows, MixColumns** et **AddRoundKey**. La structure de l'A.E.S. est basée sur l'identification des octets à des éléments du corps $\mathbb{F}_{2^8} = \mathbb{F}_{256}$ à 256

éléments.
Un chiffré intermédiaire de 128 bits est considéré comme une matrice 4×4 composée d'octets. Les transformations ShiftRows et MixColumns sont linéaires et agissent sur les lignes et les colonnes ; la transformation SubBytes utilise l'inversion dans $\mathbb{F}_{256}$ et constitue la seule partie non linéaire de l'algorithme. Enfin, la fonction AddRounKey ajoute la clé de tour. Le nombre de tours effectués dépend de la taille de la clé et peut valoir $10, 12$ ou 14 tours.

II.1.6. Modes opératoires

Pour le chiffrement, un message est traité par blocs de taille n, mais la manière d'arranger les blocs chiffrés entre eux peut varier. Voyons les quatres principaux modes opératoires : le mode ECB (de l'anglais *Electronic Code Book*), le mode CBC (pour *Cipher Block Chaining*), le mode OFB (pour *Output Feedback mode*) et enfin le mode CFB (pour *Cipher Feedback mode*).
Dans la suite, on considère une clé k, une fonction de chiffrement $E_k()$ et sa fonction de déchiffrement associée $D_k()$, un message m divisé en t blocs $m_1, \ldots, m_t$ de longueur n (quitte à faire du remplissage avec des zéros sur le dernier bloc si la longueur n'est pas exactement un multiple de n), et éventuellement des chiffrés intermédiaires c_i.

Fig. 1.3. Les modes opératoires ECB et CBC

Le *mode ECB* est le mode le plus simple : chaque bloc m_i est chiffré indépendamment. Pour $1 \leqslant i \leqslant t$, on a pour le chiffrement $c_i = E_k(m_i)$ et pour le déchiffrement $m_i = D_k(c_i)$.
L'avantage de ce mode est qu'il n'y a pas de propagation d'erreurs : si un bit d'un bloc est modifié pour une raison quelconque, alors, au niveau du déchiffrement, seul un bloc sera modifié. En revanche, un bloc clair donné m_i sera toujours chiffré de la même façon, quel que soit son emplacement dans texte, et peut donc éventuellement être reconnu par des attaques à clair connu (ou des attaques statistiques).

Le *mode CBC* est aussi un mode très utilisé. Il consiste à lier un bloc chiffré c_i avec le bloc précédent c_{i-1} : on part d'une valeur initiale $c_0 = VI$ (pour Valeur Initiale) puis, pour $1 \leqslant i \leqslant t$, on pose $c_i = E_k(m_i \oplus c_{i-1})$.
Pour le déchiffrement, on a $c_0 = VI$, $m_i = c_{i-1} \oplus D_k(c_i)$.
Ce mode a un avantage : le chiffrement d'un bloc m_i va dépendre de ce qui le précède. Cela rend plus difficile des attaques par dictionnaire où l'on essaierait de reconnaître des bouts de texte ; en revanche, en cas d'erreur de transmission sur le chiffré, il y aura une très forte propagation d'erreur pour le déchiffrement.

Le *mode OFB* permet d'utiliser un schéma de chiffrement par blocs pour faire du chiffrement à flot en simulant un générateur pseudo-aléatoire à partir de la fonction de chiffrement.
On pose $z_0 = VI$, puis on crée la séquence d'aléa $z_i = E_k(z_{i-1})$. Le chiffrement devient

$$c_i = m_i \oplus z_i.$$

Pour déchiffrer, il faut connaître $z_0 = VI$ qui doit changer pour chaque message à chiffrer. En particulier, comme tout chiffrement par blocs peut être transformé en chiffrement à flot de cette manière, les algorithmes de chiffrement à flot doivent (pour avoir un intérêt) fonctionner plus vite que l'A.E.S.

Enfin, dans le cas du *mode CFB*, c'est le chiffré qui va engendrer l'aléa

$$c_0 = VI, z_i = E_k(c_{i-1}) \text{ pour } \ c_i = m_i \oplus z_i.$$

Pour le déchiffrement, on a $c_0 = VI$, $m_i = c_i \oplus E_k(c_{i-1})$.

II.2. Chiffrement à flot

Le chiffrement à flot consiste à utiliser le chiffrement de Vernam (ou encore masque jetable) que l'on a vu dans le rappel historique avec un générateur pseudo-aléatoire.

Si l'on appelle s_i la séquence pseudo-aléatoire et m_i les bits du message, alors les bits chiffrés c_i associés s'obtiennent par $c_i = s_i \oplus m_i$.
Pour le déchiffrement, on note que $m_i = c_i \oplus s_i = m_i \oplus s_i \oplus s_i$.

On a vu qu'en utilisant le mode opératoire OFB, on pouvait faire un générateur pseudo-aléatoire à partir d'un chiffrement par blocs. L'intérêt du chiffrement à flot par rapport au chiffrement par blocs est qu'il demande en général moins de ressources pour être implémenté puisque l'on a juste besoin d'un générateur pseudo-aléatoire, quand le chiffrement par blocs nécessite de faire un nombre de tours important.

En général, le générateur est connu et seule l'initialisation du générateur est inconnue. L'initialisation est le secret partagé par les deux parties. Elle constitue donc la clé secrète du système. L'intérêt du chiffrement à flot est qu'il repose sur un générateur pseudo-aléatoire dont on peut espérer que la mise en œuvre soit moins coûteuse que le chiffrement par blocs.

II.2.1. Registres à décalage linéaires (L.F.S.R.)

Les registres à décalage linéaire, que nous étudions maintenant, nous fournissent un exemple de famille de générateurs pseudo-aléatoires avec de bonnes propriétés statistiques et très rapides. Nous allons en voir les principes (et les limitations).

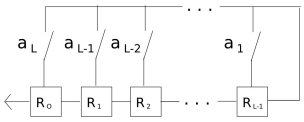

Fig. 1.4. Registre à décalage (L.F.S.R.)

Définition 1.7. (Registre à décalage, L.F.S.R.) *Un registre à décalage de longueur* L *est constitué de* L *cases mémoire* $R_0, R_1, \ldots, R_{L-1}$*, chacune pouvant contenir* 0 *ou* 1 *(un bit) et ayant une sortie et une entrée, ainsi que d'une horloge contrôlant le mouvement des données.*
À chaque top d'horloge :
- *un bit de rétroaction est calculé par combinaison de bits des cases* 0 *à* $L-1$ *;*
- *le contenu de la case* 0 *sort du registre pour former la séquence de sortie ;*
- *le contenu de la case* i *passe dans la case* $i-1$ *pour* $1 \leqslant i \leqslant L-1$ *;*
- *la case* $L-1$ *est remplacée par le bit de rétroaction.*

La sortie du générateur est aussi appelée suite chiffrante. Si la combinaison est un simple X0R (addition modulo 2), on dit que la rétroaction est linéaire et l'on parle de registre à rétroaction linéaire (*Linear Feedback Shift Register* - L.F.S.R. en anglais). Dans le cas d'une rétroaction linéaire donnée par des coefficients binaires a_i, si l'on appelle s_i la séquence de sortie, on a alors la relation de récurrence suivante entre les bits de sortie :

$$s_j = a_1 s_{j-1} \oplus a_2 s_{j-2} \oplus \ldots \oplus a_L s_{j-L}, \text{ pour } j \geq L.$$

La séquence $(s_0, s_1, \ldots, s_{L-1})$ est appelée initialisation du générateur. Le polynôme $C(x) = 1 + a_1 x + \ldots + a_L x^L$ est appelé *polynôme de rétroaction linéaire* du générateur.

Exemple 1.8. Prenons par exemple $L = 3$ et $c(x) = 1 + x + x^3$, avec une initialisation $(s_0, s_1, s_2) = (0, 1, 1)$. L'état des registres est donné par $(0, 1, 1)$. La première itération calcule $s_4 = s_2 a_1 \oplus s_1 a_2 \oplus s_0 a_3 = 1 \oplus 0 \oplus 0 = 1$. On fait alors sortir le premier bit du registre $s_0 = 0$, puis on décale les 2 derniers bits du registre et l'on remplace la dernière case du registre par le bit de rétroaction calculé $s_4 = 1$. On obtient alors comme nouvel état du registre $(1, 1, 1)$. Après une nouvelle itération, on fait sortir $s_1 = 1$ et l'état du registre devient $(1, 1, 0)$. Après quatorze itérations, on obtient $s_0 s_1 \dots s_{10} = 01110100111010$ et l'état du registre est $(0, 1, 1)$.

On peut remarquer dans l'exemple précédent que les termes de la sortie semblent se répéter à partir de s_8. Ce n'est pas un hasard.

Lemme 1.9. *Soit R un registre à rétroaction linéaire de longueur L. Si son polynôme de rétroaction $C(x)$ est de degré L, alors la séquence de sortie est périodique, de période au plus $2^L - 1$.*

Preuve. On remarque tout d'abord que l'initialisation nulle induit une séquence identiquement nulle. Le registre étant composé de L cases mémoire pouvant valoir 0 ou 1, il y a donc $2^L - 1$ états distincts non nuls possibles au maximum. Si $C(x)$ a pour degré L, chaque même état donné donne lieu à la même séquence en sortie, et donc la suite de sortie est périodique de période au plus $2^L - 1$. Le cas où $C(x)$ n'est pas de degré L correspond au cas où certains bits n'entrent pas dans la récurrence. Dans ce cas, la suite est simplement ultimement périodique (périodique au bout d'un certain nombre d'itérations). ■

On peut préciser un peu plus le caractère périodique en fonction du polynôme de rétroaction $C(x)$. Rappelons que l'on dit que C est un polynôme primitif s'il est irréductible et si, en faisant l'identification $\mathbb{F}_{2^L} = \mathbb{F}_2[x]/(C(x))$, l'élément x engendre $\mathbb{F}_{2^L}^*$.

Théorème 1.10. *Soit R un registre de longueur L avec polynôme de rétroaction linéaire $C(x)$ de degré L.*

1) Si $C(x)$ est irréductible sur $\mathbb{F}_2[x]$, alors chacun des $2^L - 1$ états non nuls possibles produit une séquence de sortie de période égale au plus petit entier N tel que $C(x)$ divise $x^N - 1$. De plus, N est toujours un diviseur de $2^L - 1$.

2) Si $C(x)$ est un polynôme primitif, alors, pour toute initialisation non nulle, R produit une séquence de période maximale $2^L - 1$.

Preuve. Admis. ■

Test 1.10.

Engendrer la suite chiffrante pour $L = 4$, $C(x) = 1 + x + x^4$ et une initialisation $(s_0, s_1, s_2, s_3) = (1, 0, 0, 1)$. Quelle est sa période ?

Test 1.11.

Même question avec $C(x) = x^4 + x^3 + x^2 + 1$.

Les registres à rétroaction linéaire peuvent donc produire des séquences avec des périodes très longues, par exemple 2^{128} bits à partir de 128 cases mémoire. Ils sont très simples a implémenter sur des circuits ; ils ont, de plus, de très bonnes propriétés statistiques, mais aussi quelques faiblesses que nous allons voir maintenant.

II.2.2. Attaque linéaire sur les registres à rétroaction linéaire

Pour un registre à décalage linéaire, on peut imaginer qu'à la fois l'initialisation et le polynôme de rétroaction ne soient pas connus. On peut facilement retrouver $C(x)$ avec une attaque à clair connu.

Supposons qu'une suite chiffrante produite par le registre soit utilisée pour chiffrer un message ; supposons, de plus, que nous connaissions à la fois une séquence suffisamment longue (au moins $2L$) du clair et du chiffré associé. Pour un certain n, on connaît alors $m_{n+1}, m_{n+2}, \dots, m_{n+2L}$ et $c_{n+1}, c_{n+2}, \dots, c_{n+2L}$. En sommant c_i et m_i (modulo 2), on peut obtenir la suite chiffrante $s_{n+1}, \dots, s_{n+2L}$. On ne connaît pas les coefficients a_i de $C(x)$, mais on sait qu'il existe une relation linéaire entre les s_i, donnée par les a_i. On peut alors se ramener à résoudre un système linéaire avec L inconnues (les a_i) et L équations données par

$$s_{n+j} = a_1 s_{n+j-1} \oplus a_2 s_{n+j-2} \oplus \dots \oplus a_L s_{n+j-L} \quad \text{pour} \quad L+1 \leqslant j \leqslant 2L.$$

On résout ce système. Une fois que l'on a obtenu $C(x)$, on peut alors engendrer toute la suite chiffrante, avant et après les bits connus, et déchiffrer tout le message [1].

Test 1.12.

On intercepte un message chiffré avec un système de chiffrement à flot produit par une suite chiffrante récurrente de type L.F.S.R. Le message binaire intercepté est 0000011011110010. On sait d'autre part que les six premiers bits du message clair sont 110101. On admet que la longueur L du registre R est au plus 3. Déchiffrer le message.

II.2.3. Variations sur les registres à décalage linéaire

Comme les registres linéaires ont à la fois de bonnes propriétés de périodicité et sont très faciles à implémenter, on voudrait les utiliser en évitant les faiblesses ci-dessus. Il s'agit alors d'en combiner plusieurs pour induire de la non-linéarité.

Le principe des *registres combinés* est d'utiliser plusieurs registres $R_1, R_2, \cdots, R_n$ différents et produisant chacun, à chaque top d'horloge, un bit de sortie x_i. Les bits de sortie sont alors utilisés avec une fonction booléenne $f : \mathbb{F}_2^n \to \mathbb{F}_2$, $f(x_1, \dots, x_n) = z$.

Exemple 1.11. Le générateur de Geffe a été utilisé dans les années 1970 avec trois registres et une fonction $f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$.

En pratique, on utilise plutôt maintenant un nombre de registres plus important (de l'ordre de 7 à 9), mais le calcul de la fonction booléenne prend alors plus de temps.

Ce type de schéma peut être attaqué par des attaques à clair connu (c'est-à-dire que l'on connaît un extrait de suite chiffrante), où l'on cherche à retrouver les initialisations de certains registres à partir d'un biais statistique entre la sortie $z = f(x_1, \dots, x_n)$ et une sortie x_i d'un registre particulier (vu dans ce contexte comme une *corrélation* entre les deux valeurs). On parle alors d'*attaque par corrélation*.
Par exemple, dans le cas du générateur de Geffe, on a 3 registres puis on applique $f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$. On remarque que $z = f(x_1, x_2, x_3) = x_2(x_1 + x_3) + x_3$, soit $z = x_3$ lorsque $x_2 = 0$ ou $x_1 = x_3$. Si l'on considère des sorties aléatoires et indépendantes, la probabilité que $x_2 = 0$ est $1/2$ et la probabilité que $x_1 = x_3$ est aussi égale à $1/2$, donc la probabilité que $z = x_3$ est $3/4$, en moyenne.

1. Si on ne connaît pas L (mais on a vu que L restait relativement petit et on peut donc éventuellement tous les essayer), l'algorithme de Berlekamp-Massey permet de retrouver la plus petite longueur d'une récurrence engendrant une suite donnée et peut s'appliquer dans ce cadre-là.

L'attaque consiste alors à retrouver la valeur du registre R_3 lié à x_3. Si le registre L_3 a pour longueur l, on essaie toutes les valeurs possibles du registre L_3 (s'il n'est pas trop grand) et, pour chacune des valeurs du registre, on engendre (puisque l'on considère que le polynôme de rétroaction est connu) une suite de valeurs de x_3 de longueur m suffisamment grande. On compare alors cette suite pour les l valeurs essayées de L_3 à la suite des z. Si les l valeurs du registre correspondent aux bonnes valeurs de x_3, comme il existe un biais entre z et x_3, on observera ce biais entre les deux suites qui seront égales en moyenne pour 3 valeurs sur 4. Si en revanche on ne teste pas les bonnes valeurs de L_3, il n'y a pas de biais a priori et l'on obtient une probabilité de l'ordre de $1/2$ en moyenne. On peut ainsi retrouver la valeur du registre L_3, puis on recommence avec les autres. On propose une telle attaque dans les exercices.

Une variante consiste à utiliser des *registres filtrés*. Dans ce cas, on considère un générateur linéaire d'une longueur donnée (comme $L = 128$ ou 256 cases mémoire) et l'on choisit n cases mémoire parmi ces L, dont on prend les contenus $x_1, \dots, x_n$ à chaque tour pour leur appliquer une fonction booléenne f pour donner la sortie $z = f(x_1, \dots, x_n)$.

III. Rappels d'arithmétique

Avant de poursuivre vers la cryptographie à clé publique, il nous faut rappeler quelques résultats classiques en arithmétique et théorie des nombres utilisés par la suite. On trouvera les preuves et un cours détaillé dans les chapitres 4 et 5 du tome *Mathématiques L2*, Pearson Education, 2007, ou dans le tome *Algèbre L3* chez le même éditeur. Un rappel sur les corps finis est proposé dans le chapitre 7 (codes correcteurs) de ce tome.

III.1. Inversion modulo N et calcul modulaire

Soit N un entier naturel non nul. Un élément a de $\mathbb{Z}/N\mathbb{Z}$ est inversible s'il existe un entier b tel que $a.b = 1 \pmod N$.

Proposition 1.12. *Soit $a \in \mathbb{Z}/N\mathbb{Z}$, alors a est inversible si et seulement si a est premier avec N.*

Preuve. Si a est premier avec N, alors, d'après le théorème de Bézout, il existe u et v dans $\mathbb{Z}$ tels que $au + vN = 1$; l'entier u vérifie donc $au = 1 \pmod N$. Réciproquement, si a est inversible dans $\mathbb{Z}/N\mathbb{Z}$, alors il existe un entier u tel que $\bar{a}\bar{u} = \bar{1}$ dans $\mathbb{Z}/N\mathbb{Z}$, soit $au = 1 \pmod N$. Il existe donc $k \in \mathbb{Z}$ tel que $au = 1 + kN$, soit $au - kN = 1$: d'après Bézout, cela montre que a et N sont premiers entre eux. ■

Cette démonstration montre que, si a est inversible modulo N, son inverse peut s'obtenir par l'algorithme d'Euclide étendu [2] qui calcule $d = pgcd(a, N)$ et des entiers u et v tels que $au + Nv = d$ (avec $|u| < N$). Quand $d = 1$, l'inverse de a dans $\mathbb{Z}/N\mathbb{Z}$ est u modulo N.

Test 1.13. Calculer les inverses de 8 modulo 13 et 7 modulo 15 avec l'algorithme d'Euclide étendu.

Remarque. Dans le cas où N est un nombre premier, $\mathbb{Z}/N\mathbb{Z}$ est un corps et tous les éléments non nuls sont inversibles.

2. Voir par exemple les chapitres 4 et 5 du tome *Mathématiques L2*, ou les chapitres 6 et 7 de ce tome.

Si x est un élément de $\mathbb{Z}/n_1n_2\mathbb{Z}$, on utilisera par la suite le fait que x peut s'écrire de manière unique sous la forme $x = an_1 + bn_2$, avec $0 \leqslant a \leqslant n_2 - 1$ et $0 \leqslant b \leqslant n_1 - 1$; cela permettra de ramener des calculs modulo n_1n_2 à des calculs séparés modulo n_1 et n_2. On trouve ce résultat dans le théorème suivant.

Théorème 1.13. (Théorème des restes chinois) *Soient $n_1, n_2, \ldots, n_r$, r entiers deux à deux premiers entre eux, soit le produit $n = n_1.n_2 \ldots n_r$ et soient $x_1, \ldots, x_r$ une suite d'entiers, alors il existe un unique entier $x \pmod n$ tel que*

$$x \equiv x_1 \pmod{n_1},\ x \equiv x_2 \pmod{n_2}, \ldots, x \equiv x_r \pmod{n_r}.$$

Preuve. Ce théorème peut se montrer par récurrence sur r. Pour $r = 2$, supposons que $\mathrm{pgcd}(n_1, n_2) = 1$. Il existe alors u et v tels que $un_1 + vn_2 = 1$. Posons $x \overset{\text{déf}}{=} x_1un_1 + x_2vn_2$; alors $x \equiv x_1un_1 \pmod{n_2}$. Mais, puisque l'on a aussi $un_1 \equiv 1 \pmod{n_2}$, on a bien $x \equiv x_1 \pmod{n_2}$ et de même, $x \equiv x_2 \pmod{n_1}$. Il existe donc une solution à ce système de congruence. Réciproquement, si y vérifie les mêmes congruences sur les x_i, alors $x \equiv y \equiv x_1 \pmod{n_1}$ et donc $(x - y) \equiv 0 \pmod{n_1}$. Il en découle que n_1 divise $x - y$ et de même, n_2 divise $x - y$ et, comme n_1 et n_2 sont premiers entre eux, $n = n_1n_2$ divise $x - y$. Pour le cas général avec $r + 1$ congruences, on suppose que le théorème est vrai jusqu'à r congruences. On considère les deux entiers n_1 et $\frac{n}{n_1}$. Ces deux entiers sont premiers entre eux puisque les n_i sont deux à deux premiers entre eux et l'on peut donc appliquer la méthode précédente et l'hypothèse de récurrence. ■

Test 1.14.

Soient $n_1 = 7$ et $n_2 = 9$. Trouver l'unique solution x modulo $n = 63$ telle que $x \equiv 5 \pmod 7$ et $x \equiv 6 \pmod 9$.

Remarque. Le théorème des restes chinois s'interprète aussi comme un isomorphisme entre $\mathbb{Z}/n\mathbb{Z}$ et $\mathbb{Z}/n_1\mathbb{Z} \times \ldots \times \mathbb{Z}/n_r\mathbb{Z}$.

Dans les protocoles cryptographiques, nous serons surtout intéressés par l'ensemble des éléments de $\mathbb{Z}/N\mathbb{Z}$ qui admettent un inverse.

Définition 1.14. *Soit N un entier positif. L'ensemble des éléments de $\mathbb{Z}/N\mathbb{Z}$ qui admettent un inverse forme un groupe, noté $(\mathbb{Z}/N\mathbb{Z})^*$ et appelé groupe multiplicatif de $\mathbb{Z}/N\mathbb{Z}$.*

Définition 1.15. (Indicatrice d'Euler) *Soit N un entier. On appelle indicatrice d'Euler $\phi(N)$ le nombre d'entiers positifs, compris entre 1 et N, qui sont premiers avec N.*

Pour le cas où $N = p$ est un nombre premier, tout entier $m \in \{1, \ldots, p - 1\}$ est premier avec p, donc $\phi(p) = p - 1$ pour p premier.
Pour $N = p^r$ (une puissance d'un premier p), les seuls entiers non premiers avec p^r sont les multiples de p : $p, 2p, \ldots, p^2, \ldots, p^{r-1}(p - 1), p^r$; il en découle que $\phi(p^r) = (p - 1)p^{r-1}$.
Pour le cas où $N = pq$ (avec p et q deux nombres premiers distincts), les nombres qui sont non premiers avec N sont tous les multiples de p et de q compris entre 1 et pq. Comme p et q sont premiers entre eux, le seul multiple commun est pq. Ainsi, on a

$$\phi(pq) = pq - \mathrm{Card}(\{\text{entiers non premiers avec } pq\}) = pq - p - q + 1$$

(le $+1$ est pour pq qui serait compté deux fois sinon), soit $\phi(pq) = (p - 1)(q - 1)$.
On en déduit alors la proposition qui suit.

Proposition 1.16. *Soit* N *un entier positif avec décomposition* $N = \Pi_{i=1}^{r} p_i^{r_i}$ *pour* p_i *des nombres premiers distincts, alors*

$$\phi(N) = \Pi_{i=1}^{r}(p_i - 1)p_i^{r_i - 1}.$$

Exemple 1.17. $\phi(72) = \phi(8)\phi(9) = 4 \times 6 = 24$ et $\phi(15) = \phi(3)\phi(5) = 2 \times 4 = 8$.

Test 1.15. Calculer $\phi(112)$ et $\phi(84)$.

On s'intéresse maintenant, pour un élément a de $\mathbb{Z}/N\mathbb{Z}$, à $a^{\phi(N)}$.
Nous rappelons, pour ce faire, les trois théorèmes fondamentaux suivants.

Théorème 1.18. (Petit théorème de Fermat) *Soient* p *un nombre premier et* a *un entier, alors*

$$a^{p-1} \equiv 1 \pmod{p},$$

si et seulement si p *ne divise pas* a.

Théorème 1.19. (Théorème d'Euler-Fermat) *Soit* N *un entier. Tout* $a \in (\mathbb{Z}/N\mathbb{Z})^*$ *vérifie*

$$a^{\phi(N)} \equiv 1 \pmod{N}.$$

Le petit théorème de Fermat est un cas particulier du théorème d'Euler-Fermat, qui est lui même un cas particulier du théorème de Lagrange appliqué au groupe multiplicatif $((\mathbb{Z}/N\mathbb{Z})^*, \cdot)$.

Théorème 1.20. (Théorème de Lagrange) *Soit* G *un groupe multiplicatif (fini) de cardinal* $|G|$*, alors, pour tout* $a \in G$*, on a*

$$a^{|G|} = 1.$$

III.2. Générateurs de $(\mathbb{Z}/N\mathbb{Z})^*$

Soit a un élément de $(\mathbb{Z}/N\mathbb{Z})^*$. On note $\mathtt{Gen}(a) \overset{\text{déf}}{=} \{a^m \pmod{N} \mid m \in \mathbb{Z}\}$ le sous-groupe multiplicatif de $(\mathbb{Z}/N\mathbb{Z})^*$ engendré par a. D'après le théorème de Lagrange, l'ordre de $\mathtt{Gen}(a)$ divise $\phi(N)$.

Définition 1.21. *On dit que* a *est un générateur de* $(\mathbb{Z}/N\mathbb{Z})^*$ *quand* $\mathtt{Gen}(a) = (\mathbb{Z}/N\mathbb{Z})^*$. *On parle parfois de générateur multiplicatif de* $(\mathbb{Z}/N\mathbb{Z})^*$.

Dans le cas où N est premier, nous avons rappelé dans le chapitre 7 qu'il existait toujours un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$. Soit g un tel générateur, alors

$$(\mathbb{Z}/p\mathbb{Z})^* = \{g^0 = 1, g^1, g^2, \ldots, g^{p-2}\}.$$

Exemple 1.22. Prenons $p = 7$, alors un générateur est 3 :

$$(\mathbb{Z}/7\mathbb{Z})^* = \{1, 2, 3, 4, 5, 6 \pmod{7}\} = \{3^0 = 1, 3^1 = 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5 \pmod{7}\}.$$

Test 1.16.

Soit p un nombre premier. Montrer que le nombre de générateurs de $(\mathbb{Z}/p\mathbb{Z})^*$ est exactement $\phi(p-1)$.

Remarque. Dans le cas où N n'est pas premier, il n'existe pas nécessairement de générateur multiplicatif de $(\mathbb{Z}/N\mathbb{Z})^*$. Par exemple, si l'on prend $N = 15$, alors on a $\phi(15) = 8$ et $(\mathbb{Z}/15\mathbb{Z})^* = \{1, 2, 4, 7, 8, 11, 13, 14 \pmod{15}\}$. On observe que $\mathrm{Gen}(2) = \{2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8 \pmod{15}\}$, $\mathrm{Gen}(13) = \{13^0 = 1, 13^1 = 13, 13^2 = 4, 13^3 = 7 \pmod{15}\}$, $\mathrm{Gen}(11) = \{11^0 = 1, 11^1 = 11 \pmod{15}\}$ et $\mathrm{Gen}(14) = \{14^0 = 1, 14 \pmod{15}\}$. On constate que tous ces groupes ont un ordre de $1, 2$ ou 4 et, comme leur réunion contient tous les éléments de $(\mathbb{Z}/15\mathbb{Z})^*$, il n'existe aucun générateur de $(\mathbb{Z}/15\mathbb{Z})^*$. Si maintenant on considère $(\mathbb{Z}/6\mathbb{Z})^* = \{1, 5 \pmod{6}\}$, alors 5 est un générateur.

Test 1.17.

1) Trouver un générateur de $(\mathbb{Z}/11\mathbb{Z})^*$.

2) Existe-t-il un générateur de $(\mathbb{Z}/18\mathbb{Z})^*$?

IV. Chiffrement à clé publique : R.S.A.

Comme nous l'avons rappelé dans l'introduction, Whitfield Diffie et Martin Hellman ont donné dans leur article de 1976, « New directions in cryptography », les bases de la cryptographie à clé publique. Dans cet article, ils ne proposaient pas d'algorithme de chiffrement à clé publique, mais uniquement un algorithme d'échange de clé que nous verrons dans la section suivante. Trouver un système de chiffrement était laissé comme un problème ouvert. Un tel système a été proposé l'année suivante par Ron Rivest, Adi Shamir et Leonard Adleman (mais publié en 1978) et a été popularisé sous le nom de R.S.A.

IV.1. Algorithme R.S.A.

On introduit une notion de couple de clés asymétriques, formé d'une clé publique K_P accessible à tous et d'une clé privée (secrète) K_S connue d'un seul individu. Il doit être facile de chiffrer un message en utilisant K_P et de déchiffrer le message si l'on a les deux clés ; mais il doit être difficile de déchiffrer le message si l'on ne connaît pas K_S.
La clé publique K_P d'Alice peut être publiée sur un annuaire. La cryptographie à clé publique fonctionnerait alors comme une boîte aux lettres dans laquelle tout le monde pourrait déposer des messages, mais dont seule Alice aurait la clé pour accéder aux messages qui seraient déposés dedans.

Test 1.18.

Calculer le nombre de couples de clés symétriques potentiellement nécessaires pour une grosse entreprise avec 100 000 collaborateurs qui voudraient pouvoir tous communiquer entre eux de manière confidentielle.

Méthode **Chiffrement R.S.A.**

1) *Génération des clés.* Soient p et q deux nombres premiers et $N = pq$. Alice choisit $e \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $1 \leqslant e \leqslant N-1$ et $\text{pgcd}(e,(q-1)(p-1)) = 1$. Alice calcule l'inverse d de e modulo $\phi(N)$:
$$ed \equiv 1 \pmod{(p-1)(q-1)}.$$
Clé publique : le couple (N,e). Clé privée : le triplet (p,q,d).

2) *Chiffrement.* Bob convertit son message M en une séquence d'éléments m de $(\mathbb{Z}/N\mathbb{Z})^*$ qui sont chiffrés séparément. Bob récupère la clé publique d'Alice $K_P = (N,e)$; il calcule le chiffré $c \equiv m^e \pmod N$ et l'envoie à Alice.

3) *Déchiffrement.* Alice reçoit c et calcule $m' \equiv c^d \pmod N$ avec sa clé privée d, alors $m = m'$.

Preuve. Vérifions que le déchiffrement fonctionne. On a $ed \equiv 1 \pmod{\phi(N)}$, donc il existe un entier k tel que $ed = 1 + k\Phi(N)$, alors
$$c^d \equiv m^{ed} \equiv m^{1+k\phi(N)} \equiv m.(m^{\phi(N)})^k \pmod N.$$
Or, d'après le théorème d'Euler-Fermat, $m^{\phi(N)} \equiv 1 \pmod N$, donc $c^d \equiv m \pmod N$ et Alice retrouve bien le message envoyé par Bob. ■

Exemple 1.23. Soient $p = 7, q = 11$ et $e = 13$. On calcule $N = 77$, $\phi(N) = 6.10 = 60$, et $d = 37$ par l'algorithme d'Euclide étendu puisque $13.37 \equiv 1 \pmod{60}$. La clé publique est $(77,13)$ et la clé privée est $(7,11,37)$. Soit $m = 9$ un message. Pour le chiffrement, on calcule $c = 9^{13} \pmod{77} = 58$; pour le déchiffrement, on a bien $c^{37} = 58^{37} \equiv 9 \pmod{77}$.

Test 1.19. Pour $p = 5, q = 11$ et $e = 7$, trouver d, calculer le chiffré de 28 et vérifier le déchiffrement.

Avant de discuter la sécurité de R.S.A., commençons par détailler la mise en œuvre des calculs.

IV.2. Mise en œuvre des calculs

Si l'on s'y prend mal, les calculs sont infaisables. Dans l'exemple ci-dessus, on a
$$58^{37} = 1765370080255312576729338996004920537428940014432503311704239636 48.$$
Il est déraisonnable de calculer ce nombre. Une première observation est que *tous les calculs* doivent être effectués modulo N. Dans l'exemple, nous travaillons modulo 77, donc nous n'allons manipuler que des nombres à deux chiffres.
Pour calculer $58^{37} \pmod{77}$, il faudrait naïvement effectuer 37 multiplications modulo 77. On fait beaucoup mieux en utilisant l'exponentiation binaire [3], vue dans le tome *Mathématiques L2*.

3. On trouve aussi ce procédé sous une multitude d'autres noms dans la littérature : *exponentiation modulaire, exponentiation rapide, square and multiply, repeated squaring*, etc.

Rappel **Exponentiation binaire**

Pour un élément g d'un groupe multiplicatif, on veut calculer g^n.
Décomposons n en base 2 : $n = n_0 + 2n_1 + 2^2 n_2 + \cdots + 2^k n_k$ où les $n_i \in \{0, 1\}$.
Ainsi, g^n peut se récrire

$$\begin{aligned} g^n &= g^{n_0} g^{2n_1} g^{2^2 n_2} \cdots g^{2^k n_k} \\ &= g^{n_0} (g^2)^{n_1} (g^{2^2})^{n_2} \cdots (g^{2^k})^{n_k} . \end{aligned}$$

L'élément g^n peut donc être obtenu en multipliant les g^{2^i} tels que $n_i = 1$.
On calcule les g^{2^i} récursivement par $g^{2^i} = (g^{2^{i-1}})^2$.
Le nombre total de multiplications est donc inférieur à deux fois le nombre de chiffres de n en base 2 (donnés par le logarithme de n en base 2).

Appliquons ce procédé dans notre exemple. On a $37 = 32+4+1 = 2^5+2^2+1$. Posons $c_0 = 58$, puis $c_1 = c_0^2 \pmod{77} \equiv 53 \pmod{77}$, ensuite $c_2 = c_1^2 \pmod{77} \equiv 37 \pmod{77}$, $c_3 = c_2^2 \pmod{77} \equiv 60 \pmod{77}$, $c_4 = c_3^2 \pmod{77} \equiv 58 \pmod{77}$, et enfin $c_5 = c_4^2 \pmod{77} \equiv 53 \pmod{77}$, alors $m' \equiv c_0.c_2.c_5 \pmod{77} \equiv 9 \pmod{77}$.

Plus généralement, si N est un nombre à n chiffres (en base 2), nous voyons que le coût total est au pire de $2n$ multiplications de nombres de n chiffres (n pour calculer les puissances successives, n pour reconstruire m'). Si l'on compte qu'une multiplication de deux nombres de n chiffres utilise n^2 opérations, nous voyons que cette exponentiation demandera donc environ $2n^3$ opérations. Le coût de la multiplication de deux nombres de n chiffres peut être amélioré, par exemple en utilisant le procédé de Karatsuba (voir le chapitre 5 du tome *Mathématiques L2*). D'autres améliorations (basées sur les méthodes modulaires et le théorème des restes chinois) sont proposées en exercice.

IV.2.1. Coût pratique du chiffrement

Pour imaginer ce que cela donne sur de grands nombres, supposons que N soit un nombre de 1024 bits. Il faudrait donc de l'ordre de 10^9 opérations arithmétiques pour le chiffrement. Pour une machine cadencée à 1 gigahertz et une architecture de 64 bits, l'opération prendrait donc environ 15 millisecondes. Cet exemple a une valeur indicative grossière, mais donne une idée des ordres de grandeur. Il peut arriver que l'on dispose de processeurs optimisés pour ces opérations (des *cryptoprocesseurs*), auquel cas les opérations de chiffrement et déchiffrement sont significativement plus rapides.
En pratique, l'opération qui coûte le plus cher est le déchiffrement : on ne peut éviter le fait que d ait n chiffres en général puisqu'il est obtenu comme inverse de e modulo $\phi(N)$. En revanche, rien n'interdit de prendre un e pas trop grand. Il est fréquemment choisi de prendre $e = 2^{16} + 1$ (c'est par exemple le cas sur la bibliothèque opensource OpenSSL) ; comme nous avons vu que le coût de l'exponentiation binaire est lié au nombre de bits de l'exposant, cela permet de rendre le chiffrement quasi quadratique.

On peut montrer que le chiffrement à clé publique comme R.S.A. est 100 à 1000 fois plus lent en général que des procédés à clé secrète. En pratique, pour chiffrer de gros messages, on pourra utiliser R.S.A. pour partager une clé secrète : on pourra alors faire de nouveau du chiffrement symétrique en toute sécurité.

IV.3. La sécurité de R.S.A.

IV.3.1. R.S.A. et factorisation

Si l'on sait factoriser N en un temps raisonnable,alors on sait calculer $\phi(N) = (p-1).(q-1)$. On retrouve alors d en inversant e modulo $\phi(N)$ par l'algorithme d'Euclide étendu. On peut alors déchiffrer tout message chiffré avec (N, e). On a donc *cassé* le système.
En fait, si l'on sait calculer $\phi(N)$, alors on retrouve p et q facilement : comme $\phi(N) = (p-1).(q-1)$ et que l'on connaît $N = pq$, la connaissance de $\phi(N)$ donne $p + q$; on trouve alors p et q en résolvant une équation du second degré.
Casser le système R.S.A. revient à être capable, pour tout chiffré $c \equiv m^e \pmod N$, de retrouver m. Il s'agit donc de savoir calculer en temps raisonnable une *racine e-ième* modulo N ; il n'y a actuellement pas de bon algorithme pour ce faire.
La meilleure attaque connue contre l'algorithme R.S.A. reste la factorisation de N. Remarquons que si l'on savait casser R.S.A., il n'est pas clair que l'on saurait pour autant factoriser N ou calculer $\phi(N)$.

IV.3.2. Taille des clés et choix des paramètres

La meilleure attaque connue étant la factorisation de N, on se base sur la complexité du meilleur algorithme de factorisation connu pour choisir la taille des clés. La meilleure attaque pour résoudre le problème de la factorisation est appellée *crible algébrique* (nous ne l'expliquerons pas ici) et a une complexité en $L_{\frac{1}{3}}(N) \overset{déf}{=} e^{\ln(N)^{\frac{1}{3}} \ln(\ln(N))^{\frac{2}{3}}}$. On choisit donc la taille des paramètres de telle sorte que la complexité de l'attaque corresponde au degré de sécurité souhaité (2^{80} opérations jusqu'en 2010). Cela conduit à une taille de clé de 1024 bits : l'entier N devrait donc avoir de l'ordre de 1024 chiffres (en représentation binaire), donc environ 308 chiffres (en représentation décimale). À partir de 2010, le N.I.S.T. recommande une taille de 2048 bits. Une taille de clés de 1024 bits impose de prendre des nombres p et q d'environ 512 bits. Il convient, de plus, de les prendre suffisamment différents (par exemple, que le reste de la division euclidienne de p par q soit grand), et tels que $p \pm 1$ et $q \pm 1$ n'aient pas de petits facteurs : en effet, il existe des algorithmes spécifiques de factorisation permettant de détecter ces cas et, alors, de retrouver p et q.
Nous proposons en exercice des attaques de R.S.A. pour le cas où l'on aurait ainsi choisi de mauvais paramètres.

Test 1.20.

On suppose que A et B utilisent le même module R.S.A. N avec deux clés publiques e_A et e_B premières entre elles. On suppose que C envoie le même message chiffré m^{e_A} et m^{e_B} à A et B. Montrer que E, qui écoute les communications, peut retrouver facilement alors le message m.
Test 1.21.

Le R.S.A. a une propriété multiplicative.
1) Montrer que pour un chiffrement R.S.A., avec c_1 =Chiffré(m_1) et c_2 =Chiffré(m_2), on a $c_1 c_2$ =Chiffré($m_1 m_2$)
2) On suppose que l'on a un message chiffré c_1 et que l'on peut faire déchiffrer des messages (mais pas c_1). Montrer qu'alors, en faisant déchiffrer un message lié à c_1, mais différent comme yc_1 (avec un y à définir), on peut retrouver le clair associé à c_1.
Test 1.22.

On a vu que par le chiffrement R.S.A., un message était toujours chiffré de la même façon et que le chiffrement était donc déterministe. Proposer une légère variation sur la façon de faire chiffrer un message pour rendre le chiffrement non déterministe. On pourra penser à n'utiliser pour le message à chiffrer qu'une partie des bits disponibles pour le chiffrement.
Test 1.23.

Afin d'améliorer la sécurité des messages, Bob choisit deux exposants e_1 et e_2 ; il demande à Alice de chiffrer d'abord son message par e_1 pour obtenir $c_1 = m^{e_1}$, puis de rechiffrer par e_2 pour obtenir $c_2 = c_1^{e_2}$. Alice envoie alors c_2. Est-ce que ce double chiffrement améliore la sécurité ?
Test 1.24.

On suppose que l'on chiffre un message m en

calculant $m^3 \pmod{101}$. Comment peut-on déchiffrer (c'est-à-dire trouver d tel $c^d = m \pmod{101}$) ? On remarquera que 101 est premier.

Test 1.25.

Les exposants $e = 1$ et $e = 2$ ne doivent pas être utilisés comme exposant public pour R.S.A. Pourquoi ?

IV.4. Génération de nombres premiers

Nous avons vu que la création de la clé impliquait le choix de nombres premiers p et q avec certaines propriétés. Ces dernières peuvent être réalisées en cherchant des nombres sous une forme sympathique, puis en testant s'ils sont premiers. La question revient alors à tester si un nombre est premier ou non.

Le théorème des nombres premiers montre que la densité des nombres premiers est de l'ordre de $\frac{1}{\ln(N)}$ autour du nombre N. Pour un entier N de l'ordre de 512 bits, cela impliquerait une probabilité raisonnable de trouver un nombre premier en faisant plusieurs dizaines d'essais *au hasard*.

IV.4.1. Test de Fermat

Une première façon de faire consiste à utiliser une réciproque du petit théorème de Fermat, appelé *test de Fermat*. Soient n un entier et a un entier tels que $1 < a < n-1$, alors, si n est premier, le théorème de Fermat montre que $a^{n-1} \equiv 1 \pmod{n}$. Si l'on prend plusieurs a au hasard et que l'on en trouve un pour lequel $a^{n-1} \not\equiv 1 \pmod{n}$, alors n n'est pas premier.
On pourrait espérer que si (pour des a aléatoires) on avait toujours $a^{n-1} = 1 \pmod{n}$, alors il y aurait de bonnes chances que n soit premier. Cette idée se heurte au fait qu'il existe des nombres (dits de Carmichael) tels que, pour ces nombres n, on a $a^{n-1} = 1 \pmod{n}$ pour tout $1 < a < n-1$. Cela signifie que, par le test de Fermat, ces nombres apparaîtraient comme potentiellement premiers, alors qu'ils ne le sont pas du tout. Le premier nombre de Carmichael est $561 = 3.11.17$. La densité de ces nombres est relativement faible, mais constitue une limite à cette méthode.

IV.4.2. Test de Rabin-Miller

En 1976 puis 1980, M. Rabin et V. Miller ont proposé une méthode qui permet d'éviter ce genre de problème. Observons que si n est premier, alors $\mathbb{Z}/n\mathbb{Z}$ est intègre et il existe exactement deux nombres dans $\mathbb{Z}/n\mathbb{Z}$ dont le carré vaut 1 (c'est-à-dire 1 et $n-1$). Par contraposée, s'il existe un nombre r tel que $1 < r < n-1$ tel que $r^2 \equiv 1 \pmod{n}$, alors n est un nombre composé. Le principe du test est de choisir des nombres a au hasard et de calculer leurs puissances modulo n par l'exponentiation binaire (rapide) pour tenter de faire apparaître des racines carrées *non triviales* de 1.
Notons $n-1 = 2^s m$ avec m impair et supposons que n est premier. Prenons un entier a au hasard. Par Fermat, on a $a^{n-1} \equiv 1 \pmod{n}$, or $a^{n-1} = (a^m)^{2^s}$. Donc le nombre $a^{n-1} = (a^m)^{2^{s-1}}$ doit valoir 1 ou -1 (car son carré vaut 1), sinon n ne serait pas premier. Si l'on trouve un nombre de la forme $a^{n-1} = (a^m)^{2^i}$ dont le carré vaut 1, mais qui ne vaut ni 1 ni -1, alors n n'est donc pas premier. Cela fournit le test suivant.

Méthode **Test de Rabin-Miller**

On se donne un entier n impair et un entier t (le degré de sécurité).
On calcule s et m tels que $n-1 = 2^s m$ avec m impair. Puis on répète t fois la séquence suivante :

1) choisir un entier a aléatoire avec $1 < a < n-1$;
2) si $a^m \not\equiv 1 \pmod{n}$ et $a^{2^r m} \not\equiv -1 \pmod{n}$ pour tout r tel que $1 \leqslant r \leqslant s-1$, alors retourner *n est composé.*

Si, au bout de t séquences, n n'est pas retourné comme composé, alors la probabilité que n ne soit pas premier est inférieure à 2^{-2t}. En effet, Rabin et Miller ont montré que, étant donné un a aléatoire, la probabilité qu'un n passant le test soit non premier est inférieure à $1/4$. Donc la probabilité de passer le test et d'être non premier est inférieure à $(1/4)^t$.

En utilisant l'exponentiation modulaire, ce test utilisera environ $\mathcal{O}(t\log(n)^3)$ opérations. En pratique, on commence souvent par effectuer le test de Fermat avec des petites valeurs de a pour débuter la recherche car les calculs sont moins coûteux ; puis on continue avec le test de Miller-Rabin en choisissant le degré t de sécurité souhaité.

V. Échange de clés et chiffrement basés sur le logarithme discret

Le problème du logarithme discret a d'abord été utilisé pour l'échange de clés dans l'article de Diffie-Hellman de 1976. En 1984, T. El Gamal a proposé des systèmes pour l'utiliser pour le chiffrement et la signature.
Soient p un nombre premier et g un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$. Soit x dans $(\mathbb{Z}/p\mathbb{Z})^*$ tel que $y \equiv g^x \pmod{p}$. Le *problème du logarithme discret* consiste à retrouver x à partir de y.

V.1. Échange de clés de Diffie-Hellman

Pour appliquer un algorithme de chiffrement symétrique, il est nécessaire que les deux individus partagent un secret commun. Le problème se pose alors pour les deux parties d'avoir une clé en commun. Cela demande a priori d'avoir eu un contact préalable (direct ou indirect) sous une forme ou sous une autre. L'algorithme d'échange de clés de Diffie-Hellman permet aux deux parties de partager une clé secrète sans avoir été en contact préalablement.

Méthode **Échange de clés de Diffie-Hellman**

1) *Entente préalable.* Alice et Bob conviennent publiquement d'un entier p et d'un générateur g de $(\mathbb{Z}/p\mathbb{Z})^*$.

2) *Protocole.*
   - A et B choisissent séparément des entiers aléatoires, a et b de $(\mathbb{Z}/p\mathbb{Z})^*$;
   - A envoie alors $g^a \pmod{p}$ à B et B envoie $g^b \pmod{p}$ à A ;
   - A calcule $(g^b)^a \pmod{p} = g^{ba} \pmod{p}$ et B calcule $(g^a)^b \pmod{p} = g^{ab} \pmod{p}$.

Le secret partagé est alors $g^{ab} \pmod{p}$.

Preuve. L'algorithme fonctionne car la mise à la puissance est commutative et $g^{ab} = g^{ba} \pmod p$. ∎

Si l'on était capable de résoudre le problème du logarithme discret par un algorithme efficace, alors on pourrait retrouver a et b à partir de $g^a \pmod p$ et $g^b \pmod p$: on pourrait donc retrouver le secret partagé $g^{ab} \pmod p$.
On peut introduire le problème calculatoire de Diffie-Hellman : étant donné $g^a \pmod p$ et $g^b \pmod p$, retrouver $g^{ab} \pmod p$. Ce problème peut sembler plus facile que le logarithme discret ; pourtant, on ne sait actuellement pas faire mieux que de le résoudre par le logarithme discret.

Test 1.26.

On considère $p = 17$ et un générateur $g = 3$. Vérifier que l'algorithme fonctionne sur un exemple avec $a = 7$ et $b = 13$.

Il existe une attaque très simple qui permet d'écouter de manière passive les communications entre A et B : l'*attaque par le milieu*.
On suppose qu'il existe un attaquant C qui peut intercepter les communications. Lorsque A commence un protocole d'échange de clés avec B, C se fait passer pour B et échange un secret avec A par le protocole d'échange de clés. De même, C échange un secret avec B en se faisant passer pour A. Lorsque A envoie par la suite des messages chiffrés en clé sysmétrique à partir de la clé échangée à B, C intercepte les messages qu'il peut déchiffrer puisqu'il a échangé une clé avec A en se faisant passer pour B. L'attaquant lit alors le message, puis le chiffre en direction de B avec la clé échangée avec B. Il procède de même avec les messages venant de B. De cette manière, A et B ont l'impression de communiquer entre eux de manière chiffrée, mais C intercepte toutes les communications entre eux sans qu'ils ne s'en rendent compte.

Cette attaque montre qu'il vaut mieux ne pas utiliser l'algorithme directement sous cette forme. Il existe une variation du protocole qui utilise des éléments d'identification et permet d'éviter cette attaque.

Remarque. Pour faire fonctionner ce protocole, il suffit en fait d'avoir un groupe (pas nécessairement $(\mathbb{Z}/p\mathbb{Z})^*$) dans lequel le problème du logarithme discret est difficile. Cela permet de nombreuses variantes de Diffie-Hellman.

V.2. Chiffrement d'El Gamal

Nous présentons maintenant la méthode d'El Gamal, qui utilise la difficulté du logarithme discret pour faire du chiffrement.

Méthode **Chiffrement d'El Gamal**

1) *Génération des clés.* Bob choisit un nombre premier p et un générateur g de $(\mathbb{Z}/p\mathbb{Z})^*$. Il prend un nombre aléatoire x_B, avec $1 \leqslant x_B \leqslant p-1$. Il calcule $X_B = g^{x_B} \pmod p$. Clé publique : X_B. Clé privée : le triplet (p, g, X_B).

2) *Chiffrement.* Alice récupère la clé publique de Bob (p, g, X_B). Elle représente le message à chiffrer m sous forme d'un entier dans $(0, \dots, p-1)$ et choisit un entier aléatoire $1 \leqslant k \leqslant p-1$. Elle calcule $\gamma = g^k \pmod p$ et $\delta = mX_B^k \pmod p$. Le chiffré c est le couple $c = (\gamma, \delta)$.

3) *Déchiffrement.*
Bob reçoit c. Il calcule $m' = \gamma^{-x_B}\delta \pmod p$ et alors $m = m'$.

Preuve. On vérifie que $\gamma^{-x_B}\delta = g^{-x_B k} g^{x_B k} m = m \pmod p$. ∎

La complexité de l'attaque pour le problème du logarithme discret est similaire à la complexité de la factorisation des entiers en $L_{1/3}(p)$. On prend donc p de taille 1024 bits (voir la section correspondante dans la partie R.S.A.).

La comparaison de ce chiffrement avec R.S.A. est intéressante. Le chiffré est constitué de deux nombres modulo p, là où R.S.A. n'en utilise qu'un. En termes de coût de calcul, il faut effectuer deux exponentiations modulaires pour le chiffrement. Rappelons que, pour R.S.A., on pouvait choisir un exposant public petit et donc diminuer le coût du chiffrement ; ici, une telle astuce n'est pas disponible.
Pour le déchiffrement, une seule exponentiation modulaire est nécessaire (comme pour R.S.A.). Ainsi, l'algorithme pourrait sembler moins performant que R.S.A. à première vue. Il a néammoins l'avantage d'être non déterministe (en ce sens que le chiffré depend d'un entier k aléatoire) : deux messages identiques seront chiffrés de deux manières différentes car ils utiliseront des aléas k différents.
En fait, ce schéma peut être vu comme un procédé de masquage similaire au chiffrement à flot de Vernam (on utilise un aléa k pour chiffrer), mais sous forme multiplicative.

VI. Fonctions de hachage

VI.1. Définitions

Définition 1.24. *Une fonction de hachage est une fonction h possédant les deux propriétés suivantes :*

1) propriété de compression : pour tout x, le nombre $h(x)$ est de taille fixée ;

2) facilité de calcul : $h(x)$ se calcule facilement.

Exemple 1.25. Soit $x = (x_0, x_1, \dots, x_m)$ pour m quelconque. La fonction qui à x associe (X_0, X_1), avec $X_0 = \sum_{i\,pair} x_i$ et $X_i = \sum_{i\,impair} x_i$, est une fonction de hachage.

Une fonction de hachage sert à produire une *empreinte* courte d'un message indépendamment de sa longueur. Elle permettra par exemple de contrôler l'intégrité des données. On peut aussi imaginer, pour signer un message, de signer un haché bien choisi (pour avoir une signature

courte). Pour réaliser ce genre d'objectif, nous devons imposer des contraintes à notre fonction de hachage.

Définition 1.26. *Soit* $h : X \to Y$ *une fonction de hachage.*

1) h *est dite à sens unique si, pour presque tout* y *de* Y*, il est calculatoirement infaisable de trouver* x *tel que* $y = h(x)$.

2) h *est dite faiblement sans collision si, pour* x *donné, il est calculatoirement infaisable de trouver* x' *tel que* $h(x') = h(x)$.

3) h *est dite sans collision s'il est calculatoirement infaisable de trouver* x *et* x' *tels que* $h(x) = h(x')$.

Nous vous proposons de découvrir les implications qui existent (ou non) entre ces trois notions dans les tests suivants.

Test 1.27.

Montrer que *sans collision* implique *faiblement sans collision.*

Test 1.28.

Soit g une fonction de hachage sans collision qui renvoie des hachés de taille n. On considère la fonction de hachage $h(x)$ suivante qui renvoie un haché de taille $n+1$. Si x a pour longueur n alors $h(x) = 1||x$ (la concaténation du bit 1 et de x), sinon $h(x) = 1||g(x)$.
Montrer que g n'est pas à sens unique, mais a la propriété d'être sans collision.

Test 1.29.

On peut montrer que, pour un module R.S.A. $n = pq$, retrouver x à partir de $x^2 \pmod n$ est une opération difficile, qui revient à factoriser n. Montrer, en utilisant ce résultat, que la propriété d'être à sens unique pour une fonction quelconque (de hachage ou pas) peut être vérifiée sans que la propriété d'être faiblement sans collision le soit.

Test 1.30.

Soient g_1 et g_2 deux fonctions de hachage telles qu'au moins une des deux soit sans collision. Soit h une fonction de hachage définie par $h(x) = g_1(x)||g_2(x)$ (la concaténation). Montrer qu'alors h est sans collision.

En pratique, le haché d'un message peut être vu comme une empreinte d'un message ou un condensé. La propriété d'être sans collision permet de se prémunir contre des attaques sur l'intégrité d'un message. Supposons, par exemple, qu'un message M soit déposé sur une page Internet et qu'on l'y laisse pendant plusieurs jours. Comment savoir si le message n'a pas été modifié ? Une solution simple consiste, avant de déposer le message, à calculer une empreinte $h(M)$ par une fonction de hachage h. Pour savoir si le message a été modifié, on calcule une empreinte (ou un haché) du message M' que l'on a retrouvé. Si $h(M') = h(M)$, alors nécessairement $M' = M$. En effet, pour un haché $h(M)$ donné, la propriété d'être faiblement sans collision assure que l'on ne peut (en pratique) trouver $M' \neq M$ tel que $h(M') = h(M)$. Garder un haché d'un message pour une comparaison postérieure avec le haché du même message éventuellement modifié permet donc de vérifier l'intégrité d'un message.

Dans le cas précédent, on utilise simplement la notion d'être faiblement sans collision. Nous verrons comment le fait d'être aussi sans collision peut être nécessaire comme, par exemple, pour la signature électronique. Dans ce cas, on signe (en pratique) un haché ; le fait de ne pas être capable de trouver une collision a priori garantit la non-répudiation d'un message, à savoir que le signataire a bien signé un message donné et pas un autre (avec le même haché).

VI.2. Attaque par paradoxe des anniversaires

Il existe une attaque classique dite par paradoxe des anniversaires qui permet de construire des collisions pour une fonction de hachage avec une bonne complexité. Avant de décrire cette attaque, nous rappelons le paradoxe des anniversaires.

VI.2.1. Le paradoxe des anniversaires.

Soit X un ensemble de n éléments. On effectue k tirages indépendants et avec remise dans X . On souhaite savoir, en moyenne, au bout de combien de tirages on a une bonne chance d'obtenir deux tirages identiques.

Soit P la probabilité qu'au moins deux tirages soient identiques si l'on effectue k tirages. On introduit la probabilité $\bar{P} = 1 - P$ que les k tirages soient distincts. On calcule

$$\bar{P} = \frac{n-1}{n}\frac{n-2}{n}\cdots\frac{n-(k-1)}{n} = \Pi_{i=1}^{k-1}(1-\frac{i}{n}),$$

soit

$$P = 1 - \Pi_{i=1}^{k-1}(1-\frac{i}{n}).$$

Exemple 1.27. Si l'on considère $n = 365$ (le nombre de jours de l'année), on peut se poser la question de la taille d'une classe pour avoir une bonne chance que deux personnes soient nées le même jour. De manière peut-être surprenante, on obtient que, pour une classe de $k = 23$ personnes, cette probabilité est de l'ordre de $0,5$. Pour une classe de $k = 40$ personnes, cette probabilité monte à $0,9$.

Test 1.31.

Calculer la probabilité P pour $n = 10$ et $k = 3$.

On cherche maintenant à évaluer comment cette probabilité évolue pour de grandes valeurs de n, typiquement $n = 2^m$ avec $m \geqslant 60$. Soit $\bar{P} = \Pi_{i=1}^{k-1}(1-\frac{i}{n})$. Les propriétés de la fonction exponentielle impliquent que, pour tout x réel, on a $e^x \geqslant 1 + x$, ce qui amène

$$\bar{P} \leqslant e^{\sum_{i=1}^{k-1} -\frac{i}{n}} = e^{-\frac{k(k-1)}{2n}}.$$

On a donc $P = 1 - \bar{P} \geqslant 1 - e^{-\frac{k(k-1)}{2n}}$. On en déduit que $P \geqslant 1/2$ pour $k \geqslant \frac{1+\sqrt{1+8\ln(2)n}}{2}$.

Cette valeur de k est équivalente (pour n grand) à $\sqrt{2\ln(2)}\sqrt{n} \simeq 1,18\sqrt{n}$. Pour $n = 365$, on retrouve bien $1,18\sqrt{n} \simeq 22.5$.

Nous allons maintenant utiliser le paradoxe des anniversaires pour donner une méthode générique permettant de trouver des collisions pour une fonction de hachage donnée.

Proposition 1.28. *Si une fonction de hachage a une sortie de taille m bits, alors il existe une probabilité supérieure à $1/2$ de trouver une collision en $\mathcal{O}(2^{\frac{m}{2}})$ opérations.*

Preuve. Une sortie de m bits donne 2^m possibilités. Le paradoxe des anniversaires indique alors qu'en moyenne, on trouve une collision avec une probabilité supérieure à $1/2$ pour $\sqrt{2\ln(2)}.2^{\frac{m}{2}}$ tirages aléatoires. Il suffit donc de considérer au moins un tel nombre de messages distincts à hacher, de les conserver dans une liste et de comparer toute nouvelle valeur de la fonction de hachage sur un nouveau message à celles déjà obtenues. On obtient ainsi une complexité en $\mathcal{O}(2^{\frac{m}{2}})$. ■

VI.2.2. Probabilités et attaque en moyenne.

Dans la proposition précédente, on n'obtient pas une probabilité de 1 de construire une collision, mais simplement de 1/2. L'attaque par paradoxe des anniversaires est probabiliste : quel que soit le nombre de tirages que l'on fait, il existe toujours une possibilité théorique de ne pas trouver de collision. En pratique, pour estimer la complexité d'une attaque, on estime une complexité en moyenne. Ainsi, si un événement nécessaire pour une attaque a une chance sur q de se produire, on considérera simplement qu'il faudra le répéter q fois pour que cela marche. On multipliera donc simplement la complexité par q. En pratique, donc, on considère que tant que la probabilité n'est pas trop petite, elle n'empêchera pas l'attaque : on incluera simplement cette probabilité dans le coût total de l'attaque.

La proposition précédente implique que, pour obtenir une sécurité de 2^{80} pour les attaques cherchant des collisions, il est nécessaire de prendre une sortie d'au moins 160 bits pour une fonction de hachage. C'est apparemment long comme nombre, mais c'est beaucoup plus court que la taille (1024 bits) des entiers préconisés dans R.S.A.

VI.3. Fonction de hachage itérée

VI.3.1. Schéma général

Pour construire une fonction de hachage, on utilise le schéma de Damgard qui part d'une fonction de compression que l'on itère. Une fonction de compression est une fonction f de $\mathbb{F}_2^n \times \mathbb{F}_2^r \to \mathbb{F}_2^r$ qui prend en entrée deux valeurs de longueur n bits et r bits et renvoie en sortie un vecteur de r bits.

On découpe le message x à hacher en t morceaux $x_1, \ldots, x_t$ de taille n. On part d'un bloc H_0 constant de longueur r, puis on itère à chaque étape par $H_i = f(x_i, H_{i-1})$ pour $1 \leqslant i \leqslant t+1$. Ainsi, à chaque étape, on intègre dans le haché intermédiaire H_i une partie du message x_i, tout en conservant une taille de haché constante. Éventuellement, on utilise une fonction finale g pour obtenir $h(x) = g(H_{t+1})$.

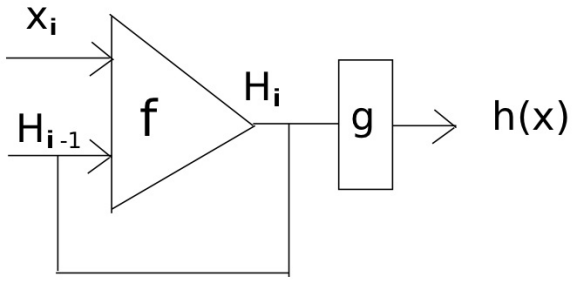

Fig. 1.5. Principe de hachage itéré

Un résultat de Merkle et Damgard assure que si l'on ajoute à la fin du message à hacher la longueur du message en bits, une collision sur la fonction de hachage implique une collision sur la fonction de compression. Ce résultat montre que si la fonction de compression est sans collision, alors la fonction de hachage est sans collision.

Il existe trois grands types de fonctions : les fonctions de compression basées sur des algorithmes de chiffrement existant que l'on estime sûrs, des fonctions de compressions spécialisées (c'est le cas des algorithmes les plus utilisés comme MD5, SHA ou RIPEMD), et enfin des fonctions dont la sécurité se ramène à un problème difficile (comme la factorisation ou le logarithme

discret). Cette dernière classe de fonctions a l'avantage d'avoir une sécurité que l'on peut ramener à un problème connu, mais on obtient des fonctions souvent trop lentes en pratique.
Dans la suite, on considère particulièrement les deux premiers types de famille.

VI.3.2. Fonctions de compression basées sur du chiffrement par blocs

On peut imaginer plusieurs types de schémas. Par exemple, une fonction où l'on utilise le bloc H_{i-1} comme clé pour chiffrer x_i,

$$H_i = E_{g(H_{i-1})}(x_i) \oplus x_i,$$

où g désigne une fonction qui transforme H_{i-1} en une longueur adaptée à l'algorithme de chiffrement et $E_k()$ un algorithme de chiffrement par blocs.

On peut aussi considérer un schéma où x_i est utilisé comme clé de chiffrement :

$$H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}.$$

Il existe aussi des variations où l'on utilise deux tours de chiffrement. Ce type de fonction de compression a l'avantage d'avoir une sécurité que l'on peut relier à la sécurité de la fonction de chiffrement. En revanche, les contraintes pour que f soit sans collision imposent une longueur de chiffré longue qui rend en pratique ce type de fonctions plus lentes que les fonctions de compressions spécialisées.

VI.4. Fonctions de chiffrement spécialisées

Pour la fonction de compression, on choisit des fonctions rapides non linéaires qui ont pour but d'amener de la confusion à chaque utilisation de la fonction de compression f.

VI.4.1. Une fonction de hachage itérée

On montre dans l'exemple pédagogique suivant (non réaliste) comment une fonction de hachage itérée opère.

Exemple 1.29.

On considère comme fonction de compression une fonction $f : \mathbb{F}_2^8 \times \mathbb{F}_2^4 \to \mathbb{F}_2^4$ qui rend un haché de longueur 4 et fonctionne de la manière suivante. On considère la fonction Rot_k qui opère une rotation circulaire de k positions vers la droite. On écrit le bloc x_i de longueur 8 comme une séquence de deux blocs A et B de longueur 4 et, à partir de H_{i-1}, on écrit $H_i = f(x_i, H_{i-1}) = H_{i-1} \oplus g(A, B, H_{i-1})$ avec

$$g(A, B, H_{i-1}) = (\mathrm{Rot}_1(A) \text{ ET } H_{i-1}) \text{ OU } (B \oplus \mathrm{Rot}_2(H_{i-1})),$$

où le ET et le OU logique sont appliqués bit à bit et $\oplus$ désigne la somme modulo 2 (XOR). On initialise par $H_0 = 1011$.
Calculons le haché, par la fonction de hachage itérée, de $M = 10001010001101$.

Préparons le message à hacher. Le message a pour longueur 14. On ajoute deux 0 au message pour obtenir une longueur multiple de 8, puis un nouveau bloc de taille 8 contenant comme information la longueur (ici 14). On obtient donc une séquence de 3 blocs de longueur 8 :

$$x_1, x_2, x_3 = 10001010, 00110100, 00001110.$$

Pour le premier passage, on part de $H_0 = 1011$ et $x_1 = (A, B) = (1000, 1010)$. On applique alors la fonction de compression f en calculant

$$\begin{aligned} g(A, B, H_0) &= \mathrm{Rot}_1(A) \text{ ET } H_0) \text{ OU } (B \oplus \mathrm{Rot}_2(H_0) \\ &= (0100 \text{ ET } 1011) \text{ OU } (1010 \oplus 1110) \\ &= 0000 \text{ OU } 0100 = 0100, \end{aligned}$$

soit $H_1 = f(x_1, H_0) = H_0 \oplus g(A, B, H_0) = 1011 \oplus 0100 = 1111$. On réitère alors avec $x_2 = (0011, 0100)$ et $H_1 = 1111$ et l'on trouve $g(A, B, H_1) = (1011)$ et $H_2 = 1111 \oplus 1011 = 0100$. Le dernier passage avec $x_3 = (0000, 1110)$ et $H_2 = 0100$ donne $g(A, B, H_2) = 1111$ et $H_3 = 0100 \oplus 1111 = 1011$. Le haché est donc $h(x) = 1011$.

VI.4.2. Le cas de SHA

Ron Rivest a introduit pour l'algorithme MD4 (Message Digest - 4) une structure de fonctions spécialisées qui a ensuite été reprise pour MD-5 et le standard SHA-1 (pour Secure Hash Algorithm). L'algorithme SHA utilise comme fonction de compression une fonction $\mathbb{F}_2^{512} \times \mathbb{F}_2^{160} \to \mathbb{F}_2^{160}$. Les blocs de taille 512 sont séparés en 16 blocs de taille 32 bits (pour utiliser la structure 32 bits des ordinateurs) et le haché intermédiaire a une taille de 160, soit 5 blocs de taille 32. La fonction de compression mélange, entre autres, l'utilisation d'opérateurs de rotation, de OU, de ET et de XOR.

Test 1.32.

On suppose donnée une valeur initiale VI et une bonne fonction de chiffrement $E_k(m)$ qui chiffre un message m avec une clé k de taille n. On considère la fonction de hachage h définie pour un message $m = (m_1, \cdots, m_l)$ (on suppose que tous les blocs m_i ont une même longueur n) par $h(m) = E_{m_1}(VI) \oplus E_{m_2}(VI) \oplus \cdots \oplus E_{m_l}(VI)$.
1) Donner une attaque très simple qui relie le fait de trouver un antécédent pour h, à la sécurité de E.
2) Rappeler la notion de fonction de hachage sans collision. Est-ce que h est sans collision ?

VII. Signature

VII.1. Définition

On souhaite obtenir de manière électronique des propriétés un peu plus fortes que celles que l'on espère d'une signature manuscrite classique :

1) Personne ne peut imiter la signature.
2) Tout le monde peut vérifier la signature.
3) La signature n'est pas réutilisable.
4) Si le message signé est altéré, la signature n'est plus valide.
5) Non-répudiation : un signataire ne peut nier avoir signé un message.

On peut définir une signature digitale comme suit.

Définition 1.30. (Signature digitale à clé publique)
- Une signature digitale est une suite d'informations associée à un message et satisfaisant en particulier aux cinq propriétés ci-dessus.
- Un algorithme de génération de signature $S(K_S, m)$ *produit une signature* S *à partir d'un message* m *et d'une clé privée* K_S.
- Un algorithme de vérification $V(K_P, m, S)$ *vérifie la signature* S *à partir d'un message* m *et d'une clé publique* K_P.
- Un schéma de signature est composé d'un algorithme de signature et d'un algorithme de vérification.
- Un procédé de signature comprend l'algorithme de signature ainsi que le formatage.

VII.2. Signatures

Le schéma de signature le plus couramment utilisé consiste à appliquer un algorithme de signature sur un haché d'un message plutôt que sur le message total (pour profiter du fait que le haché est plus court et de taille fixe).

VII.2.1. Signature R.S.A.

La signature R.S.A. utilise exactement les mêmes clés que pour le chiffrement R.S.A. La clé privée K_S d'Alice est un triplet (p, q, d) avec $N = pq$ et $ed = 1 \pmod{(p-1)(q-1)}$. Sa clé publique K_P est le couple (N, e).

Méthode **Signature R.S.A.**

1) *Signature.* Pour signer un message m, Alice calcule $\tilde{m} = h(m)$ pour une fonction de hachage h sans collision. Elle détermine ensuite $s = (\tilde{m})^d \pmod N$ avec sa clé privée.

2) *Vérification.* Bob reçoit un couple (m', s'). Pour vérifier la signature, il teste si $h(m') = (s')^e \pmod N$ avec la clé publique d'Alice.

Preuve. L'algorithme fonctionne car si m' et s' ont bien été construits, alors $(s')^e = (\tilde{m}^{ed}) \pmod N = \tilde{m} \pmod N = h(m') \pmod N) = h(m) \pmod N$. Réciproquement, si m a été altéré, alors $h(m') \neq h(m)$; et, si les bons couples de clés publiques/privées ne sont pas utilisés, alors $(s')^e \neq h(m)$ et donc la vérification ne peut se faire.
Vérifions maintenant que cette signature vérifie bien les cinq propriétés définies au début de la section.
- Personne ne peut imiter la signature car il faut connaître la clé secrète pour signer.
- Tout le monde peut vérifier à l'aide la clé publique associée à la personne.
- La signature n'est pas réutilisable car elle est associée à un haché donné et, par construction d'une fonction de hachage, on ne peut trouver de collisions.
- Si le message est altéré, la signature n'est plus valide car le haché sera modifié dans la vérification.
- Une signature ne peut être répudiée (dans le sens de nier que l'on a signé un message) car, tout d'abord, le signataire est le seul à connaître le secret associé à la clé publique et, de plus, on ne peut trouver de collisions sur le haché d'un message : l'auteur ne peut donc nier avoir signé un haché donné. ■

Une des nombreuses variantes de signatures digitales est la signature aveugle, où l'on souhaite pouvoir effectuer une signature sans que le signataire sache ce qu'il signe (voir le test ci-dessous). Un exemple d'application est le vote électronique : on peut imaginer qu'un bulletin, pour être valide, doive être signé par une autorité sans que celle-ci connaisse le vote.

Test 1.33.

Trouver un algorithme à partir de la signature R.S.A. qui permet de faire un algorithme de signature aveugle. On a un message m que l'on souhaite faire signer sans que le signataire sache ce qu'il signe, et un couple de clés publique/privée. Indice : multiplier le message par une valeur adéquate.

VII.2.2. Signature D.S.A.

On peut comprendre la signature R.S.A. comme une inversion du chiffrement puisque l'on signe avec la clé privée du chiffrement et que l'on vérifie avec la clé publique. Cela est dû au fait que, pour l'algorithme R.S.A., on peut voir les fonctions $x \to x^d \pmod N$ et $x \to x^e \pmod N$ comme inverses l'une de l'autre. Cette propriété est un peu exceptionelle et, en général, les algorithmes de chiffrement ne peuvent s'adapter aussi directement pour produire des algorithmes de signature.

Du fait que l'algorithme R.S.A. était breveté et qu'il était très facile de transformer la signature R.S.A. en un algorithme de chiffrement R.S.A., le N.I.S.T. a proposé en 1991 un standard de signature qui ne pouvait s'adapter simplement en schéma de chiffrement. Ce schéma est dérivé du schéma de signature d'El Gamal (voir le test plus loin). Il est basé sur le logarithme discret. Nous le décrivons ici.

Méthode **Digital Signature Algorithm (D.S.A.)**

On convient d'une fonction de hachage H (par exemple S.H.A.-1).

1) *Génération des clés.* On choisit un premier $q \in]2^{159}, 2^{160}[$, un entier $t \in [0, 8]$ et un nombre premier $p \in]2^{511+64t}, 2^{512+64t}[$ tels que $q|(p-1)$, c'est-à-dire

$$p - 1 = qz.$$

On prend un entier $h \in]1, p-1[$ et l'on pose $g \overset{déf}{=} h^z \pmod p$: g est alors d'ordre q.
On choisit un entier $x \in]1, q-1[$ et l'on calcule $y \overset{déf}{=} g^x \pmod p$.
Clé publique : (p, q, g, y). Clé privée : x.

2) *Signature d'un message* m. Alice choisit k avec $1 < k < q$ et calcule $r \equiv [g^k \pmod p] \pmod q$ et $s \equiv k^{-1}(H(m) + xr) \pmod q$. La signature est le couple (r, s).

3) *Vérification.* Bob reçoit (m, r, s). Il calcule $w \equiv s^{-1} \pmod q$, puis $u_1 \equiv wH(m) \pmod q$ et $u_2 \equiv rw \pmod q$. La signature est acceptée si

$$v \overset{déf}{=} [g^{u_1} y^{u_2} \pmod p] \pmod q = r.$$

Preuve. On remarque que $g^q \equiv h^{qz} \equiv h^{p-1} \equiv 1 \pmod p$, donc l'ordre de g divise q ; comme q est premier, l'ordre de g est exactement q.
On peut récrire la relation $s \equiv k^{-1}(H(m) + xr)$ en $k \equiv s^{-1}(H(m) + xr) \equiv u_1 + xu_2 \pmod q$. Ainsi, comme g a ordre q, on a $g^k \equiv g^{u_1} g^{xu_2} \equiv g^{u_1} y^{u_2} \pmod p$. Pour la validité, on vérifie $r \equiv [g^k \pmod p] \pmod q \equiv [g^{u_1} y^{u_2} \pmod p] \pmod q \equiv v$. ∎

Le N.I.S.T. a défini une norme D.S.S. (Digital Signature Standard) qui consiste à utiliser le D.S.A. avec une fonction de hachage comme S.H.A.-1.

Test 1.34.

Soit h une fonction de hachage à valeur dans F_2^n. On considère le cas d'un attaquant qui, pour abuser une signature, souhaite construire deux messages ayant le même haché, mais avec deux significations *choisies* comme étant différentes.
1) Donner une attaque en $O(2^{n/2})$ pour la fonction de hachage h qui permet de construire deux messages avec deux significations choisies différentes, mais avec le même haché.
2) Montrer comment l'attaque du 1) peut être utilisée avec la signature R.S.A. ou D.S.A. pour abuser un vérificateur de signature.

Test 1.35.

Pour la signature El Gamal, on prend comme clé publique $y = g^x \pmod p$ et comme clé privée x. La signature est composée de deux morceaux (r, s) avec $r = g^k \pmod p$ pour k aléatoire dans $[0..p-1]$ et $s = (h(m)-xr)k^{-1} \pmod{p-1}$ pour $h(m)$ un haché du message m. On accepte la signature (r, s) de m si $g^{h(m)} = y^r r^s \pmod p$.
1) Vérifier que le schéma fonctionne.
2) Quel est l'avantage de D.S.A. par rapport à cet algorithme en termes de la taille de la signature ?

VII.3. Infrastructure de gestion de clé

Un des problèmes majeurs de la cryptographie à clé publique est la gestion des clés (*Public Key Infrastructure*, ou P.K.I., en anglais). Comment être sûr qu'une clé publique correspond bien à une personne donnée ? C'est tout le problème (difficile en pratique) de la gestion des clés. Une manière d'appréhender cette situation est l'utilisation de certificats. On suppose l'existence d'une autorité (dite autorité de certification) CA comme l'État, la Poste, une banque ou autre, en qui on a confiance. On suppose que la clé publique de cette autorité est facilement disponible ou bien donnée en même temps que son propre couple de clé privée et publique. Un certificat émis par CA consiste en un message reliant l'identité d'un individu à sa clé publique, le tout étant signé par CA. On peut placer la liste des certificats dans un annuaire. Pour obtenir la clé publique de quelqu'un, on récupère un certificat associé à la personne. Comme on a la clé publique de CA, on peut vérifier que CA a signé le certificat reliant l'identité du possesseur de la clé et sa clé. Comme on fait confiance à CA, on peut alors utiliser la clé publique de la personne pour chiffrer par exemple.

Test 1.36.

Proposer une méthode à base de certificats pour vérifier une signature reçue par un individu, sans avoir à consulter un annuaire ou une base de données.

L'utilisation de certificats pose des problèmes pratiques comme la gestion des clés compromises ou la certification croisée entre plusieurs autorités.

VIII. Authentification

La fonction d'authentification est une fonction très importante. L'authentification la plus simple et la plus utilisée couramment est le login/password. Cette méthode a plusieurs inconvénients, comme le fait d'être sensible à une attaque par rejeu (où un attaquant intercepte le login/password et les rejoue). En effet, si quelqu'un intercepte le mot de passe, il est par la suite capable de se faire passer pour la personne. On peut imaginer des variantes qui complexifient un peu cette idée, comme le fait d'avoir une liste de mots de passe définis à l'avance ou d'ajouter des nombres aléatoires. On parle d'authentification faible.

Il est en revanche possible de faire de l'authentification à partir d'algorithmes de chiffrement symétriques dans le cas de l'existence d'un secret partagé.
L'idée générale, pour se protéger des attaques par rejeu, est de faire un schéma défi/réponse. Pour s'authentifier, plutôt que de donner une information ou un mot de passe qui peut être réutilisé, on envoie un défi pour lequel la réponse renvoyée montre que l'on connaît un secret (mais sans divulguer ce secret).

VIII.1. Authentification par chiffrement symétrique

Considérons le cas où deux personnes, P (le prouveur) et V (le vérifieur), souhaitent s'authentifier alors qu'elles ont un secret commun K (clé symétrique partagée) et un procédé de

chiffrement E_K (respectivement de déchiffrement D_K). On peut alors procéder de la manière suivante.

Méthode **Authentification à clé secrète**

1) Le vérifieur V envoie un défi (un nombre aléatoire) r_P à P.
2) P renvoie $E_K(r_P, id_V)$ à P, avec id_V l'identité de V (un numéro, un nom ou autre).
3) V déchiffre avec D_K et vérifie qu'il obtient r_P et son identité id_V.

Il existe beaucoup de variantes, avec de l'authentification mutuelle par exemple. Ce type d'authentification a été, par exemple, utilisé pour l'authentification par carte à puce ; il impose d'avoir accès à la clé secrète associée à une carte donnée.

VIII.2. Authentification à clé publique

Il est possible d'adapter cette même idée de défi/réponse dans un contexte de clé publique. Cela peut concerner le chiffrement ou la signature.

Test 1.37.

Proposer un schéma basé sur le chiffrement ou sur la signature pour faire de l'authentification à clé publique.

VIII.3. Schémas à divulgation nulle de connaissance (*zero-knowledge*)

Nous venons de voir comment faire de l'authentification (à clé publique) avec des schémas de signature ou de chiffrement. Souvenons-nous que ces protocoles sont plutôt coûteux en temps de calcul. En 1984, Shamir a introduit une nouvelle manière de procéder où l'on prouve que l'on connaît un secret avec une probabilité donnée ; cela permet d'ajuster le degré de sécurité recherché au coût de l'algorithme et permet de ne pas avoir à mener nécessairement un chiffrement ou une signature en entier.

Les schémas à divulgation nulle de connaissance (*zero-knowledge* en anglais) procèdent en trois étapes. Dans la première étape, le prouveur P envoie un *engagement* au vérifieur V ; V renvoie alors à P un défi, puis P renvoie sa réponse à V qui vérifie que la réponse est bonne.

On impose à ces schémas d'avoir les propriétés suivantes :

- cohérence : pour tout défi, P est capable de donner une réponse ;
- signifiance : la probabilité qu'un adversaire puisse se faire passer pour P est faible ;
- divulgation nulle : un adversaire qui écoute ne peut déduire aucune information sur le secret de P.

Le premier schéma proposé a été le schéma de Fiat-Shamir en 1986.

Méthode **Schéma d'authentification de Fiat-Shamir**

Soit $N = pq$ un module R.S.A. On choisit un nombre aléatoire $a \in [0, N-1]$ et l'on calcule $A \overset{\text{déf}}{=} a^2 \pmod N$ (a est une racine carrée de A).
Clé publique : (A, N). Clé secrète : (a, p, q).
Le protocole de Fiat-Shamir consiste à réitérer suffisamment, en fonction de la sécurité souhaitée, la séquence suivante (aussi appelée un *tour*) :

1) P choisit un entier aléatoire $k \in]0, N-1[$ et envoie $K = k^2 \pmod N$ à V ;
2) V envoie un défi r ($r = 0$ ou $r = 1$) à P ;
3) P calcule $y = ka^r \pmod N$ et renvoie y (la réponse) ;
4) V vérifie que $y^2 = KA^r \pmod N$.

Gros gain sur le coût (une seule multiplication) malgré les 2000 bits de com.
Très peu utilisé notamment à cause d'une probabilité trop faible

Ce schéma sert à authentifier P dans la mesure où il y a une très forte probabilité que celui qui aurait répondu correctement aux défis de V (en utilisant la clé publique de P) connaisse la clé secrète de P.
Pour le démontrer, il nous faut montrer qu'un attaquant qui ne connaît pas la clé privée de P a, au mieux, une chance sur deux de répondre convenablement à un test. En répétant le protocole t fois (les cryptographes disent souvent « en faisant t tours »), on arrive à une probabilité de falsification de 2^{-t}.

Preuve. On peut vérifier que l'algorithme fonctionne car $y = ka^r \pmod N$, $A = a^2 \pmod N$ et $K = k^2 \pmod N$. Vérifions les trois propriétés exigées d'un schéma à divulgation nulle de connaissance.
- Le protocole est bien cohérent puisque, pour chaque défi, on peut renvoyer une réponse.
- Quelle est la probabilité qu'un adversaire puisse se faire passer pour P ? Un adversaire peut anticiper le défi de V et y répondre sans connaître a. S'il anticipe $r = 0$, il procède normalement et renvoie $y = k \pmod N$; s'il anticipe $r = 1$, il choisit comme engagement $K = k^2A^{-1} \pmod N$, puis comme réponse $y = k \pmod N$. V vérifie alors que $y^2 = k^2 = KA = k^2A^{-1}A \pmod N = k^2 \pmod N$. Nous voyons donc qu'un attaquant peut anticiper soit $r = 0$, soit $r = 1$, mais pas les deux à la fois ; la probabilité d'arriver à se faire passer frauduleusement pour P est donc $1/2$.
- La propriété de divulgation nulle de connaissance vient du fait que si un attaquant observe des communications entre P et V, il ne voit jamais passer le secret de P (la racine carrée a), mais soit un entier aléatoire k, soit ka. Comme k est nouveau à chaque fois (et qu'il y en a beaucoup), l'attaquant ne peut en tirer aucune information. Bien sûr, P ne doit pas réutiliser le même k. ■

Étudions les paramètres de sécurité. Pour N, on prend un module R.S.A. avec la longueur de clé adéquate.
Si l'on savait calculer des racines carrées modulo N (i.e., étant donné un entier X, trouver un x tel que $x^2 \equiv X \pmod N$), alors on saurait factoriser N. En effet, X admet deux racines carrées modulo p (car p est premier donc $\mathbb{Z}/p\mathbb{Z}$ est intègre) et modulo q ; par le lemme chinois, X admet donc quatre racines distinctes modulo N. Supposons que l'on trouve x et y tels que $x \neq \pm y$ et $x^2 \equiv y^2 \pmod N$, alors $x^2 - y^2 = (x-y)(x+y)$ est un multiple de N : il suffit donc de calculer le PGCD de N et de $x - y$ ou $x + y$ pour retrouver des facteurs de N.

IX. Codes correcteurs d'erreurs et cryptographie à clé publique

En 1978, alors que R. Rivest, A. Shamir et L. Adleman proposaient le célèbre algorithme R.S.A., R. McEliece a proposé un algorithme de chiffrement basé sur les codes correcteurs d'erreurs. Cet algorithme, bien qu'étant plus rapide que R.S.A., a le gros désavantage d'impliquer une très grosse taille de clé publique (plusieurs centaines de milliers de bits). Cette dernière propriété limitant considérablement son utilisation, le système (bien que toujours non cassé) a été beaucoup moins étudié que les autres systèmes basés sur la théorie des nombres.

En 1994, P. Shor a proposé un algorithme utilisant un ordinateur quantique putatif, qui permettrait de factoriser un entier R.S.A. en temps polynomial par rapport à la taille de la clé. Plus tard, il a été montré que, si un ordinateur quantique venait à exister, tous les systèmes basés sur la factorisation ou le logarithme discret seraient cassés. L'algorithme de Shor ainsi que les progrès techniques sur la mémoire des ordinateurs ont conduit à un développement de l'étude de systèmes cryptographiques alternatifs aux systèmes basés sur la théorie des nombres et résistants aux algorithmes utilisant un ordinateur quantique.

Ainsi, outre les systèmes basés sur les codes (pour lesquels il n'est pas connu d'algorithme quantique attaquant spécifiquement le système), il existe aussi des systèmes basés sur les réseaux arithmétiques ou l'algèbre des polynômes multivariés qui pourraient aussi résister à des attaques quantiques. L'ensemble de ces systèmes est regroupé sous l'appelation *cryptographie post quantique*. Il faut cependant préciser que l'existence d'un ordinateur quantique efficace reste actuellement très hypothétique.

Nous présentons dans cette section le chiffrement de McEliece basé sur les codes correcteurs d'erreur. Nous supposons dans cette partie une connaissance du chapitre 7.

IX.1. Un problème difficile en théorie des codes

Lorsque l'on travaille en cryptographie, on cherche à se ramener à un problème difficile. Dans le cas des codes, le problème que nous utiliserons est le problème du décodage par syndrome (S.D. pour Syndrom Decoding).
Soit H une matrice $k \times n$ sur $\mathbb{F}_2^n$, soit s un mot (un syndrome) de $\mathbb{F}_2^k$ et soit w un entier positif. Le *problème du décodage par syndrome* est le suivant : Existe-t-il un mot x de $\mathbb{F}_2^n$ de poids au plus w tel que $Hx^t = s$?

Test 1.38.

Montrer que si l'on ne met pas de contrainte de poids dans le problème du décodage par syndrome, on obtient un problème très facile à résoudre.

Test 1.39.

Montrer que le problème du décodage d'un code aléatoire pour un mot avec une erreur de poids w peut se ramener à ce problème.

IX.2. Décodage d'un code aléatoire par ensemble d'informations

On suppose disposer d'une matrice génératrice G d'un code *aléatoire* $[n, k, d]$; on cherche à décoder un mot $y = xG + e$, où e désigne une erreur de poids t. Le *décodage par ensemble d'information* consiste à choisir aléatoirement k colonnes parmi n en espérant que ces k colonnes correspondent à des emplacements sans erreurs (un *ensemble d'information*).
Si l'on obtient k colonnes sans erreurs, alors on peut inverser la sous-matrice correspondante M (de taille $k \times k$) de G. On calcule M^{-1} appliquée aux positions sans erreurs de y : on retrouve

alors x, puis on calcule $\epsilon \stackrel{déf}{=} y - xG$. Si le poids de ϵ est t, alors on a bien décodé.

Toute la complexité de l'algorithme consiste à trouver un ensemble sans erreurs. On peut calculer la probabilité de trouver k colonnes parmi n en évitant t colonnes, c'est-à-dire $\binom{n-t}{k}$ divisé par le nombre $\binom{n}{k}$ de choix possibles. Comme une matrice aléatoire (carrée) sur $\mathbb{F}_2$ a une bonne probabilité d'être inversible (pour être précis il faut compter le nombre de bases de l'espace, mais, intuitivement, ce nombre est de l'ordre de $1/2$, puisque le déterminant peut valoir 0 ou 1), on en déduit la complexité de l'attaque par le nombre moyen d'essais nécessaires pour trouver un bon ensemble fois le coût de l'inversion de la matrice, soit $\mathcal{O}(k^3)\frac{\binom{n}{k}}{\binom{n-t}{k}}$.

IX.3. Schéma de chiffrement de McEliece

Méthode **Cryptosystème de McEliece**

1) *Génération des clés.* On considère C un code $[n, k, d]$ de matrice génératrice G que l'on sait décoder jusqu'à t erreurs. Soit S une matrice inversible $k \times k$ quelconque et soit P une matrice de permutation aléatoire sur n colonnes.
Clé publique : la matrice $G' = SGP$. Clé privée : S,G.

2) *Chiffrement d'un message* m. On calcule $c = mG' + e$ où e est une erreur aléatoire de poids t.

3) *Déchiffrement du message* c *reçu.* On calcule $y = cP^{-1}$; on décode y en mS, puis on retrouve m en multipliant par S^{-1}.

Preuve. Lorsque l'on calcule cP^{-1}, on obtient $y = x'G + eP^{-1}$ avec $x' = mS$. Comme la permutation P^{-1} préserve le poids, l'erreur eP^{-1} est toujours de poids majoré par t. Connaissant la matrice G, on peut décoder y en $x' = mS$ et retrouver m. ■

La clé publique G' est de taille kn. La taille de la clé privée est de l'ordre de $\mathcal{O}(n)$ (en prenant un ensemble court générant S et P).
La complexité du chiffrement est celle d'un produit matrice-vecteur, soit $\mathcal{O}(kn)$; la complexité du déchiffrement est celle du décodage du code, soit en général de l'ordre de $\mathcal{O}(n^2)$.
Si l'on prend k de l'ordre de $n/2$ ou $n/3$, on obtient donc que la taille de la clé publique (ainsi que le chiffrement et le déchiffrement) est en $\mathcal{O}(n^2)$.

En pratique, McEliece a proposé d'utiliser la famille des codes de Goppa binaires pour des paramètres $[1024, 524, 101]$ avec $t = 50$. Ces paramètres, proposés en 1978 pour une complexité de 2^{60}, n'ont été cassés pratiquement qu'en 2008. Ils conduisent à une taille de clé de l'ordre de 500 000 bits. Pour obtenir une complexité en 2^{80}, on peut par exemple prendre des codes de Goppa binaires de paramètres $[2048, 1600, 46]$. L'analyse de complexité précédente montre donc que ce système est beaucoup plus rapide que R.S.A. : pour un même n de l'ordre de 1024 bits, on obtient un déchiffrement cubique (pour R.S.A.) contre un déchiffrement quadratique (pour McEliece). Mais la très grosse taille de clé reste le problème principal pour l'utilisation pratique.

On peut potentiellement utiliser ce système avec de nombreuses familles de codes. Néanmoins, si l'on utilise une famille trop structurée comme les codes de Reed-Solomon généralisés ou les codes de Reed-Muller, il est possible de faire des attaques structurelles qui permettent de retrouver la structure du code masquée et les matrices S et P. L'intérêt des codes de Goppa est qu'il s'agit d'une famille très importante de codes que l'on sait décoder et pour lesquels la

structure (en raison de leur grand nombre) est plus difficile à retrouver.

Test 1.40.

Calculer la complexité de l'attaque pour les paramètres proposés originellement par McEliece.

X. Exercices

1.1. ⋆

Supposons qu'Alice et Bob partagent une clé aléatoire K dans $\{0,1,2\}$ et qu'Alice veuille envoyer à Bob un message M de $\{0,1,2\}$.
1) On suppose tout d'abord qu'elle procède en convertissant K et M en ensembles de deux bits (00, 01, 10) et qu'elle fait un XOR entre les deux représentations binaires. Montrer qu'un tel schéma n'est pas bon, en ce sens qu'il y a de l'information qui fuit et que ce schéma n'est pas parfaitement sûr. On pourra montrer que tous les chiffrés c_1, c_2 (où c_i est un bit) n'ont pas la même probabilité d'exister.
2) Proposer un autre schéma à base de modulo qui serait parfaitement sûr.

1.2. ⋆⋆

On considère les trois registres à décalage R_1, R_2 et R3, de polynômes de rétroaction respectifs $C_1(x) = x^3 + x + 1$, $C_2(x) = x^5 + x + 1$ et $C_3(x) = x^4 + x + 1$. On considère le générateur aléatoire obtenu en combinant les trois registres R_1, R_2 et R3 de sorties respectives x_1, x_2 et x_3, par la fonction booléenne $z = x_1 + x_2 x_3$.
1) Expliquer pourquoi il est plus intéressant d'attaquer par corrélation le registre R_1 que les registres R_2 et R_3. Donner la corrélation entre x_1 et z par une table de vérité et de façon théorique.
2) On suppose que l'on observe pour le générateur combiné précédent la sortie 1000011100. Retrouver l'initialisation du registre R_1.

1.3. ⋆⋆

On considère un module R.S.A., $n = pq$ et d l'exposant privé. Soit un m un message à signer. On cherche à calculer $S = m^d \pmod n$. On note $d_p = d \pmod{p-1}$, $d_q = d \pmod{q-1}$ et $i_q = q^{-1} \pmod p$. Soient $S_p = m^{d_p} \pmod p$ et $S_q = m^{d_q} \pmod q$.
1) Rappeler le théorème des restes chinois. Montrer que $S \pmod p = S_p$ et $S \pmod q = S_q$. Expliquer alors pourquoi on peut retrouver S à partir de S_p et S_q.
2) Montrer que $S = S_q + q(i_q \cdot (S_p - S_q)) \pmod n$.
3) Expliquer l'intérêt (en termes de coût calculatoire) de calculer S par cette méthode plutôt que directement en calculant $m^d \pmod n$?

1.4. ⋆⋆

Supposons que l'on ait 3 modules R.S.A. N_1, N_2 et N_3 distincts, mais que chacun des systèmes utilise la valeur d'exposant 3. Montrer que si un même message m tel que $m^3 < N_1 N_2 N_3$ est envoyé pour les trois modules N_1, N_2 et N_3, alors il est possible de retrouver m par les restes chinois.

1.5. ⋆⋆

Soit $N = pq$ un module R.S.A. Soient $a \in (\mathbb{Z}/N\mathbb{Z})^*$ et $g = aN + 1 \pmod{N^2}$. On considère le schéma de chiffrement suivant. La clé publique est (N, g). Pour chiffrer un message $m \in (\mathbb{Z}/N\mathbb{Z})^*$, on procède de la façon suivante : on prend un h aléatoire dans $\{1, \cdots, N-1\}$ et l'on calcule $C = g^m . h^N \pmod{N^2}$. On voudrait trouver un algorithme de déchiffrement.
1) Soit $x \in \{1, \cdots, N-1\}$. Montrer que, pour un g donné et $B = g^x \pmod{N^2}$, il existe un algorithme efficace pour retrouver $x \pmod N$ (c'est-à-dire que, pour $0 < x < N$, le problème de logarithme discret en base g est facile). On pourra utiliser le fait que $g = aN + 1$.
2) Montrer que, si l'on connaît g et la factorisation de N, déchiffrer $C = g^m . h^N \pmod{N^2}$ peut être réalisé efficacement. Montrer que $C \pmod N = h^N \pmod N$.
3) On veut établir que l'on peut construire des chiffrés à partir de chiffrés connus (on appelle cette propriété la *malléabilité*). Montrer qu'étant donné N ainsi que le chiffré de x et y, il est possible de construire le chiffré de $x + y$ et le chiffré de $c.x$ pour c dans $(\mathbb{Z}/N\mathbb{Z})^*$. On dit alors que ce chiffrement est un homomorphisme additif.

1.6. ⋆⋆

On suppose qu'un groupe de n personnes qui n'ont pas de secret commun veulent partager un secret commun pour communiquer entre elles de manière confidentielle. Proposer un schéma basé sur l'échange de clé de Diffie-Hellman qui permette cela. Compter le nombre global d'exponentiations modulaires nécessaires. Essayer d'optimiser ce nombre, par rapport à chaque

individu et globalement.

1.7. ⋆

On considère le schéma de signature suivant. On suppose que l'on a une fonction de hachage f qui renvoie des hachés de longueur n. On va maintenant expliquer comment, à partir de f, on peut signer un message m de longueur k. Notons $m = (m_1, m_2, \ldots, m_k)$ avec $m_i \in \{0,1\}$. Pour $1 \leq i \leq k$ et $j \in \{0,1\}$, on prend $2k$ valeurs aléatoires y_{ij} de longueur k et l'on calcule $z_{ij} = f(y_{ij})$. Les $2k$ nombres z_{ij} forment la clé publique et les $2k$ nombres y_{ij} sont la clé secrète. Pour signer un message $m = (m_1, m_2, \ldots, m_k)$ de k bits, on a

$$\begin{aligned} \mathtt{Signature}(m) &= (y_{1m_1}, y_{2m_2}, \ldots, y_{km_k}) \\ &= (s_1, \ldots, s_k). \end{aligned}$$

1) Calculer pour $n = 256$ et $k = 256$ les tailles des clés publiques et privées. Comparer aux tailles de clés pour R.S.A. ou D.S.A.
2) Justifier que pour une seule signature, la sécurité du schéma repose sur la sécurité de la fonction de hachage f.
3) Peut-on prendre k petit pour le protocole, par exemple $k = 1$ ou 2 ?
4) Montrer qu'en prenant une attaque à messages choisis en deux signatures pour deux messages choisis, on peut récupérer toute la clé publique. Justifier la notion d'usage unique pour ce protocole.

1.8. ⋆

On considère le protocole de Schnorr. Soient p et q deux entiers (grands) tels que q divise $p-1$, et soit g un entier d'ordre q modulo p. Le secret détenu par Alice est un entier $a \in [0, q-1]$ et la donnée de $A = g^{-a} \pmod p$ est rendue publique. Le protocole est alors le suivant :
- Alice fournit un engagement aléatoire k dans l'intervalle $[0, q-1]$ et calcule $K = g^k \pmod p$. Elle transmet K à Bob.
- Bob choisit un défi r au hasard dans $[0, q-1]$ et le transmet à Alice.
- Alice calcule la réponse $y = (k + ar) \pmod q$ et la transmet à Bob. Bob vérifie que $g^y A^r = K \pmod p$.
1) Faire un schéma de ce protocole. Vérifier que le protocole fonctionne.
2) Montrer que le protocole est cohérent et signifiant pour une probabilité $1/q$.
3) Quel est l'intérêt de ce protocole par rapport au protocole de Fiat-Shamir (en termes de nombre de passes).

1.9. ⋆⋆⋆

Il est possible de faire un algorithme de signature avec un protocole de type zero-knowledge avec une probabilité de triche donnée.
1) Montrer qu'en fixant par avance les engagements dans un protocole à divulgation nulle de connaissance et en les reliant par une fonction de hachage aux défis, on peut obtenir une signature pour une sécurité quelconque.
2) Dans le cas de Fiat-Shamir, avec $n = 1024$, quelle est la taille de la signature ?

1.10. ⋆⋆⋆

Pour des raisons de sécurité, il peut être intéressant que seule une coalition d'un nombre k de personnes parmi n soit capable de retrouver un secret. Par exemple, au moment de la création de la clé privée, on suppose qu'une autorité, avant de détruire la clé (et après l'avoir transmise à son propriétaire), donne certains éléments liés à la clé à 3 individus, de telle sorte qu'en cas de perte de clé privée, au minimum 2 individus parmi les 3 doivent collaborer pour retrouver cette clé privée, et cela afin de limiter les fuites possibles sur la clé tout en gardant un moyen de la retrouver.
Proposer une méthode pour partager un secret utilisant des polynômes de degré k et la notion d'interpolation de Lagrange pour distribuer des données à n personnes, de telle sorte qu'au moins k personnes parmi ces n doivent collaborer pour retrouver le secret et qu'une coalition de $k-1$ ne puisse rien retrouver.

1.11. ⋆

1) On suppose qu'Alice partage une clé de chiffrement par blocs K_{AB} avec Bob et une clé K_{AC} avec Charlie. Donner une méthode pour qu'Alice chiffre un long message M de m blocs qui ne soit déchiffrable que par une coopération entre Bob et Charlie. On peut supposer que Bob et Charlie partagent un canal secret pour leur communication. Le chiffré devra être de taille fixe, à peine plus grand que m blocs et Alice ne devra chiffrer les m blocs du message qu'une seule fois.
2) On suppose maintenant qu'Alice partage une clé de chiffrement par blocs avec Bob (K_{AB}), Charlie (K_{AC}) et David (K_{AD}). Donner une méthode de chiffrement de telle sorte qu'Alice envoie un message de m blocs chiffrés par une seule clé, mais que pour déchiffrer le message, il y a besoin qu'au moins deux personnes parmi Bob, Charlie et David coopèrent pour déchiffer le message. Indice : il faut ajouter simplement trois blocs chiffrés bien choisis aux m blocs chiffrés.
3) Donner une idée de la méthode pour généraliser cette idée au cas de n personnes dont tout sous-groupe de k personnes puissent déchiffrer le message (on pourra utiliser l'exercice précédent).

1.12. ⋆⋆

Pour le chiffrement de McEliece, la taille de la clé publique est nk ; il pourrait être plus intéressant de mettre cette matrice sous forme systématique pour diminuer sa taille. Le problème est que l'on risque alors, en chiffrant (en multipliant par une

telle matrice), de donner des informations sur le message.
Proposer une variation sur le système de chiffrement en utilisant une fonction de hachage sur l'erreur ajoutée, qui permet d'utiliser pour le chiffrement une matrice sous forme systématique.

1.13. ⋆⋆⋆

Un des problèmes avec les codes est qu'il n'y a pas de signature très efficace. On se propose de trouver un tel schéma.

1) Montrer que la densité de l'ensemble des mots de l'espace décodables par un code de Goppa $[2^m, 2^m - mt, 2t + 1]$ pour t relativement petit est $1/t!$.
2) En déduire un algorithme de chiffrement utilisant une fonction de hachage où l'on décode jusqu'à trouver un mot décodable provenant d'une variation sur le haché du message.
3) Proposer des paramètres pour $t = 9$ et une sécurité de 2^{80}. Que pensez-vous des paramètres de cet algorithme ?

Première partie
Solutions des tests

Deuxième partie

Solutions des tests

0.1. Si l'on connaît deux clairs et deux chiffrés, comme il y a deux inconnues, on se ramène à la résolution d'un système linéaire avec deux inconnues et deux équations.

0.2. TEL EST PRIS QUI CROYAIT PRENDRE.

0.3. Dans ce cas, la matrice M est inconnue. Si l'on passe les coefficients de M en inconnues, on se ramène à la résolution d'un système linéaire. Un message et son chiffré donnent trois équations. Avec au minimum trois clairs et leurs chiffrés, on peut retrouver les neuf coefficients inconnus de M.

0.4. On applique la procédure de déchiffrement en reprenant le processus à l'envers.

0.5. Toutes les opérations impliquées dans le chiffrement sont linéaires. On peut facilement contourner l'addition de la clé de tours en considérant la somme des chiffrés $C_1 \oplus C_2$ par rapport à la somme des clairs $M_1 \oplus M_2$. Dans ce cas, l'action de la clé de tour est annulée car dans le chiffrement, à chaque tour, on va sommer deux fois la clé de tour qui va donc s'annuler. Il existe alors une relation linéaire entre $M_1 \oplus M_2$ et $C_1 \oplus C_2$. Avec suffisamment de couples clairs/chiffrés, déchiffrer un message se ramène donc à résoudre un système linéaire.

0.6. La preuve suit le cas du double D.E.S., en remplaçant 56 par n, et le fait que, dans ce cas, la recherche dans une liste triée se fait en $\mathcal{O}(n)$ opérations.

0.7. Pour le D.E.SV, on peut écrire $m \overset{D.E.S._{K_2}}{\longrightarrow} r \overset{\oplus K_1}{\longrightarrow} c$. En prenant pour l'attaque par le milieu la valeur r, on a séparé le chiffrement en deux parties indépendantes ne dépendant que de K_1 ou K_2 ; on peut donc mener l'attaque par le milieu. De même, pour D.E.SW : $m \overset{\oplus K_2}{\longrightarrow} r \overset{D.E.S._{K_1}}{\longrightarrow} c$. Pour le D.E.SX, le fait d'avoir trois clés empêche de casser en deux morceaux indépendants et l'on ne peut mener cette attaque.

0.8. On procède avec une attaque par le milieu en $m \overset{D.E.S._{k_2} D.E.S._{k_2}}{\longrightarrow} r \overset{D.E.S._{k_1}}{\longrightarrow} c$. On prend r pour faire l'attaque au milieu avec l'ensemble des clés k_2 d'un côté et l'ensemble des clés k_1 de l'autre.

0.9. On applique la procédure de déchiffrement.

0.10. La suite engendrée est 1001000111101011001..., de période maximale 15.

0.11. On remarque que $C(x) = (1+x)(1+x+x^3)$ n'est pas irréductible. La suite donne 10011101001... période de longueur 7.

0.12. La suite chiffrante pour les 6 premiers bits est 110100. On résout le système linéaire indiqué dans le paragraphe avec, comme inconnues, les coefficients de $C(x)$: on trouve $C(x) = 1 + x + x^3$. On en déduit la suite de la suite chiffrante 1101001110100111 et le message 1101010101010101.

0.13. $5 \pmod{13}$ et $13 \pmod{15}$.

0.14. 33

0.15. $\phi(112) = \phi(7.16) = \phi(7).\phi(2^4) = 6.8 = 48$. $\phi(84) = \phi(3)\phi(4)\phi(7) = 2.2.6 = 24$.

0.16. Si α est un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$, alors tous les α^i tels que i est premier avec $p-1$ sont des générateurs.

0.17. 2 est un générateur de $(\mathbb{Z}/11\mathbb{Z})^*$. $(\mathbb{Z}/18\mathbb{Z})^* = \{1, 5, 7, 11, 13, 17\}$, 5 est un générateur.

0.18. Pour n personnes, le nombre de couples de clés est $n(n-1)/2$.

0.19. $\phi(55) = 40$. $e = 7$, $7.23 \pmod{40} = 1$, soit $d = 23$ $28^7 \pmod{55} \equiv 52, 52^{23} \pmod{55} \equiv 28$.

0.20. Si e_A et e_B sont premiers entre eux, l'algorithme d'Euclide étendu produit deux entiers u et v tels que $ue_A + ve_B = 1$. Supposons par exemple que $u > 0$ et $v < 0$ (donc $-v > 0$). Alors on peut calculer d tel que $d.c_B \equiv 1 \pmod{N}$. Comme $c_A = m^{e_A} \pmod{N}$ et $c_B = m^{e_B} \pmod{N}$, alors $c_A^u * d^{-v} \equiv c_A^u * c_B^v \equiv m^{ue_A + ve_B} \pmod{N} \equiv m \pmod{N}$.

0.21. Soit $c = m^d$. Pour un nombre x pris au hasard, on envoie $c' = x^e c$. On reçoit $c'^d = xm$, puis on multiplie par x pour retrouver m.

0.22. Au lieu d'utiliser tous les bits (par exemple 1024), on prend simplement 944 bits pour le message et l'on ajoute 80 bits aléatoires, qui changent pour chaque message.

0.23. Non, le chiffrement est séquentiel ; la complexité est la même pour attaquer chaque module. Cela ne change donc rien globalement.

0.24. Dans ce cas, le groupe multiplicatif a pour ordre 100 donc on peut prendre $d = 67$: comme $3.67 = 1 \pmod{100}$, on peut déchiffrer.

0.25. Si c'est 1, le message est envoyé en clair ; quant à 2, il n'est jamais premier avec $\phi(N)$ qui est toujours pair.

0.26. $3^7 \pmod{17} = 11, 3^{13} \pmod{17} = 12, 12^7 = 11^{13} = 7 \pmod{17}$.

0.27. Si pour x' donné, on peut trouver x tel que $h(x) = h(x')$, alors on peut utiliser ce même couple pour montrer que h n'est pas sans collision. Donc *non* faiblement sans collision implique *non* sans collision : par contraposée, on obtient le résultat. Si une fonction n'est pas faiblement sans collision, alors, pour x donné, on peut trouver x' tel que $h(x') = h(x)$. Comme on est capable de trouver un antécédent à $y = h(x)$, la fonction n'est pas à sens unique.

0.28. Tout d'abord, h n'est pas sans collision puisque l'on peut facilement trouver un antécédent pour tout y de longueur $n + 1$ de la forme $1||x$, puisqu'alors $h(x) = y$. Maintenant, supposons que h ne soit pas sans collision ; alors, il existe x et x' avec $x \neq x'$ tels que $h(x) = h(x')$. Si x et x' ne sont pas de longueur n, alors cela implique une collision sur g, ce qui n'est pas possible car g est sans collision. Si x et x' ont la même longueur n, alors $x = x'$, ce qui n'est pas non plus possible. Si enfin x ou x' a pour longueur n et l'autre pas, leurs hachés ne peuvent être égaux, donc h est sans collision.

0.29. La fonction $x \to x^2 \pmod{n}$ est un exemple de telle fonction. Elle est bien à sens unique. En revanche, si l'on connaît une racine carrée x de $x^2 \pmod{n}$, alors $-x \neq x$ est aussi une racine avec le même carré.

0.30. S'il existe une collision sur h alors, par construction, on peut trouver une collision pour g_1 et g_2, ce qui n'est pas possible puisqu'au moins une des deux est sans collision.

0.31. $1/10.2/9 = 1/45$.

0.32. 1) Si l'on est capable d'inverser E, alors on est capable de trouver un antécédent pour h. 2) h^n n'est pas sans collision car il suffit de prendre une permutation sur les m_i pour trouver une collision.

0.33. Soit m le message à signer. On cherche à obtenir $m^d \pmod{N}$. On envoie $x^e m$ pour signature. On reçoit donc en retour $(x^e m)^d = xm^d \pmod{N}$. Comme x est aléatoire, le signataire n'a aucun moyen de savoir quel message est signé.

0.34. 1) Supposons que l'on ait un message de $n/2$ mots avec une signification donnée. On prend le même message, mais en ajoutant soit un nouveau caractère blanc entre les mots, soit en ne le faisant pas. Cela donne $2^{n/2}$ message qui ont le même sens, mais il peut y avoir un léger décalage entre les mots auxquels on ne fait pas forcément attention. Par le paradoxe des anniversaires, on peut donc trouver une collision pour un haché de longueur n avec ces $2^{n/2}$ possibilités. 2) La vérification de la signature se fait uniquement à partir du haché d'un message, donc deux messages avec le même haché auront la même signature.

0.35. Par définition, on a $h(m) = xr + ks \pmod{p-1}$. Donc, par le petit théorème de Fermat, on a $g^{h(m)} = g^{xr} g^{sk} = (g^x)^r (g^k)^s = y^r r^s \pmod{p}$. L'intérêt de D.S.A. est que la signature a 320 bits contre 1024 pour El Gamal (à securité équivalente).

0.36. On suppose que tout le monde a la clé publique d'une autorité. Pour signer un message, on envoie le message signé avec sa clé publique plus le certificat de la clé signé par l'autorité. En recevant le message, on commence par vérifier la validité du certificat avec la clé publique de l'autorité. Si c'est correct, la clé publique du signataire est validée et l'on peut vérifier la signature. De cette façon, il suffit simplement d'avoir la clé publique d'une autorité et l'on n'a pas besoin de passer par un annuaire. La vérification se fait donc sans connexion nécessaire.

0.37. On établit un protocole de défi-réponse. Par exemple, pour du chiffrement à clé publique, on veut authentifier la personne qui connaît la clé secrète associée à une clé publique. On choisit un message aléatoire que l'on chiffre avec la clé publique. Le défi est de déchiffrer le message chiffré. Si la personne est capable de renvoyer le message déchiffré, c'est qu'elle connaît le secret. Pour la signature, le défi est de signer un message aléatoire que l'on vérifie avec la clé publique.

0.38. Soit une matrice H $k \times n$. S'il n'y a pas de contraintes de poids, on trouve une sous-matrice M $k \times k$ de H inversible (comme le déterminant vaut 0 ou 1, on peut facilement en trouver si on la prend au hasard). Une solution du problème est donc $x = M^{-1} s$ en mettant à 0 les colonnes ne correspondant pas à M.

0.39. On se donne un code de matrice génératrice G, de matrice de parité H et $y = xG + e$ avec $w(e) = w$. On a $H.y^t = H.e^t$ car $G.H^t = 0$. Ainsi, trouver e consiste à résoudre le problème du décodage par syndrome pour $s = H.y^t$ et le poids w de e.

0.40. Avec la complexité proposée, on obtient à peu près 2^{70} en pratique. Avec des améliorations sur l'attaque, on la ramène à une complexité inférieure à 2^{60}.

Troisième partie
Solutions des exercices

Quatrième partie
Solutions des exercices

0.41. 1) On fait un tableau avec, en entrée, pour les lignes et les colonnes 00, 01 et 10 correspondant aux clés et aux clairs possibles. On voit que le chiffré 00 a $1/3$ de chances d'arriver (pour des entrées et clés aléatoires) et les autres $2/9$, donc il y a une fuite d'information et le fait de connaître le chiffré donne un biais sur le clair.
2) On écrit les trois possibilités sous la forme $\{0,1,2\}$ et les clés aussi. Le chiffrement est la somme modulo 3. De cette façon, tout est symétrique et il n'y a pas de fuite d'information.

0.42. 1) On attaque sur toutes les initialisations possibles ; R_1 est de longueur plus petite donc on aura moins d'essais à faire. On a $z = x_1$ dès que $x_2 = 0$ ou $x_3 = 0$, soit $3/4$ du temps.
2) On essaie les huit initialisations possibles, l'initialisation 101 donne 101001110, qui a une corrélation proche de $3/4$.

0.43. 1) On a $d = d_p + k(p-1)$ pour $k \in \mathbb{Z}$. Alors, $S \pmod p = m^d = m^{d_p + k(p-1)} \pmod p = m^{d_p}(m^{p-1})^k \pmod p$. Comme p est premier, $m^{p-1} = 1 \pmod p$, ce qui donne le résultat.
2) On peut vérifier directement l'égalité ou la retrouver par le théorème des restes chinois puisque l'on connaît $S \pmod q$ et $S \pmod p$.
3) Pour retrouver S, il suffit alors de calculer S_p et S_q. Si l'on prend S avec n bits, p et q auront à peu près $n/2$ bits. Le calcul de S se fait en $\mathcal{O}(n^3)$ et celui de S_p et S_q en $\mathcal{O}(\frac{n^3}{8})$ chacun. On fait deux calculs, donc la complexité pour calculer S_p et S_q est en $\mathcal{O}(\frac{n^3}{4})$. Comme retrouver S a simplement le coût d'une multiplication, on gagne un facteur 4.

0.44. Le chiffré c_1 vérifie $c_1 = m^3 \pmod{N_1}$; de même, $c_2 = m^3 \pmod{N_2}$ et $c_3 = m^3 \pmod{N_3}$. Par les restes chinois, on peut donc retrouver $m^3 \pmod{N_1N_2N_3}$. Comme $m^3 < N_1N_2N_3$, on peut enlever le modulo et retrouver m.

0.45. 1) On a $g = aN + 1$. Si $B \equiv g^x \pmod{N^2}$ alors, par le développement avec le binôme de Newton, on obtient $(N+1)^x = 1 + xaN + N^2(P(N))$, où P est un certain polynôme en N. Comme on travaille modulo N^2, on obtient $g^x \pmod{N^2} \equiv 1 + xaN \pmod{N^2} \equiv B$, d'où $(B-1) = xaN + kN^2$ pour $k \in \mathbb{Z}$. Il en découle que $x \equiv a^{-1}\frac{(B-1)}{N} \pmod N$.
2) $C^{\phi(N)} \pmod{N^2} \equiv g^{\phi(N)m}h^{N\phi(N)} \pmod{N^2}$. Mais, comme $\phi(N^2) \equiv N\phi(N)$, $h^{N\phi(N)} \equiv 1 \pmod{N^2}$, on obtient $C^{\phi(N)} \equiv g^{\phi(N)am} \pmod{N^2}$. Or $\phi(N)a$ est inversible mod N^2 donc on peut appliquer le 1).
3) $\text{Chiffré}(x) = g^x h_x^N$ et $\text{Chiffré}(y) = g^y h_y^N$ donc $\text{Chiffré}(x+y) = g^{x+y}(h_x h_y)^N$ ($h_x h_y$ peut être considéré comme un h aléatoire). De même, $\text{Chiffré}(ux) = (g^x)^u (h_c^u)^N$.

0.46. On désigne un leader qui choisit un secret a. Il reçoit de chacun des membres du groupes M le nombre $x = g^{b\overline{M}^1} \pmod p$ et renvoie $y = x^a \pmod p$. Chaque membre du groupe calcule alors $y^{bM} \pmod p$ et obtient le même $g^a \pmod p$.

0.47. 1) $2 \times 256 \times 256 = 16$ kilobits. C'est beaucoup plus gros que les autres tailles de clés. 2) Pour casser le schéma, il faut être capable d'inverser la fonction de hachage pour les valeurs liées à la clé. C'est a priori infaisable d'après les propriétés de la fonction de hachage. 3) Non, car il n'y aurait pas assez de signatures possibles (2^k). 4) À chaque fois que l'on donne une signature, on donne des antécédents pour les valeurs de la signature. Si l'on prend deux vecteurs $a = (a_1, \cdots, a_k)$ et $b = (b_1, \cdots, b_k)$ tels que $a_i \neq b_i$, alors on peut obtenir tous les antécédents et l'on est capable de signer.

0.48. 1) On a bien $g^y A^r = g^{k+ar}.g^{-ar} = g^k = K \pmod p$.
2) Le protocole est cohérent car on peut toujours donner une réponse au défi. Dans le protocole, on peut anticiper le défi 0 donc la probabilité de tromperie est au moins $1/q$. On ne peut anticiper plus d'un défi.
3) Si l'on prend $q = 2^k$, on obtient une sécurité en un tour quand il en faudrait k pour Fiat-Shamir.

0.49. 1) On choisit au départ un paramètre de sécurité en 2^{-k}. On choisit k engagements distincts comme si l'on allait faire k tours. Ces engagements sont concaténés au message à signer. On hache alors le tout et l'on en extrait k défis. On calcule les réponses associées aux défis. La signature est le message plus les engagements prédéfinis ainsi que les réponses. Pour vérifier la signature, on vérifie que les défis ont été calculés convenablement et l'on vérifie le protocole. Comme les défis sont choisis aléatoirement (à cause de la fonction de hachage) après le choix des engagements, on est sûr qu'ils n'ont pas été anticipés.
2) Pour une sécurité en 2^{80}, il faut prendre 160 bits de défi pour éviter les attaques par paradoxe des anniversaires, soit $k = 160$. Puisqu'en un tour il y a 2.1024 bits échangés en tout, cela donne une signature de 320 kilobits.

0.50. Soit la valeur secrète r. On choisit un corps fini tel que le secret soit de la taille d'un élément du corps. Par exemple, un secret de 128 bits donne le corps $K = \mathbb{F}_{2^{128}}$. Pour un secret retrouvable par au moins k personnes parmi n, on choisit un polynôme f sur $K[x]$ de degré k tel que $f(0) = r$ (il suffit de fixer la constante de f à r). On distribue alors aux n personnes une valeur $f(a_i)$, pour des a_i distincts et non nuls. Si un groupe de k personnes collabore, elles sont capables de reconstruire f par interpolation (voir le chapitre 2) car f est de

degré k. Si moins de k personnes collaborent, elles retrouveront un polynôme à une indéterminée près. Comme le secret est justement une constante, elles ne peuvent retrouver d'informations supplémentaires sur le secret.

0.51. 1) A choisit une clé de session k puis la casse en deux morceaux k_1 et k_2. Elle envoie un message commençant par $E_{K_{AB}}(k_1)$ et $E_{K_{AC}}(k_2)$, puis $E_k(M)$. Pour déchiffrer, B et C auront à collaborer pour reconstruire k à partir de k_1 et k_2.
2) On écrit cette fois $k = k_1 + k_2 + k_3$ et l'on envoie, chiffrés par les clés symétriques, le couple $\{k_1, k_2\}$ à B, le couple $\{k_2, k_3\}$ à C et le couple $\{k_1, k_3\}$ à D avant d'envoyer le message M chiffré par la clé de session k. Si deux parmi trois collaborent, ils peuvent retrouver k_1, k_2 et k_3 puis retrouver k, mais un seul ne le peut pas.
3) On peut soit généraliser un point de vue combinatoire comme au 2). soit appliquer le schéma de partage de secret de Shamir de l'exercice précédent.

0.52. Lorsque l'on chiffre, on calcule $mG' + e$. Si G' est sous forme systématique, comme l'erreur est proportionnellement faible, on peut retrouver des informations sur les premiers bits de m. Pour pouvoir quand même utiliser une forme systématique, on fixe e et, au lieu de chiffrer m, on chiffre $m + h(e)$ pour une fonction h de hachage. Pour déchiffrer, on trouve l'erreur e et l'on décode $m + h(e)$. On peut donc retrouver $m = m + h(e) - h(e)$.

0.53. 1) Pour t petit, le nombre de mots dans une boule de rayon t sur $\mathbb{F}_2^n$ est très proche de $\binom{n}{t}$ (dans notre cas $n = 2^m$). La densité des mots décodables est donc le nombre de mots du code fois le nombre de mots dans la boule divisé par le nombre de mots de l'espace, soit $\frac{\binom{n}{t} 2^{2^m - mt}}{2^{2^m}}$. On peut approcher $\binom{n}{t}$ par $\frac{n^t}{t!}$ ce qui, en remplaçant dans l'égalité de la densité, donne une densité en $1/t!$.
2) La question 1) nous dit que si l'on prend un mot au hasard dans l'espace, il y a en moyenne une chance sur $t!$ qu'il soit décodable. On fait donc une signature qui fonctionne comme une fonction *pseudo-inversible* de type R.S.A. (mais pour R.S.A., la signature vient de l'inversabilité de la fonction de chiffrement). On reprend les mêmes clés publiques/privées que pour le schéma de McEliece. À partir du message à signer m, on peut donc mettre en place un compteur et décoder un mot de l'espace $\mathbb{F}_2^m$ dérivant de $h(m \oplus i)$ (pour une fonction h de hachage) en augmentant i jusqu'à ce que l'on obtienne un mot décodable (en moyenne au bout de $t!$ essais). Le secret est donc le fait d'être capable de décoder G'. Mais comme on ne sait le faire que pour certains mots de l'espace, on dérive des mots aléatoires de l'espace jusqu'à pouvoir les décoder. La signature est donc le mot de l'espace $x = h(m \oplus i_0)$ et son décodage par la matrice G'. Pour vérifier, on vérifie que $x = h(m \oplus i_0)$ et que le mot décodé est bien dans le code.
3) Comme il faut faire $t!$ décodages en moyenne avant de trouver le bon mot, on ne peut pas prendre t trop grand. En même temps, pour t petit, on peut attaquer plus facilement. Si l'on se fonde sur l'attaque décrite dans le cours, on obtient $m \geq 16$. En stockant uniquement la matrice duale, on obtient une clé d'au moins 8 mégabits. C'est beaucoup ! Mais cela donne un schéma de signature avec les codes.

Index