

Table des matières

Partie I Algèbre Appliquée	1
1 Introduction à la cryptographie	3
I Rapide historique de la cryptographie	5
II Rappels d’arithmétiques	9
II.1 Inversion modulo N et calcul modulaire	9
II.2 Générateurs de $(\mathbb{Z}/N\mathbb{Z})^*$	11
III Chiffrement à clé secrète	12
III.1 Chiffrement par blocs	12
III.2 Chiffrement à flot	17
IV Chiffrement à clé publique : R.S.A	21
IV.1 Algorithme R.S.A	21
IV.2 Mise en oeuvre des calculs	22
IV.3 La sécurité de R.S.A	23
IV.4 Génération de nombres premiers	24
V Echange de clés et chiffrement basés sur le logarithme discret	25
V.1 Problème du logarithme discret	25
V.2 Echange des clés de Diffie-Hellman	26
V.3 Chiffrement d’El Gamal	27
VI Fonctions de hachage	27
VI.1 Définitions	28
VI.2 Attaque par paradoxe des anniversaires	28
VI.3 Fonction de hachage itérée	29
VI.4 Fonctions de chiffrement spécialisées	30
VII Signature	31
VII.1 Définition	31
VII.2 Signatures	32
VII.3 Infrastructure de gestion de clé	34
VIII Authentification	34
VIII.1 Authentification par chiffrement symétrique	35
VIII.2 Authentification à clé publique	35
VIII.3 Schémas à divulgation nulle de connaissance (« zero-knowledge »)	35
IX Codes correcteurs d’erreurs et cryptographie à clé publique	37
IX.1 Un problème difficile en théorie des codes	37
IX.2 Décodage d’un code aléatoire par ensemble d’informations	37
IX.3 Schéma de chiffrement de McEliece	38
X Exercices	39
Partie II Solutions des tests	42
Partie III Solutions des tests	43
Partie IV Solutions des exercices	45
Partie V Solutions des exercices	46

Première partie
ALGÈBRE APPLIQUÉE

Chapitre 1

INTRODUCTION À LA CRYPTOGRAPHIE

LA notion de cryptographie remonte très loin dans le temps. Historiquement, et jusqu'à relativement récemment, elle était surtout utilisée à des fins 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 s'est peu à peu transformée d'une *science du secret* en une *science de la confiance* au fur et à mesure que de nouvelles applications apparaissaient ; en particulier, le développement du réseau Internet 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 problématiques 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.

Clés privées et clés publiques. Historiquement la cryptographie permettait à deux personnes partageant un secret commun de pouvoir échanger entre elles des messages chiffrés qu'elles seules étaient censés pouvoir déchiffrer à partir de leur secret (ou clé) commune, la clé pouvant être utilisée à la fois pour chiffrer et déchiffrer un message donné. On parle alors de *cryptographie symétrique*. Les limites de ce type de cryptographie 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 on veut 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 (qu'on appelle traditionnellement Alice et Bob ou, plus prosaïquement, A et B) mais plutôt d'un couple de clés (ou bi-clé) : 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 puisse 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 problématique, comme par exemple la signature, la cryptographie à clé publique reste incontournable.

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

- ◇ 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.
- ◇ INTÉGRITÉ DES DONNÉES : ou comment être certain qu'un document n'a pas été modifié
- ◇ AUTHENTIFICATION : Bob veut être sûr que le message a bien été envoyé par Alice.

- ◊ 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).
- ◊ CONTROLE D'ACCÈS : être capable de contrôler qui accède aux données.

La cryptologie permet de proposer des outils concrets pour résoudre les quatre premières problématiques, la dernière relevant plutôt de la sécurité des systèmes et des réseaux.

Exemples de protocoles relevant de la cryptologie. En plus de ces mécanismes de base pour la sécurité des systèmes d'information, la cryptologie s'intéresse à des protocoles plus spécifiques, par exemple :

- ◊ 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.
- ◊ PARTAGE DE SECRET : comment partager un secret à plusieurs, par exemple si on veut qu'un document ne soit reconstitué que si plusieurs personnes interagissent entre elles nécessairement.
- ◊ ÉCHANGE DE CLÉS : comment deux personnes qui n'ont pas de secret préalable en commun, puisse être capable de partager un secret commun.
- ◊ VOTE ÉLECTRONIQUE : comment voter de manière anonyme.
- ◊ ARGENT ÉLECTRONIQUE : comment manipuler de l'argent électronique sans pouvoir faire de la fausse monnaie.
- ◊ etc...

La cryptographie est un domaine en évolution rapide et de nouvelles problématiques apparaissent régulièrement. Les outils permettant de donner des solutions à ces problèmes sont essentiellement basés sur les problèmes de théorie des nombres comme la factorisation ou le logarithme discret mais pas seulement.

Attaques sur un système. On dit qu'un système est cryptanalysé si on met en évidence une attaque qui permet de faire mieux que la meilleure attaque connue sur le système (en général) sur laquelle est basée la sécurité d'un système. On parle de 'cassage total' d'un système, si on est capable de déchiffrer un message (ou de signer le cas échéant) sans connaître la clé (ou *a fortiori* de retrouver la clé). Il existe plusieurs niveaux de cryptanalyse, par exemple on peut cryptanalyser des paramètres donnés ou donner une attaque qui peut casser même des paramètres largement améliorés (c'est par exemple le cas si on trouve une attaque en temps polynomial). En particulier un système est cryptanalysé si on met en évidence une attaque qui a une complexité inférieure au niveau de sécurité annoncé pour le système. Si un niveau de sécurité est très élevé, une attaque est considérée comme une cryptanalyse même si en pratique on ne peut déchiffrer un message ou retrouver une clé explicitement. On appelle *attaque par force brute* une attaque où on essaie tous les paramètres d'un système (par exemple on essaie toutes les clés de chiffrement possibles). On parle d'attaque sur la clé lorsqu'on essaie de retrouver la clé.

Il peut aussi exister plusieurs types d'hypothèses pour attaquer un algorithme. Par exemple pour un algorithme de chiffrement à clé symétrique :

1. Attaque à chiffré seul on ne connaît que des chiffrés
2. Attaque à clair connu : on dispose de messages clairs et de chiffrés associés
3. Attaque à chiffré choisi : on peut choisir le message clair et obtenir un chiffré associé

En général, on attend d'un système qu'il soit résistant à un type d'attaque qui donne le plus d'information et qui peut être menée en pratique : l'attaque à chiffrés choisis. Lorsque l'on choisit des tailles de paramètres, on choisit des valeurs qui ne soient pas sensibles aux attaques par force brute.

Seuil de sécurité et loi de Moore. On évalue la sécurité d'un système en fonction du nombre d'opérations élémentaires (une somme de 2 bits par exemple) nécessaires *a priori* pour cryptanalyser (ou casser) le système par les meilleures attaques connues. On dit qu'un système est cryptanalysé si on met en évidence une attaque qui permet de faire mieux que la meilleure attaque connue sur un système. On parle de 'cassage total' d'un système, si on est capable de retrouver la clé cryptanalyser un système même pour des paramètres plus importants (par exemple en trouvant une attaque polynomiale). Plus généralement on dit qu'un système est cryptanalysé. Dans le cas où l'on choisit un seuil de sécurité on choisit ce nombre doit toujours se situer au delà de ce qu'il est théoriquement possible de faire. Par exemple si on considère 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 10000 ordinateurs en réseaux à une fréquence de 2 GHz pendant une année, soit $10000 \cdot 64 \cdot 2 \cdot 10^9 \cdot 3600 \cdot 24 \cdot 365 \approx 2^{75}$ 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 40 ans) affirme que la vitesse des ordinateurs double tous les 18 mois, ce qui donne un rapport d'à peu près 100 tous les 10 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} .

Dans ce chapitre, après avoir fait un rapide historique de la cryptographie, nous rappelons les bases d'arithmétique nécessaires à la compréhension des algorithmes. Nous présentons ensuite différents protocoles cryptographiques pour le chiffrement à clé publique, notamment la signature et l'authentification basés sur la théorie des nombres. Nous présenterons enfin des systèmes de cryptographie alternatifs comme les systèmes basés sur les codes correcteurs d'erreurs présentés au chapitre précédent.

I. RAPIDE HISTORIQUE DE LA CRYPTOGRAPHIE

Premiers systèmes. Les méthodes historiques de la cryptographie commencent par la scytale grecque au 4-ième siècle avant JC où, pour communiquer de manière chiffrée, deux individus avaient en commun la 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.



Une scytale

Chiffrement de César. César pour communiquer avec ses généraux partait d'un texte en latin puis décalait toutes les lettres de 3 : $A \rightarrow D, B \rightarrow E, C \rightarrow F, \dots, Z \rightarrow C$. Par exemple le message VENI VIDI VICI devient YHQL YLGL YLFL. Pour déchiffrer il suffit de décaler les lettres du message de trois mais dans l'autre sens. A nouveau la notion de secret partagé ici 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 (où l'on permute les lettres) et par substitution (où l'on remplace chaque lettre par une autre) forment la base théorique des systèmes de chiffrement. La notion de substitution *engendre* de la confusion pour retrouver le message et la

transposition *diffuse* cette confusion.

Substitutions monoalphabétiques On parle de chiffrement monoalphabétique lorsque une lettre est toujours remplacée une même lettre.

Le système de chiffrement de César peut être vu comme comme un chiffrement décalé :

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

ce chiffrement peut être généralisé en un chiffrement affine de la forme :

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

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

Test 1.1.

Attaque à clair connu.

Montrer que si on connaît un message d'au moins 2 lettres et son chiffré par substitution mono-

alphabétique affine, alors on peut retrouver les paramètres a et b .



le manuscrit de Voynich

Dans ce cas l'ensemble de la taille 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, soit un total de $13 \cdot 26 = 338$ manière différente de chiffrer. On peut donc de manière raisonnable tester toutes les clés possibles et voir celles qui amènent à un texte compréhensible, on parle alors d'attaque par force brute car on essaie toutes les clés de chiffrement possibles. Cette exemple montre la nécessité que l'ensemble des clés soit important.

Le cas du chiffrement affine est un cas particulier de substitution monoalphabétique où la méthode de substitution est simple. On peut imaginer que la clé secrète soit la liste complète des substitution ; il suffit de connaître la liste des lettres mais dans un autre ordre. Par exemple, si on fixe HURFABOPL...K cela signifiera que $A \rightarrow H, B \rightarrow U, C \rightarrow R, \dots, Z \rightarrow K$. 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 par contre attaquer par la fréquence des lettres de l'alphabet. En effet dans chaque si 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 les plus fréquentes sur un texte très sont en moyenne :

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%...

On peut utiliser la remarque précédente pour donner une attaque valide sur un texte long chiffré de manière monoalphabétique . Si on a un texte long chiffré la fréquence de chaque lettre même si elle a été substituée sera la même, en menant une étude des fréquences statistiques à partir du texte chiffré, si celui ci est long on peut donc retrouver des fréquences et essayer de les faire coller 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 mais pas suffisante.

Test 1.2.

Déchiffrer le texte suivant : ABT BRA UOPM

HGP FONXQPA UOBWMOB chiffré par substitution monoalphabétique.

Chiffrement de Vigenère . Le chiffrement de Vigenère est une variation sur le chiffrement monoalphabétique classique. Introduit au 16e siècle, il procède à partir d’une clé secrète commune comme une phrase qui va être placée juste en dessous du message à chiffrer et on va décaler les lettres pour les chiffrer en fonction de la valeur de la lettre associée de la clé $A \rightarrow 0, B \rightarrow 1, \dots, Z \rightarrow 25$.

Par exemple prenons la clé **CRYPTO** et le texte : ‘CHIFFREMENT’, on écrit :

CHIFFREMENTS
CRYPTOCRYPTO
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 et dans un deuxième temps, on procède à nouveau par analyse statistique, une fois qu’on a la longueur de la clé on sait que si 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 simple. Cette cryptanalyse nécessite par contre un grand texte chiffré.

Chiffrement produit . Pendant la première guerre mondiale les allemands utilisaient un chiffrement produit en ce sens qu’il alternait à la fois des substitutions mêlées à des transpositions. Ce chiffrement le chiffrement ADFGX procédait de la manière suivante, on considère le carré :



Vigenère

	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 lu comme un élément du tableau ayant une coordonnée verticale et horizontale en A, D, F, G, 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 reordonne alors les colonnes en fonction de l’ordre alphabétique des lettres du mot clé et on lit le message chiffré en lignes. Par exemple avec la clé « NICE » on écrit :

NICE		CEIN
CDAG		AGDC
XDFX	qui devient	FXDX
FAAF		AFAF
XX		XX

et le chiffré est « AGDCFXDXAFAFX ».

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

Chiffrement par blocs . Dans les exemples que nous avons vus jusqu’à présent essentiellement les lettres sont 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 lettre, mais à partir de blocs de lettre. Le système est très facile à casser mais il a été le premier système à utiliser une structure de bloc 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 DES ou l’AES sont des chiffrement par blocs et l’algèbre est présente à nouveau la plupart des systèmes.

Le système marche de la manière suivante, on considère des blocs de 3 lettres $m = (x, y, z)$ considérées comme des entiers modulo 26, on considère de plus une matrice M 3×3 à coefficients dans \mathbb{Z} telle que le déterminant de M soit premier avec 26. Le chiffré c est le résultat du produit $c = mM$. Le fait que M ait un déterminant premier avec 26 assure qu’il existe une matrice 3×3 N à valeur dans \mathbb{Q} telle que $MN = I$ la matrice identité. 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

on connaît quelques clairs (différent du vecteur nul) et leurs chiffrés on peut retrouver M , N et donc casser le système.

Chiffrement de Vernam. En 1918, Vernam a introduit un système qui est encore très utilisé aujourd’hui. Le principe est le suivant : on suppose qu’on peut 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’ils peuvent synchroniser ; par exemple, historiquement, ces séquences étaient prédéfinies au préalable en fonction d’un jour donné.

Le chiffrement s’effectue bit à bit. Considérons les bits du message m_i , les bits du message chiffré c seront $c_i = m_i \oplus s_i$ où la somme s’effectue modulo 2 (ou XOR de 2 bits). Pour déchiffrer, on 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’aléa « 10101110 » on obtient :

m	01001101	et pour le déchiffrement	c	11100011
s	10101110		s	10101110
c	11100011		m	01001101

A première vue le système paraît intrigant puisque en moyenne un bit sur deux n’est pas modifié, et donc on fait le parallèle avec un chiffrement qui ne modifierait qu’une lettre sur deux dans texte, mais la situation est différente puisqu’ici il n’existe que deux possibilités pour un bit, alors qu’il existe 26 lettres possibles. Le fait de ne pas savoir quels bits ne sont pas modifiés rend le système incassable théoriquement. En fait dans la mesure où n’importe quelle suite de bits de chiffré peut correspondre à n’importe quel clair possible lorsqu’on suppose que toutes les clés (les suites de bits aléatoires) sont possible, on peut montrer que ce chiffrement est cryptographiquement sur (c’est même le seul prouvé sur théoriquement). En pratique la difficulté va être de construire un générateur d’aléa non prévisible et la synchronisation. On reviendra plus en détail sur la manière de construire des générateurs aléatoires dans la section sur le chiffrement à flot.

Chiffrement mécanique. La fin de la première guerre mondiale a vu le début de la mécanisation du chiffrement où 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 chiffrement 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. RAPPELS D’ARITHMÉTIQUES

Dans cette section, nous rappelons les résultats essentiels en arithmétique et théorie des nombres utilisés par la suite. On trouvera les preuves et un cours détaillé dans le volume L2 (chapitre 4) ou dans le volume L3 Algèbre.

II.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 1.1. *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 $au = 1$ 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, ceci 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 étendu¹ qui calcule $d = \text{pgcd}(a, N)$ et des entiers u et v tels que $au + Nv = d$. Quand $d = 1$, l’inverse de a dans $\mathbb{Z}/N\mathbb{Z}$ est v modulo N .

Test 1.4.

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 sont inversibles à part 0.

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

1. voir les chapitres 4 et 5 du volume L2, ou les chapitres 6 et 7 de ce volume

Théorème 1.2. Théorème des restes chinois Soient n_1, n_2, \dots, n_r , r entiers deux à deux premiers entre eux, soit le produit $n = n_1 \cdot n_2 \cdot \dots \cdot n_r$ et soit x_1, \dots, x_r une suite d'entiers alors il existe un unique entier $x \pmod{n}$ tel que :

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

PREUVE. Ce théorème peut se montrer par récurrence sur r . Pour $r = 2$, supposons que $\text{pgcd}(n_1, n_2) = 1$. Il existe alors u et v tels que $un_1 + vn_2 = 1$. Posons $x \stackrel{\text{d'ef}}{=} x_1un_1 + x_2vn_2$; alors $x \equiv x_1un_1 \pmod{n_2}$. Mais, puisqu'on a aussi $un_1 \equiv 1 \pmod{n_2}$, on a bien $x \equiv x_1 \pmod{n_2}$. 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 $x - y$; 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 on peut donc appliquer la méthode précédente et l'hypothèse de récurrence. ■

Test 1.5.

Soit $n_1 = 7$ 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. d'un point de vue des groupes, le théorème des restes chinois s'interprète aussi comme le fait que $\mathbb{Z}/n\mathbb{Z} \simeq \mathbb{Z}/n_1\mathbb{Z} \times \dots \times \mathbb{Z}/n_r\mathbb{Z}$.

Dans les protocoles cryptographiques on va surtout être intéressé par l'ensemble des éléments de $\mathbb{Z}/N\mathbb{Z}$ qui admettent un inverse.

Définition 1.3. 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 le groupe multiplicatif de $\mathbb{Z}/N\mathbb{Z}$.

Définition 1.4. 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 $m \in \{1, \dots, p - 1\}$ 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 , $p, 2p, \dots, p^2, \dots, p^{r-1}(p - 1), p^r$, soit $\phi(p^r) = (p - 1)p^{r-1}$. Pour le cas où p et q sont deux entiers premiers entre eux, les nombres qui sont non premiers avec $N = pq$ 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 , $\phi(pq) = pq - \text{entiers non premiers avec } pq = pq - p - q + 1$ (le '+1' est pour pq qui est compté deux fois sinon), soit $\phi(pq) = (p - 1)(q - 1)$. On en déduit alors :

Proposition 1.5. Soit N un entier positif avec décomposition $N = \prod_{i=1}^r p_i^{r_i}$ pour p_i des nombres premiers, alors :

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

| **EXEMPLE 1.6.** $\phi(72) = \phi(8)\phi(9) = 4 \cdot 6 = 24$ $\phi(15) = \phi(3)\phi(5) = 2 \cdot 4 = 8$.

Test 1.6.

Calculer $\phi(112), \phi(84)$.

On s'intéresse maintenant pour un élément a de $\phi(N)$ à $a^{\phi(N)}$. On a les deux théorèmes fondamentaux suivants :

Théorème 1.7. Petit Théorème de Fermat Soit p un nombre premier et a un entier,

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

si et seulement si p ne divise pas a .

Théorème 1.8. Théorème d'Euler Soit N un entier alors si $a \in (\mathbb{Z}/N\mathbb{Z})^*$,

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

Le petit Théorème de Fermat est un cas particulier du Théorème d'Euler qui est lui même un cas particulier du Théorème de Lagrange appliqué au groupe $(\mathbb{Z}/N\mathbb{Z})^*$:

Théorème 1.9. Théorème de Lagrange Si G est un groupe multiplicatif unitaire, alors pour tout $a \in G$:

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

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

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

Définition 1.10. On dit que a est un générateur de $(\mathbb{Z}/N\mathbb{Z})^*$ quand $\text{Gen}(a) = (\mathbb{Z}/N\mathbb{Z})^*$. On parle parfois de générateur multiplicatif de $(\mathbb{Z}/N\mathbb{Z})^*$.

Notons que, selon la valeur de N , il n'existe pas toujours de générateurs 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 :

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

EXEMPLE 1.11. 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 1.7.

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 on prend $N = 15$, $\phi(15) = 8$, et $(\mathbb{Z}/15\mathbb{Z})^* = \{1, 2, 4, 7, 8, 11, 13, 14 \pmod{15}\}$. On observe que $\text{Gen}(2) = \{2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8 \pmod{15}\}$, $\text{Gen}(13) = \{13^0 = 1, 13^1 = 13, 13^2 = 4, 13^3 = 7 \pmod{15}\}$, $\text{Gen}(11) = \{11^0 =$

$1, 11^1 = 11 \pmod{15}$ et $\text{Gen}(14) = \{14^0 = 1, 14 \pmod{15}\}$; 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 $(\mathbb{Z}/6\mathbb{Z})^* = \{1, 5 \pmod{6}\}$, 5 est un générateur.

Test 1.8.

1. Trouver un générateur de $(\mathbb{Z}/11\mathbb{Z})^*$.
2. Existe-t-il un générateur de $(\mathbb{Z}/18\mathbb{Z})^*$?

III. CHIFFREMENT À CLÉ SECRÈTE

Il existe deux grands types de chiffrement à clé secrète (ou symétrique) utilisés aujourd'hui. Le chiffrement par blocs dont le plus connu est le chiffrement standard DES (Data Encryption Standard), qui aujourd'hui est remplacé par le nouveau standard AES (Advanced Encryption Standard). Ce type de chiffrement est considéré sur le principe comme étant relativement sûr (même si des chiffrements particuliers peuvent ne pas l'être). Ce type de chiffrement a comme limitation relative d'être assez lent structurellement puisqu'il demande un certain nombre de répétitions d'étapes élémentaires. Le chiffrement à flot (ou masque jetable ou chiffrement de Vernam), se base sur la notion de générateur pseudo-aléatoire synchronisé pour le chiffrement et le déchiffrement. Ce chiffrement procède bit à bit et est en général à la fois plus simple à implémenter et en général plus rapide que le chiffrement par bloc (en général on recherche un facteur de rapidité de 10, sachant que ce chiffre est à relativiser en fonction du type d'implémentation (hardware ou software) et de biens d'autres facteurs...). Ce type chiffrement est par exemple utilisé dans le téléphone portable (chiffrement A5/1) ou dans la norme Bluetooth (chiffrement E₀). Par contre ce chiffrement est considéré comme relativement moins sûr que le chiffrement par blocs, mais la nécessité de chiffrement à la fois rapide et simple en fait un secteur de recherche très actif même si les points de vue peuvent être partagé sur la sécurité pratique de tels protocoles. Dans cette section on présente les bases des chiffrements par blocs et à flot. Dans un complément on présente plus en détails les chiffrements par blocs DES et AES.

III.1. Chiffrement par blocs

III.1.1. Définitions générales

Définition 1.12. Soit n, l des entiers et $k \in \mathbb{F}_2^l$. Un chiffrement par bloc est une fonction $E : \mathbb{F}_2^n \times k \rightarrow \mathbb{F}_2^n$ tel que pour tout $k \in \mathbb{F}_2^l$, la fonction $E_k() : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ est inversible, pour tout k , où k est la clé du chiffrement. La fonction inverse de $E_k()$ est notée $D_k()$. L'entier n est la taille des blocs.

Définition 1.13. Chiffrement itératif par blocs

On rappelle qu'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. Un chiffrement itératif par blocs est un chiffrement résultant de l'opération itérée d'une fonction F à chaque bloc de clair, F étant un chiffrement produit. Chaque application de F est appelé "tour". Un tel chiffrement est caractérisé par : son nombre de tour, la taille des blocs, la taille de la clé. On peut aussi utiliser pour chaque tour des sous-clés engendrées à partir de la clé de chiffrement initiale.

Pour rendre le chiffrement inversible, F doit être inversible. Nous allons expliquer les deux type de schémas référants pour le chiffrement par blocs : le schéma de Feistel (sur lequel est basé le D.E.S) et le schéma de chiffrement substitutions/permutations de Shannon.

III.1.2. Schéma de Feistel

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

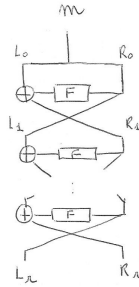


FIG. 1.1. Schéma de Feistel

Définition 1.14. Un schéma de Feistel est un chiffrement itératif associant à un bloc de clair de $2t$ bits (L_0, R_0) , un bloc chiffré (R_r, L_r) de taille $2t$ par un procédé à r tours, $r \leq 1$. Plus précisément à chaque tour i le schéma transforme (L_{i-1}, R_{i-1}) en (L_i, R_i) à l'aide d'une sous-clé K_i dérivée de la clé K utilisée pour le chiffrement, par : $L_i = R_{i-1}, R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$ où f est un chiffrement produit de $\mathbb{F}_2^t \rightarrow \mathbb{F}_2^t$.

Pour faire le lien par rapport à la définition de la fonction F d'un chiffrement par bloc, dans le cas d'un schéma de Feistel cette fonction est donnée par la fonction qui associe le couple (L_i, R_i) au couple (L_{i+1}, R_{i+1}) .

Proposition 1.15. 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 qu'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_r, R_r) on est capable de retrouver tour après tour les différents (L_i, R_i) jusqu'au clair (L_0, R_0) . On peut remarquer qu'on peut réutiliser le circuit utilisé pour le chiffement mais en partant de (R_r, L_r) au lieu de (L_0, R_0) et en inversant l'ordre des sous-clé K_i . ■

EXEMPLE 1.16. Considérons un exemple simple (et complètement irréaliste) où $2t = 8$ avec pour $f(x, k)$ la fonction qui commence par faire une rotation de x d'une position vers la droite, puis additionne bit à bit la clé de tour K_i et enfin applique l'opération $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ définie par $S(x_1, x_2, x_3, x_4) = (x_1 x_2 \oplus x_3, x_1 x_2 x_3 \oplus x_4, x_2 x_3 \oplus 1, x_1 x_3 x_4 \oplus x_2)$. On a schématiquement : $f_{K_i} : x \xrightarrow{\text{Rot}} \xrightarrow{\oplus K_i} \xrightarrow{S} f(x, K_i)$. De plus on considère l'algorithme de dérivation des clés suivant : à partir de la clé de chiffement $K = K_0$ 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 chiffement 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 de une position vers la droite, soit (1100) , on ajoute K_1 pour obtenir (1011) puis on applique la fonction S et on obtient $(1.0 \oplus 1, 1.0.1 \oplus 1, 0.1 + 1, 1.1.1 \oplus 0) = (1, 1, 1, 1)$. Soit $f(R_0, K_1) = (1111)$, on calcule $R_1 = L_0 \oplus f(R_0, K_1) = (1010) \oplus (1111) = (0101)$, puis $L_1 = R_0 = (1001)$. A 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 1.9.

Déchiffrer le chiffré de l'exemple avec la clé de chiffrement. **Test 1.10.**

Montrer que dans un schéma de Feistel si f est composé uniquement d'opérations linéaires et de l'addition d'une sous-clé de tour, 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 .

Le cas du DES Le DES est un algorithme basé sur le schéma de Feistel. Historiquement le bureau national des standards américains (NBS qui allait devenir le National Institute of Standards and Technologies (N.I.S.T)) lancé un appel à proposition pour un standard de schéma de chiffrement. IBM a candidaté en 1974 avec un système basé sur un schéma de Feistel appelé LUCIFER. Ce système de chiffrement a été envoyé pour analyse à la NSA qui l'a modifié légèrement en 1975, c'est ce schéma qui a été standardisé en 1977 sous le nom de DES comme libre de droits.

Le DES 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. Le chiffrement devait être réévalué tous les 5 ans.

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 DES 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 DES) qui sont chacune 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 one ou deux positions à chaque tour à partir de la clé de chiffrement.

Le Triple DES Pour pallier la faiblesse relative du DES due à la taille de la clé, qui a été très vite pointée du doigt, il a été proposé une variation utilisant le DES comme brique de base : le triple DES ou TDES. Le triple DES utilise 2 clés DES au lieu d'une, et se définit par $TDES_{K_1, K_2}(m) = DES_{K_1}(DES_{K_2}^{-1}(DES_{K_3}(m)))$, le déchiffrement se faisant par $TDES_{K_1, K_2}^{-1}(c) = DES_{K_1}^{-1}(DES_{K_2}(DES_{K_1}^{-1}(C)))$. La taille de la clé passe alors de 56 bits à 112 bits. Le triple DES a été standardisé en **PG XXX 1986 ???** et est encore très utilisé de nos jours même s'il tend à se faire remplacer par le nouveau standard AES dont on va parler par la suite.

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

$$M \xrightarrow{DES_{K_1}} R \xrightarrow{DES_{K_2}} C$$

L'attaque est une attaque à clair connu où on suppose qu'on connaît quelques couples clair/chiffré pour la même clé $(K_1, K_2) : (M_i, C_i)$. 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 $DES_{K_1}(M_1)$. 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 $DES_{K_1}(M_1)$ 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 $DES_{K_2}^{-1}(C_1)$, pour chaque clé K'_2 on va alors trouver un $R_1 = DES_{K'_2}^{-1}(C_1)$ auquel on va faire correspondre une clé K'_1 telle que $DES_{K'_1}(M_1) = R_1$:

$$M_1 \xrightarrow{DES_{K'_1}} R_1 \xleftarrow{DES_{K'_2}^{-1}} C_1$$

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 : $DES_{K_1}(DES_{K_2}(M_1)) = C_1$. 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 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 on peut alors tester tous les couples de clés obtenus qui 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 $56 * 2^{56}$ (où 56 est le log en base 2 de 2^{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 est mémoire est aussi de l'ordre de $2^{56} * 56$ (à un petit facteur près) le coût de stockage des listes. 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*. Ce donne une sécurité pour le double DES trop faible.

Test 1.11.

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

Test 1.12.

On considère le cas du $DES_{K_1, K_2}(M) = K_1 \oplus DES_{K_2}(M)$ et $DES_{K_1, K_2}(M) = DES_{K_1}(M \oplus K_2)$, montrer qu'on peut mener une attaque par le milieu pour ces deux systèmes. Est-ce que le

$DES_{K_1, K_2, K_3}(M) = K_1 \oplus DES_{K_2}(M \oplus K_3)$ est sensible à cette attaque?

Test 1.13.

On considère le schéma utilisant 3 DES mais 2 clés k_1 et k_2 : $DES_{k_1}(DES_{k_2}(DES_{k_2}(M)))$. (L'ordre des clés a été modifié par rapport au TDES classique). Montrer comment on peut attaquer ce schéma avec une attque de type meet-in-the-middle.

III.1.3. 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.

SCHEMA S/N de SHANNON????

EXEMPLE 1.17. On considère un schéma à deux tours très simple à titre d'exemple sur \mathbb{F}_2^4 où un tour est composé de l'action d'une permutation P qui est une rotation d'une position vers la droite. Ainsi que d'une application inversible $F(x, K_i)$ (qui assure la confusion) qui applique d'abord la transformation inverse $x \rightarrow x^{-1}$, pour $x = (x_1, x_2, x_3, x_4)$ considéré comme un élément de \mathbb{F}_{16} , $x_1 + x_2X + x_3X^2 + x_4X^3$, et où \mathbb{F}_{16} est construit comme $\mathbb{F}_2[X]/(X^4 + X + 1)$, avec l'inverse de $(0000) = (0000)$, puis additionne la clé de tour K_i . On considère à nouveau que la clé de tour K_i est construite à partir d'une rotation vers la gauche de une position de K_{i-1} avec $K_0 = K$ la clé de chiffrement.

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

Pour le déchiffrement on considère les opérations inverses en additionnant la clé K_2 puis en calculant l'inverse, 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 1.14.

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

Le cas de l'AES En 1996, le N.I.S.T a lancé une compétition pour remplacer le DES, 12 algorithmes ont été proposés, une première sélection en a retenu 5, puis l'algorithme Rijndael proposé par John Daemen et Vincent Rijmen a été sélectionné et est devenu en 2000 le nouveau standard AES. 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'AES-128, AES-192 ou AES-256. Un tour d'AES est constitué de 4 transformations élémentaires : **SubBytes**, **ShiftRows**, **MixColumns** et **AddRoundKey**. La structure de l'AES est basé sur des octets considérés comme des éléments du \mathbb{F}_{256} . Un chiffré intermédiaire de 128 bits est considéré comme une matrice 4×4 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 AddRoundKey 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.

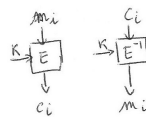
III.1.4. Sécurité du chiffrement par blocs

Toute la problématique lorsqu'on conçoit un système de chiffrement par blocs est de trouver des fonctions F ou f (pour le schéma de Feistel) qui soient à la fois sûres mais aussi rapides. Un algorithme doit être résistant aux attaques connues (comme par exemple les attaques classiques non développées ici que sont la cryptanalyse différentielle ou la cryptanalyse linéaire) mais il est très difficile d'avoir des preuves sur la sécurité des systèmes. Au jour d'aujourd'hui AES-256 est considéré comme sûr.

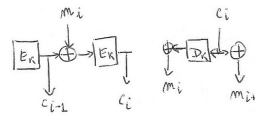
III.1.5. Modes opératoires

Pour le chiffrement, un message est bien sur 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, \dots, 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 des chiffrés c_i .



Le mode ECB



Le mode CBC

► **Mode ECB** Le mode ECB est le mode le plus simple où chaque bloc m_i est chiffré indépendamment. On a pour $1 \leq i \leq t$, $c_i = E_k(m_i)$ 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 1 bit d'un bloc est modifié pour une raison quelconque, au niveau du déchiffrement seul un bloc sera modifié. Par contre un bloc donné clair 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.

► **Mode CBC** 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 $1 \leq i \leq t$ on pose $c_i = E_k(m_i \oplus c_{i-1})$. Pour le déchiffrement : $c_0 = VI$, $m_i = c_{i-1} \oplus D_k(c_i)$. Ce mode a l'avantage que le chiffrement d'un bloc m_i va dépendre de ce qui le précède. Cela rend plus difficile des attaques par dictionnaire où on essaierait de reconnaître des bouts de texte, par contre en cas d'erreur de transmission sur le chiffré il y aura une très forte propagation d'erreur pour le déchiffrement.

► **Mode OFB** Ce mode permet d'utiliser un schéma de chiffrement par bloc 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 $z_0 = IV$ qui doit changer pour chaque message à chiffrer. En particulier comme tout chiffrement par bloc peut être transformé en chiffrement à flot de cette manière, les algorithmes de chiffrement à flot pour avoir un intérêt doivent aller plus vite que l'AES.

► **Mode CFB** Dans ce cas c'est le chiffré qui va engendrer l'aléa : $c_0 = IV$, $z_i = E_k(c_{i-1})$ pour $c_i = m_i \oplus z_i$. Pour le déchiffrement : $c_0 = IV$, $m_i = c_i \oplus E_k(c_{i-1})$.

III.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 on appelle s_i la séquence pseudo-aléatoire, m_i les bits du messages et c_i les bits chiffrés associés on a pour le chiffrement $c_i = s_i \oplus m_i$ et pour le déchiffrement $m_i = c_i \oplus s_i = m_i \oplus s_i \oplus s_i$.

L'intérêt du chiffrement à flot par rapport au chiffrement par blocs est qu'il demande en général moins de ressources pour être implémenté puisqu'on a juste besoin d'un générateur pseudo-aléatoire quand le chiffrement par bloc nécessite de faire un nombre de tours importants. Cela en fait un chiffrement intéressant pour des environnements contraints comme le téléphone portable où un chiffrement de ce type (le chiffrement A5/1) est utilisé pour la norme GSM.

En général le générateur est connu, seule l'initialisation du générateur est inconnue. L'initialisation est le secret partagé par les deux partis et est la clé secrète du système. L'intérêt du chiffrement à flot est qu'il repose sur un générateur pseudo-aléatoire

Toute la difficulté du chiffrement à flot consiste à trouver des générateurs pseudo-aléatoires qui soient à la fois sûr et rapide. On a vu dans la section précédente comment utiliser un algorithme de chiffrement par blocs pour faire du chiffrement à flot (mode OFB et CFB). Pour être intéressant un générateur pseudo-aléatoire doit donc être au moins plus rapide ou plus léger à implémenter que les générateurs dérivant des chiffrements par blocs.

La famille des registres à décalage linéaire est une exemple de famille avec de bonne propriétés statistiques et très rapide. Elle ne peut cependant être utilisée telle quelle car on peut l'attaquer facilement.

III.2.1. Registres à décalage linéaires

Définition 1.18. Un registre à décalage de longueur L est constitué de L cases mémoires numérotées de 0 à $L-1$, chacune pouvant contenir 0 ou 1 (1 bit) et ayant une sortie et une entrée, ainsi que d'une horloge controlant le mouvement des données. A 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 $1 \leq i \leq L-1$.
- la case $L-1$ est remplacé par le bit de rétroaction

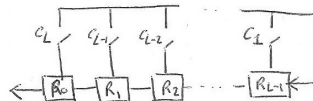


FIG. 1.2. Registre à décalage (L.F.S.R)

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

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

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

EXEMPLE 1.19. 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 $s_3 = s_2 a_1 \oplus s_1 a_2 \oplus s_0 a_3 = 1 \oplus 0 \oplus 0 = 1$, on fait alors sortir le premier bit du registre $s_0 = 0$, puis on décale les 2 derniers bits du registre et on remplace la dernière case du registre par le bit de rétroaction calculé $s_3 = 1$. 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 14 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 .

Lemme 1.20. Soit R un registre à rétroaction linéaire de longueur L . Alors si $C(x)$ est de degré L , la séquence de sortie est périodique de période au plus $2^L - 1$.

PREUVE. On remarque tout d'abord que l'initialisation nulle amène une séquence identiquement nulle. Le registre a L états pouvant valoir 0 ou 1, soit $2^L - 1$ états distincts non nuls possibles au maximum. Si $C(x)$ a 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 de $C(x)$, c'est ce que nous faisons dans le théorème suivant qu'on admettra :

Théorème 1.21. *Soit R un registre de longueur L avec polynôme de rétroaction linéaire $C(x)$ de degré L , alors :*

1. *si $C(x)$ est irréductible sur $\mathbb{F}_2[x]$ alors chacun des $2^L - 1$ états non nuls possible produit une séquence de sortie 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 (ie : tel que $C(x)$ est irréductible et x engendre $\mathbb{F}_{2^L}^*$ vu comme $\mathbb{F}_{2^L} = \mathbb{F}_2[x]/(C(x))$) alors R engendre une séquence de période maximale $2^L - 1$ pour toute initialisation non nulle.*

Test 1.15.

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 pé-

riode ?

Test 1.16.

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 mémoires. Ils sont très simples à implémenter sur des circuits, ils ont de plus de très bonnes propriétés statistiques, mais ne sont pas utilisables tels quels en cryptographie.

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

Pour un générateur à rétroaction 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 telle suite chiffrante soit utilisée pour chiffrer un message, et qu'on connaisse à la fois une séquence suffisamment grande (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 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 inconnus (les a_i) et L équations données par :

$$s_{n+j} = a_1 s_{n+j-1} \oplus a_2 s_{n+j-2} \oplus \dots \oplus a_L s_{n+j-L}$$

pour $L + 1 \leq j \leq 2L$. Une fois qu'on a obtenu $C(x)$ on peut alors engendrer toute la suite chiffrante, avant et après les bits connus et déchiffrer tous le message. 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 la.

Test 1.17.

On intercepte un message chiffré avec un système de chiffrement à flot produit par une suite chiffrante récurrente de type LFSR. Le message

binaire intercepté est : 0000011011110010 On sait d'autrepart 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.

III.2.3. Variations sur les registres à décalage linéaire

Comme il n'est pas possible d'utiliser directement des registres linéaires pour chiffrer mais que ceux ci ont à la fois de bonnes propriétés de périodicité et sont très faciles à implémenter, on cherche à essayer de les utiliser mais pas directement. Plusieurs solutions sont possibles pour les utiliser en tant que brique de base d'une variation. En particulier il est nécessaire d'introduire une part de non-linéarité par contre les attaques linéaires élémentaires, tout en gardant le caractère rapide de cette construction. De nombreuses variations ont été proposées depuis des années

Nous proposons par la suite 2 variations bien connues (parmi de nombreuses).

► **Registres combinés**

L'idée dans ce cas est d'utiliser plusieurs registres R_1, R_2, \dots, R_n différents produisant chacun à chaque top d'horloge un bit de sortie x_i . Les bits de sortie sont alors utilisés avec une fonction booléenne $f(x_1, \dots, x_n) = z$ de $\mathbb{F}_2^n \rightarrow \mathbb{F}_2$.

EXEMPLE 1.22. Le générateur de Geffe a été utilisé dans les années '70 avec trois registres et une fonction $f(x_1, x_2, x_3) = x_1 x_2 \oplus x_2 x_3 \oplus x_3$.

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 du temps.

Attaque par corrélation Ce type de schéma peut être attaqué par des attaques à clair connu (c'est à dire qu'on connaît un bon de suite chiffrante) où 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). Par exemple dans le cas du générateur de Geffe, on a 3 registres puis on applique $f(x_1, x_2, x_3) = x_1 x_2 \oplus x_2 x_3 \oplus x_3$. 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 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.

L'attaque consiste alors a retrouver la valeur du registre R_3 lié à x_3 . Si le registre L_3 a longueur l on essaie toutes les valeurs possibles du registre L_3 (s'il n'est pas trop grand) et pour chacune de ses valeurs du registre on engendre (puisque'on considère connu le polynôme de retroaction) une suite de valeurs de x_3 de longueur m suffisamment grand.. 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 par contre on ne teste pas les bonnes valeurs de L_3 , il n'y a pas de biais a priori et 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.

► **Registres filtrés** Dans ce cas on considère un générateur linéaire d'une longueur donnée (comme $L=128$ ou 256 cases mémoires) et on choisit n cases mémoires 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 $z = f(x_1, \dots, x_n)$.

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 Len Adleman (mais publié en 1978) et a été popularisé sous le nom de 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 on a les deux clés ; mais il doit être difficile de déchiffrer le message si 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 à lettre 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.

IV.1. Algorithme R.S.A

Méthode

Chiffrement R.S.A

► Génération des clés

Soient p et q deux nombres premiers et $N = pq$. Alice choisit $e \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $1 \leq e \leq N - 1$ 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) .

► 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.

► 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

$$c^d \equiv m^{ed} \equiv m^{1+k\phi(N)} \equiv m \cdot (m^{\phi(N)})^k \pmod{N}.$$

Or, d’après le théorème d’Euler, $m^{\phi(N)} \equiv 1 \pmod{\phi(N)}$ donc $c^d \equiv m \pmod{\phi(N)}$ et Alice retrouve bien le message envoyé par Bob. ■

EXEMPLE 1.23. Soit $p = 7, q = 11$ 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 oeuvre des calculs.

IV.2. Mise en oeuvre des calculs

Si l’on s’y prend mal, les calculs sont infaisables. Dans l’exemple ci-dessus, on a

$$58^{37} = 176537008025531257672933899600492053742894001443250331170423963648.$$

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 binaire², vue au volume L2 :

Rappel

Exponentiation binaire

Pour un élément g d’un groupe multiplicatif, on veut calculer g^n .
 Décomposons n en base 2 : $n = n_0 + 2n_1 + 4n_2 + \dots + 2^k n_k$ où les $n_i \in \{0, 1\}$.
 Ainsi, g^n peut se réécrire

$$\begin{aligned} g^n &= g^{n_0} g^{2n_1} g^{4n_2} \dots g^{2^k n_k} \\ &= g^{n_0} (g^2)^{n_1} (g^4)^{n_2} \dots (g^{2^k})^{n_k} \end{aligned}$$

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 on compte qu’une multiplication de deux nombres de

2. on trouve aussi ce procédé sous une multitude d’autres noms dans la littérature : « exponentiation modulaire », « exponentiation rapide », « square and multiply », « repeated squaring », etc.

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 ou autres améliorations (voir le chapitre 5 du volume L2). D'autres améliorations (basées sur les méthodes modulaires et le théorème des restes chinois) sont proposées en exercice.

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 arithmétique pour le chiffrement. Pour une machine cadencée à 1 GigaHertz et une architecture 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 qu'on dispose de processeurs optimisés pour ces opérations (des *cryptoprocresseurs*), auquel cas les opérations de chiffrement et déchiffrement sont significativement plus rapides.

En pratique l'opération qui coute 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 $\phi(N)$. Par contre, 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.

Clé publique ou clé secrète ? 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 message, 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

R.S.A et factorisation. Si 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 on sait calculer $\phi(N)$, alors on retrouve p et q facilement : comme $\phi(N) = (p-1).(q-1)$ et qu'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 $e^{i^e m^e}$ 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 on savait casser R.S.A, il n'est pas clair que l'on saurait pour autant factoriser N ou calculer $\phi(N)$.

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 appelée « crible algébrique » (nous ne l'expliquerons pas ici) et a une complexité en $L_{\frac{1}{3}}(N) \stackrel{\text{déf}}{=} e^{\ln(N) \frac{1}{3} \ln(\ln(N)) \frac{2}{3}}$. 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). A 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 suffi-

samment 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 d'alors 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. Attaque multiplicative

Le R.S.A a une propriété multiplicative.

1. Montrer que pour un chiffrement R.S.A, avec $c_1 = \text{Chiffr}(m_1)$ et $c_2 = \text{Chiffr}(m_2)$ qu'alors $c_1 c_2 = \text{Chiffr}(m_1 m_2)$

2. On suppose qu'on a un message chiffré c_1 et qu'on peut faire déchiffrer des messages (mais pas c_1) montrer qu'alors en faisant déchiffrer un message lié à c_1 mais différent, 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 qu'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 devient alors de 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, ceci impliquerait une probabilité raisonnable de trouver un nombre premier en faisant plusieurs dizaines d'essais « au hasard ».

Test de Fermat. Une première façon de faire consiste à utiliser une réciproque du petit théorème de Fermat, appelé *test de Fermat*. Soit n un entier et a un entier tel que $1 < a < n - 1$. Alors, si n est premier, le théorème de Fermat montre que $a^{n-1} \equiv 1 \pmod{n}$. Si on prend plusieurs a au hasard et qu'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 $a^{n-1} \equiv 1 \pmod{n}$, alors il y aurait de bonnes chances que n soit premier. Cette idée se heurte au fait qu'il existe des nombres (dit de Carmichael) tels que, pour ces nombres n , l'on ait $a^{n-1} \equiv 1 \pmod{n}$ 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.

Test de Rabin-Miller. En 1976 puis 1980, Rabin et 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 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, $a^{n-1} \equiv 1 \pmod{n}$. 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 on trouve un nombre de la forme $a^{n-1} = (a^m)^{2^t}$ dont le carré vaut 1 mais qui ne vaut ni 1 ni -1 , alors n n'est donc pas premier. Ceci 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 $0 \leq r \leq s - 1$ 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^{-2^t} .

PREUVE. 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 débiter 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 clé dans l'article de Diffie-Hellman de 1976. En 1984, El Gamal a proposé des systèmes pour l'utiliser pour le chiffrement et la signature.

V.1. Problème du logarithme discret

Problème du logarithme discret. Soit p un nombre premier, 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.2. Echange des clés de Diffie-Hellman

Pour appliquer un algorithme de clé 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 des clés de Diffie-Hellman

► **Entente préalable** Les deux partis A et B se mettent d’accord publiquement sur un entier p et un générateur g de $(\mathbb{Z}/p\mathbb{Z})^*$.

► **Protocole**

1. A et B choisissent séparément des entiers aléatoire a et b de $(\mathbb{Z}/p\mathbb{Z})^*$
2. A envoie alors $g^a \pmod{p}$ à B et B envoie $g^b \pmod{p}$ à A
3. 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 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 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 $b = 13$.

rifier 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 .

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 symétrique 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 soit difficile. Ceci permet de nombreuses variantes de Diffie-Hellman.

V.3. Chiffrement d'El Gamal

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

Méthode	<p style="text-align: center;">Chiffrement d'El Gamal</p> <p>► 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 $1 \leq x_B \leq p - 1$. Il calcule $X_B = g^{x_B} \pmod{p}$. CLÉ PUBLIQUE : X_B. CLÉ PRIVÉE : le triplet (p, g, X_B).</p> <p>► 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 $1 \leq k \leq p - 1$. Elle calcule $\gamma = g^k \pmod{p}$ et $\delta = mX_B^k \pmod{p}$. Le chiffré c est le couple $c = (\gamma, \delta)$.</p> <p>► Déchiffrement Bob reçoit c. Il calcule $m' = \gamma^{-x_B} \delta \pmod{p}$ et alors $m = m'$.</p>
----------------	--

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

Taille des paramètres. 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 le paragraphe correspondant dans la partie R.S.A).

Comparaison avec R.S.A. Le chiffré est constitué de deux nombres modulo p , là où R.S.A n'en utilise qu'un. En terme 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 néanmoins l'avantage d'être non déterministe (en ce sens que le chiffré dépend 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 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. fonction 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 $X_0 = \sum_{i \text{ pair}} x_i$ et $X_1 = \sum_{i \text{ impair}} x_i$, 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) : ce sera l’objet de la partie suivante.

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

Définition 1.26. Soit $h : X \rightarrow Y$ 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 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 » et que « à sens unique » implique « faiblement sans collision ».

Test 1.28.

Montrer qu’une fonction peut être « sans collision » mais pas « à sens unique ».

VI.2. Attaque par paradoxe des anniversaires

Soit un 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 :

$$\bar{P} = \frac{n-1}{n} \frac{n-2}{n} \dots \frac{n-(k-1)}{n} = \prod_{i=1}^k \left(1 - \frac{i}{n}\right).$$

Soit :

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

EXEMPLE 1.27. Si on considère $n = 365$ (le nombre de jour 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 assez surprenante intuitivement, 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 1.29.

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 $m \leq 60$. Soit $x = \prod_{i=1}^k (1 - \frac{i}{n})$. On a alors :

$$\ln(x) = \ln(1 - \frac{i}{n}) = \sum_{i=1}^{k-1} \ln(1 - \frac{i}{n})$$

On peut alors faire un développement limité pour $k \ll n$ qui donne :

$$\ln(x) = - \sum_{i=1}^{k-1} \frac{i}{n} + O(\frac{i^2}{n^2}) = - \frac{1}{n} \sum_{i=1}^{k-1} i + O(\frac{k^3}{n^2}) = - \frac{k^2}{2n} + O(\frac{k^3}{n^2}).$$

Soit $P \approx e^{-\frac{k^2}{2n}}$.

Nous pouvons donc en déduire que, pour $k \approx \sqrt{n}$ (qui reste très inférieur à n), P est de l'ordre de $1/\sqrt{e}$ soit une valeur relativement proche de 0.6. Ceci montre que, si on prend $k \approx \sqrt{n}$, la probabilité d'obtenir deux tirages identiques est raisonnable.

Proposition 1.28. *Si une fonction de hachage a une sortie de taille m bits, alors il existe une probabilité raisonnable 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 pour $\mathcal{O}(2^{\frac{m}{2}})$ tirages aléatoires. Il suffit de considérer $\mathcal{O}(2^{\frac{m}{2}})$ valeurs possibles à hacher, de les conserver dans une liste et de comparer toute nouvelle valeur de la fonction de hachage à celles déjà obtenues. ■

La proposition précédente implique que, pour obtenir une sécurité de 2^{80} pour les attaques en collision, il est nécessaire de prendre une sortie d'au moins 160 bits pour une fonction de hachage.

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 qu'on itère. Une fonction de compression est une fonction f de $\mathbb{F}_2^n \times \mathbb{F}_2^r \rightarrow \mathbb{F}_2^r$ qui prend en entrée deux valeurs de longueur n bits et r bits et renvoie en sortie un vecteur de r bits. On itère cette fonction pour obtenir une fonction de hachage h de la manière suivante. Soit x une suite de bits que l'on veut hacher, on écrit x comme une suite de t blocs de longueur n , x_1, \dots, x_t auxquels on ajoute un bloc x_{t+1} contenant la longueur de x (on suppose que r est suffisamment grand - typiquement au moins 64 - pour contenir la longueur du message. Si la longueur de x ne tombe pas juste on remplit le dernier bloc de x_t de x par des valeurs fixées comme une suite de '0' ou une suite de '1'. On part d'un bloc H_0 constant de longueur r puis on itère par $H_i = f(x_i, H_{i-1})$ pour $1 \leq i \leq t + 1$. Eventuellement on utilise une fonction finale g pour obtenir $h(x) = g(H_{t+2})$.

On peut alors montrer qu'une collision sur h implique une collision sur la fonction de compression f . Toute la difficulté consiste alors à trouver des fonctions de compression résistantes aux collisions.

SCHEMA GENERAL

Il existe trois grands types de fonctions : les fonctions de compression basées sur des algorithmes de chiffrement existant qu'on estime surs, 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.

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

Il existe plusieurs types de schémas. Par exemple une fonction où on utilise le bloc H_{i-1} comme clé pour chiffrer x_i

$$H_i = \text{Enc}_{g(H_{i-1})}(x_i) \oplus x_i$$

pour g une fonction qui transforme H_{i-1} en une longueur adaptée à l'algorithme de chiffrement et $\text{Enc}_k()$ un algorithme de chiffrement par blocs.

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

$$H_i = \text{Enc}_{x_i}(H_{i-1}) \oplus H_{i-1}.$$

Il existe aussi des variations où 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, par contre les contraintes pour que f soit sans collision demande une longueur de chiffré longue qui rend en pratique ce type de fonctions plus lente que les fonctions de compressions spécialisées.

VI.4. Fonctions de chiffrement spécialisées

Ron Rivest a introduit pour l'algorithme MD4 (Message Digest - 4) une structure de telles fonctions qui a été reprise par la suite pour les algorithmes MD5 et SHA-1 (Secure Hash Algorithm).

Nous présentons ici l'algorithme SHA-1, recommandé par le N.I.S.T :

L'algorithme SHA1 prend en entrée un message de longueur au plus $2^{64} - 1$ bits, la fonction de compression est une fonction $\mathbb{F}_2^{512} \times \mathbb{F}_2^{160} \rightarrow \mathbb{F}_2^{160}$, le haché a longueur 160.

Pour cet algorithme on considère 80 fonctions de $\mathbb{F}_2^{32} \times \mathbb{F}_2^{32} \times \mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$, définies par :

1. $f_t(B, C, D) = (B \wedge C) \vee (\bar{B} \wedge D)$ pour $0 \leq t \leq 19$
2. $f_t(B, C, D) = B \oplus C \oplus D$ pour $20 \leq t \leq 39$
3. $f_t(B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ pour $40 \leq t \leq 59$
4. $f_t(B, C, D) = B \oplus C \oplus D$ pour $60 \leq t \leq 79$

avec \wedge l'opération ET logique exécutée bit à bit et \vee l'opération OU logique, l'opération \bar{B} représente N le complémentaire bit à bit et \oplus le XOR (ou addition modulo 2). On utilise aussi des constantes écrites en hexadécimal définies par : $K_t = 5A827999$ pour $0 \leq t \leq 19$, $K_t = 6ED9EBA1$ pour $20 \leq t \leq 39$, $K_t = 8F1BBCDC$ pour $40 \leq t \leq 59$ et $K_t = AC62C1D6$ pour $60 \leq t \leq 79$.

On transforme le message x pour l'avoir sous une forme adéquate de la manière suivante : on ajoute 1 à la fin de x puis on ajoute des 0 pour obtenir une longueur valant $n.512 - 64$, enfin on ajoute la longueur en bits de x dans les 64 derniers bits. On écrit alors le message x comme une séquence de n blocs de 512 bits $x_1 x_2 \dots x_n$. On utilise pour initialiser H_0 la séquence de 5 blocs : $H_0 = 67452301, H_1 = \text{EFCDA}889, H_2 = 98\text{BADCFE}, H_3 = 10325476, H_4 = \text{C3D2E1F0}$. On note par la suite $\text{Rot}_k(x)$ la permutation circulaire vers la gauche de k bits pour un mot de 16 bits, '+' désigne l'addition modulo 2^{16} correspondant aux mots de 16 bits.

La fonction de compression f pour les n blocs x_i est alors définie par :

Méthode

Fonction de compression de l'algorithme SHA-1

On utilise pour SHA-1 la fonction de compression suivante pour chacun des n blocs de 512. On part de 5 mots de 32 bits H_0, H_1, H_2, H_3 et H_4 , initialisés par des constantes au départ puis modifiés par l'introduction des blocs x_i de longueur 512.

1. On écrit $x_i = y_0 y_1 \dots y_{15}$ comme 16 mots de 32 bits
2. Pour $t = 16, 17, \dots, 79$ on calcule $y_t = \text{Rot}_1(y_{t-3} \oplus y_{t-8} \oplus y_{t-14} \oplus y_{t-16})$
3. On pose $A = H_0, B = H_1, C = H_2, D = H_3$ et $E = H_4$.
4. pour $t = 0, 1, \dots, 79$ calculer $\text{Rot}_5(A) + f_t(B, C, D) + E + y_t + K_t, E = D, D = C, C = \text{Rot}_{36}(B), B = A, A = T$.
5. Calculer $H_0 = H_0 + A, H_1 = H_1 + B, H_2 = H_2 + C, H_3 = H_3 + D, H_4 = H_4 + E$

Dans ce cas il n'y a pas de fonction finale g et la valeur du haché est la séquence $H_0 H_1 H_2 H_3 H_4$ à la fin des n répétitions précédentes.

Remarque. Il y a eu en 2005 des attaques sur les fonctions de hachages. Des collisions ont été trouvées sur diverses fonctions de compression qui ont conduit le N.I.S.T à lancer un nouvel appel à fonction de hachage en 2008. Le fait d'avoir trouver des collisions pour les fonctions de compression n'implique pas de trouver des collisions pour les fonctions de hachage mais constitue une alerte claire qui a conduit à ce nouveau concours.

Fonctions basées sur des problèmes difficiles Il est possible de construire des fonctions de compression pour lesquels on peut prouver que trouver une collision implique la résolution d'un problème difficile. Des fonctions ont été proposées basées sur la résolution de la factorisation dans les années 80's. Depuis de nouvelles fonctions ont été proposées basées sur d'autres problèmes très variés comme les codes correcteurs, les réseaux arithmétiques ou la cryptographie multivariée.

Test 1.30.

On suppose donnée une valeur initiale IV 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 $m = (m_1, \dots, m_l)$ (on suppose que

tous les blocs m_i ont une même longueur n) par $h(m) = E_{m_1}(IV) \oplus E_{m_2}(IV) \oplus \dots \oplus E_{m_l}(IV)$.
 a. Donner une attaque très simple qui relie le fait de trouver un antécédent pour h , à la sécurité de E .
 b. 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 qu'on espère d'une signature manuscrite classique :

- ◇ Personne ne peut imiter la signature.
- ◇ Tout le monde peut vérifier la signature.
- ◇ La signature n'est pas réutilisable.
- ◇ Si le message signé est altéré, la signature n'est plus valide.

- ◊ Non répudiation : un signataire ne peut nier avoir signé un message.

On peut définir une signature digitale comme suit.

Définition 1.29. (Signature digitale à clé publique)

Une signature digitale est une suite d’informations associée à un message et satisfaisant 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

Il existe plusieurs grands types de schémas 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 $ed = 1 \pmod{(p-1)(q-1)}$ et sa clé publique K_P le couple (N, e) .

Méthode	Signature R.S.A
Signature d’un message m par Alice.	
▶ Signature	
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.	
▶ Vérification de la signature	
Bob reçoit un couple (m', s') . Il vérifie que $h(m') = (s')^e \pmod{N}$ 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. ■

Une des nombreuses variantes de signatures digitale 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, doit être signé par une autorité sans que celle-ci connaisse le vote.

Test 1.31. Signature aveugle

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 qu'on sou-

haite 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 du au fait que, pour l'algorithme R.S.A, on peut voir les fonctions $x \rightarrow x^d \pmod{N}$ et $x \rightarrow x^e \pmod{N}$ comme inverse l'une de l'autre. Cette propriété est un peu exceptionnelle 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).

► **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 prends un entier $h \in]1, p-1[$ et on pose $g \stackrel{\text{d'éf}}{=} h^z \pmod{p}$: g est alors d'ordre q .

On choisit un entier $x \in]1, q-1[$ et on calcule $y \stackrel{\text{d'éf}}{=} g^x \pmod{p}$.

CLÉ PUBLIQUE : (p, q, g, y) . CLÉ PRIVÉE : x .

► **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) .

► **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

$$v \stackrel{\text{d'éf}}{=} [g^{u_1} y^{u_2} \pmod{p}] \pmod{q} = r.$$

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 réécrire 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 un standard D.S.S (Digital Signature Standard) qui consiste à utiliser le D.S.A avec la fonction de hachage S.H.A-1.

Test 1.32.

Soit h une fonction de hachage à valeur dans \mathbb{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.

- a. 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.
- b. Montrer comment l'attaque du a. peut être utilisée avec la signature R.S.A ou D.S.A pour

abuser un verificateur de signature.

Test 1.33.

Signature El Gamal

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 $g^{h(m)} = y^r r^s \pmod p$.

- a. Vérifier que le schéma fonctionne.
- b. Quel est l'avantage de D.S.A par rapport à cet algorithme par rapport à 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 approche pour 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 reliée 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 la 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 1.34.

Proposer une méthode à base de certificat 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 croisées 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 au rejeu. 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 par contre 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 du rejeu est de faire un schéma défi/réponse. Ou, pour s'authentifier, plutôt que de donner un mot de passe ou une information qui peut être réutilisée, on envoie un défi pour laquelle 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 (resp. 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. Ceci peut concerner le chiffrement ou la signature.

Test 1.35.

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 clef publique) avec des schémas de signature ou de chiffrement. Souvenons nous que ces protocoles sont plutôt couteux 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 ; ceci 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.

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

- ◇ consistance : pour tout défi, P est capable de donner une réponse
- ◇ signifiante : la probabilité qu’un adversaire puisse se faire passer pour P est faible.
- ◇ à 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 on calcule $A \stackrel{\text{déf}}{=} a^2 \pmod{n}$ (a est une racine carrée de A).

CLÉ PUBLIQUE : (A, N) . CLÉ SECRÈTE : (a, p, q) .

Le protocole de Fiat-Shamir consiste à répéter 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 $y^2 = KA^r \pmod{N}$.

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 clef publique de P) connaisse la clef secrète de P .

Pour démontrer ceci, il nous faut montrer qu'un attaquant qui ne connaît pas la clef 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 $y = ka^r \pmod{N}$ et $A = a^2 \pmod{N}$ et $K = k^2 \pmod{N}$. Vérifions les trois propriétés exigées d'un schéma à divulgation nulle de connaissance.

- ◊ Le protocole est bien consistant 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 $K = k^2 A^{-1} \pmod{N}$ puis comme réponse $y = k \pmod{N}$. V vérifie alors que $y^2 = k^2 = KA = k^2 A^{-1} A \pmod{N} = k^2 \pmod{N}$. 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 est la plus subtile à montrer. Il s'agit de prouver qu'un attaquant qui observe des échanges entre P et V n'obtient aucune information sur le secret de P . Pour montrer cette propriété on va simuler une séquence de tours entre un attaquant et V . On suppose qu'on veut se faire passer pour P avec une sécurité $1/2^t$ et qu'on mène le protocole en cherchant à anticiper le défi 0 ou 1 de V . En moyenne, l'attaquant va anticiper le défi correctement une fois sur deux. Donc, au bout de (en moyenne) $2t$ tours, il aura anticipé t fois de manière positive. L'attaquant garde alors les t tours où il a bien anticipé. Mis bouts à bouts, il forment donc une séquence de t tours réussis. or l'attaquant ne connaît pas le secret de P cela montre donc qu'il ne peut connaître le secret de P .

Paramètres de sécurité Pour N , on prend un module R.S.A avec la longueur de clé adéquate. Si 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 qu'on trouve x et y tels que $x \not\equiv \pm y$ 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 Rivest, Shamir et 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.

Ordinateur quantique et factorisation. 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 amené à un développement de l'étude de systèmes cryptographiques alternatifs aux systèmes basés sur la théorie des nombres et résistants aux algorithmes utilisant sur l'ordinateur quantique.

Cryptographie post-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'appellation *cryptographie post-quantique*. Il faut cependant préciser qu'actuellement l'existence d'un ordinateur quantique efficace reste 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 (SD - pour Syndrom Decoding).

Problème du décodage par syndrome Soit H une matrice $k \times n$ sur F_2^n . Soit s un mot (un syndrome) de F_2^k et soit w un entier positif. Existe-t-il un mot x de F_2^n de poids au plus w tel que $Hx^t = s$?

Test 1.36.

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

Test 1.37.

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 $\epsilon \stackrel{\text{déf}}{=} y - xG$. 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 par le nombre de bons choix possibles (c'est à dire $\binom{n-t}{k}$) divisé par le nombre de choix possible (c'est à dire $\binom{n}{k}$). Comme une matrice aléatoire (carrée) a une bonne probabilité d'être inversible (disons, minorée par $1/2$) 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

► Génération des clés

On considère C un code $[n, k, d]$ de matrice génératrice G qu'on sait décoder jusqu'à t erreurs. Soit S une matrice inversible $k \times k$ quelconque et soit P une matrice de permutations aléatoire sur n colonnes.

CLÉ PUBLIQUE : la matrice $G' = SGP$. CLÉ PRIVÉE : S, G .

► Chiffrement d'un message m

On calcule $c = mG' + e$ où e est une erreur aléatoire de poids t .

► 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. Lorsqu'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 . ■

Paramètres du système. 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 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 $\mathcal{O}(n^2)$.

Paramètres de Mc Eliece. En pratique, Mc Eliece 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} il conviendrait de prendre des codes de Goppa binaire 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 Mc Eliece). 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 qu'on sait décoder et pour lesquels la structure (de par leur grand nombre) est plus difficile à retrouver.

Test 1.38.

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 $\{0, 1, 2\}$.
 a. 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 sur. On pourra montrer que tous les chiffrés c_1, c_2 (où c_i est un bit) n'ont pas la même probabilités d'exister.
 b. Proposer un autre schéma à base de modulo qui serait parfaitement sur.

1.2. **

On considère les 3 registres à décalage : R_1, R_2, R_3 , de polynôme 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, R_3 de sorties respectives x_1, x_2, x_3 , par la fonction booléenne $z = x_1 + x_2 x_3$.
 a. 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.
 b. On suppose qu'on observe pour le générateur combiné précédent la sortie : 1000011100. Retrouver l'initialisation du registre R_1 .

1.3. Déchiffrement R.S.A par les restes chinois **

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 $S_q = m^{d_q} \pmod{q}$.

a. 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 .
 b. Montrer que $S = S_q + q(i_q * (S_p - S_q) \pmod{p})$.
 c. Expliquer l'intérêt (en terme de coût calculatoire) de calculer S par cette méthode plutôt que directement par en calculant $m^d \pmod{n}$?

1.4. Attaque de R.S.A par les restes chinois **

Supposons qu'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. Variation sur le chiffrement de Paillier **

Soit $N = pq$ un module R.S.A. Soit $a \in (\mathbb{Z}/N^2\mathbb{Z})^*$ $g = aN + 1 \pmod{N^2}$. On considère le schéma de chiffrement suivant. La clé publique est (N, g) . Pour chiffrer un message $m \in \mathbb{Z}_N$ on procède de la façon suivante : on prend un h aléatoire dans $[1 \dots N - 1]$, et on calcule $C = g^m \cdot h^N \pmod{N^2}$. On veut trouver un algorithme de déchiffrement.
 a. Montrer que le problème de log discret en base g pour $0 < x < N$ est facile. C'est à dire montrer que pour un g donné et $B = g^x \pmod{N^2}$ il existe un algorithme efficace pour retrouver $x \pmod{N}$. Utiliser le fait que $g = aN + 1$.
 b. Montrer que pour g donné ainsi que la factorisation de N , déchiffrer $C = g^m \cdot h^N \pmod{N^2}$ peut être fait efficacement. Montrer que $C \pmod{N} = h^N \pmod{N}$.
 c. Montrer qu'on peut reconstruire certains messages. Plus précisément : soient a, b , des entiers de $[1, N]$, montrer qu'étant donné N, c ainsi que le chiffré de x et y , il est possible de construire

le chiffré de $x+y$ et le chiffré de $u \cdot x$ pour u dans $(\mathbb{Z}/N\mathbb{Z})^*$. Ce chiffrement est dit homomorphe additif.

1.6. Distribution de clés de groupe **

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

1.7. Schéma de signature de Lamport à usage unique *

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

$$\text{Signature}(m) = (y_{1m_1}, y_{2m_2}, \dots, y_{km_k}) = (s_1, \dots, s_k)$$

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

1.8. Protocole de Schnorr *

On considère le protocole de Schnorr. Soit 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 $A = g^{-a} \text{ mod } p$ est rendue publique. Le protocole est alors le suivant : (1) Alice fournit un engagement aléatoire k dans l'intervalle $[0, q - 1]$ et calcule $K = g^k \text{ mod } p$. Elle transmet K à Bob. (2) Bob choisit un défi r au hasard dans $[0, q - 1]$ et le transmet à Alice. (3) Alice calcule la réponse $y = (k + ar) \text{ mod } q$ et la transmet à Bob. Bob vérifie que $g^y A^r = K \text{ mod } p$.

- a. Faire un schéma de ce protocole. Vérifier que le protocole fonctionne.

- b. Montrer que le protocole est consistant et signifiant pour une probabilité $1/q$.
- c. Quel est l'intérêt de ce protocole par rapport au protocole de Fiat-Shamir (en terme de nombre de passes).

1.9. Paradigme de Fiat-Shamir ***

Il est possible de faire un algorithme de signature avec un protocole de type zero-knowledge avec une probabilité de triche donné.

- a. 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.
- b. Dans le cas de Fiat-Shamir, avec $n = 1024$ quelle est la taille de la signature ?

1.10. Partage de secret de Shamir ***

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 transmis 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. 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. *

- a. On suppose qu'Alice partage une clé de chiffrement par bloc 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.
- b. On suppose maintenant que Alice partage une clé de chiffrement par bloc 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 déchiffrer le message. (Indice : il faut ajouter simplement trois blocs chiffrés bien choisis aux m blocs chiffrés).

c. 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'exo précédent).

1.12. McEliece en forme systématique **

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, mais le problème est qu'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. Signature Courtois-Finiasz-Sendrier ***

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.

a. 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 $1/t!$.

b. En déduire un algorithme de chiffrement utilisant une fonction de hachage ou on décode jusqu'à trouver un mot décodable provenant d'une variation sur le haché du message.

c. Proposer des paramètres pour $t = 9$ et une sécurité de 2^{80} . Que pensez vous des paramètres de cet algorithme ?

Deuxième partie

SOLUTIONS DES TESTS

Troisième partie

SOLUTIONS DES TESTS

0.1. Si 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 on passe les coefficients de M en inconnues, on se ramène à la résolution d'un système. Un message et son chiffré donne 3 équations, avec au minimum 3 clairs et leurs chiffrés on peut retrouver les 9 coefficients inconnus de M .

0.4. 1 (mod 13) et 13 (mod 15).

0.5. 33

0.6. $\phi(112) = \phi(7 \cdot 16) = \phi(6) \cdot \phi(2^4) = 5 \cdot 8 = 40$. $\phi(84) = \phi(3) \phi(4) \phi(7) = 2 \cdot 2 \cdot 6 = 24$.

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

0.8. 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.

0.9. On applique la procédure de déchiffrement.

0.10. Dans ce cas 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 $M_1 \oplus M_2$. En 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 à résoudre un système linéaire.

0.11. La preuve suit le cas du double DES 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.

0.12. Pour le DESV on peut écrire $m \xrightarrow{\text{DES}_{K_2}} r \xrightarrow{\oplus K_1} c$. En prenant pour l'attaque par le milieu 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 DESW : $m \xrightarrow{\oplus K_2} r \xrightarrow{\text{DES}_{K_1}} c$. Pour le DESX, le fait d'avoir 3 clés empêche de casser en deux morceaux indépendants et on ne peut mener cette attaque.

0.13. On procède avec une attaque par le milieu en $m \xrightarrow{\text{DES}_{k_2}} r \xrightarrow{\text{DES}_{k_1}} c$. 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.

0.14. On applique la procédure de déchiffrement.

0.15. La suite engendrée est : 1001000111101011001..., période maximale 15.

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

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

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

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

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 $d \cdot c_B \equiv 1 \pmod{N}$. 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 x un nombre au hasard on envoie $c' = x^e c$, on reçoit $c'^d = x m$ 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 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 ordre 100 donc on peut prendre $d = 67$: comme $3 \cdot 67 = 1 \pmod{100}$, 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.** On prend, pour n fixé, la fonction de $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ qui à x associe x . Cette fonction est bien sans collision, mais elle n'est pas à sens unique puisque l'on peut facilement trouver des antécédents.
- 0.29.** $1/10 \cdot 2/9 = 1/45$.
- 0.30.** a. Si on est capable d'inverser E alors on est capable de trouver un antécédent pour h . b. h^n n'est pas sans collision car il suffit de prendre une permutation sur les m_i pour trouver une collision.
- 0.31.** 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 $(x^e m)^d = x m^d \pmod{N}$ comme x est aléatoire le signataire n'a aucun moyen de savoir quel message est signé.
- 0.32.** a. 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 pas. Cela donne $2^{n/2}$ message qui ont le même sens, mais il peut y avoir un léger décalage entre les mots auquel on ne fait pas forcément attention. Par paradoxe des anniversaires on peut donc trouver une collision pour un haché de longueur n avec ces $2^{n/2}$ possibilités. b. 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.
- 0.33.** Par définition on a : $h(m) = xr + ks \pmod{p-1}$. Soit par le petit théorème de Fermat : $g^{h(m)} = g^{xr} g^{sk} = (g^x)^r (g^k)^s = y^r r^s \pmod{p}$. L'intérêt de D.S.A est que la signature est de 320 bits contre 1024 pour El Gamal (à sécurité équivalente).
- 0.34.** On suppose que tout le monde a la clé publique d'une autorité. Pour signer un message on envoie, 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, puis si c'est bon la clé publique du signataire est validée et on peut vérifier la signature. De façon il suffit simplement d'avoir la clé publique d'une autorité et pas besoin de passer par un annuaire. La vérification se fait donc sans connexion nécessaire.
- 0.35.** On fait 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 qu'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'il connaît le secret. Pour la signature le défi est de signer un message aléatoire qu'on vérifie avec la clé publique.
- 0.36.** 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 (si on la prend au hasard comme le déterminant vaut 0 ou 1, on peut facilement en trouver). Une solution du problème est donc $x = M^{-1} s$ en mettant à 0 les colonnes ne correspondant pas à M .
- 0.37.** Si on a à résoudre un problème de décodage. Matrice génératrice G , de matrice de parité H et $y = xG + e$ avec $w(e) = w$. On a $H \cdot y^t = H \cdot e^t$ car $G \cdot H^t = 0$. Ainsi trouver e se ramène à résoudre le problème de décodage par syndrome pour $s = H \cdot y^t$ et le poids w de e .
- 0.38.** 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} .

Quatrième partie

SOLUTIONS DES EXERCICES

Cinquième partie

SOLUTIONS DES EXERCICES

0.39.a. 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 entrée 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. b. On écrit les 3 possibilités sous la forme $\{0, 1, 2\}$ et es 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.

0.40.a. On attaque sur toutes les initialisations possibles, R_1 est de longueur plus petite donc moins d’essais à faire. $z = x_1$ dès que $x_2 = 0$ ou $x_3 = 0$ soit 3/4 du temps. c. On essaie les 8 initialisation possibles, l’initialisation 101 donne 101001110 qui a une corrélation proche de 3/4.

0.41.a. $d = d_p + k(p - 1)$ pour $k \in \mathbb{Z}$ de même $d = d_q + k'(q - 1)$. $S \pmod p = m^d = m^{d_p + k(p-1)} \pmod p$ comme p est premier $m^{p-1} = 1 \pmod p$ ce qui donne le résultat de même pour S_q . b. On peut vérifier directement l’égalité ou la retrouver par le théorème des restes chinois puisqu’on connaît $S \pmod q$ et $S \pmod p$. c. Pour retrouver S il suffit alors de calculer S_p et S_q . Si 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 cout d’une multiplication, on gagne un facteur 4.

0.42.Le chiffré $c_1 = m^3 \pmod{N_1}$, 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 $m^3 \pmod{N_1 N_2 N_3}$. Comme $m^3 < N_1 N_2 N_3$ on peut enlever le modulo et retrouver m .

0.43.a. On a $g = aN + 1$, donc si $B = g^x \pmod{N^2}$ par le développement avec le binôme de Newton $(N + 1)^x = 1 + xaN + N^2(P(N))$ où P est un certain polynôme en N . Comme on travaille modulo N^2 on obtient $g^x \pmod{N^2} = 1 + xaN \pmod{N^2} = B$ et $x = \frac{a^{-1}(B-1)}{N}$. b. $C^{\phi(N)} \pmod{N^2} = g^{\phi(N)} m h^{N\phi(N)} \pmod{N^2}$. Mais comme $\phi(N^2) = N\phi(N)$, $h^{N\phi(N)} = 1 \pmod{N^2}$ et donc $C^{\phi(N)} = g^{\phi(N)} a^m \pmod{N^2}$. $\phi(N)a$ est inversible mod N^2 don on peut appliquer le a. c. $\text{Chiffré}(x) = g^x h_x^N$ et $\text{Chiffré}(y) = g^y h_y^N$ donc $\text{Chiffré}(x + y) = g^{x+y} (h_x h_y)^N$, de même $\text{Chiffré}(ux) = (g^x)^u (h_c^u)^N$.

0.44.On prend un leader qui choisit un secret a , il reçoit de chacun des membres du groupes M , $x = g^{b_M^{-1}} \pmod p$ et renvoie $x^a \pmod p$. Chaque membre du groupe calcule alors x^{b_M} $\pmod p$ et obtient le même $g^a \pmod p$.

0.45.a. 2.256.256 = 16kilobits. C’est beaucoup plus gros que les autres tailles de clés. b. Pour casser le schéma il faut être capable d’inverser la fonction de hachage pour les valeurs liées à la clé. Ce qui est a priori infaisable d’après les propriétés de la fonction de hachage. c. Non car il n’y aurait pas assez de signatures possibles (2^k). d. A chaque fois qu’on donne une signature on donne des antécédents pour les valeurs de la signature. Si on prend deux vecteurs $a(a_1, \dots, a_k)$ et $b(b_1, \dots, b_k)$ tels que $a_i \neq b_i$, on peut alors obtenir tous les antécédents. Et on est capable de signer.

0.46.a. On a bien $g^y A^r = g^{k+ar} \cdot g^{-ar} = g^k = K \pmod p$. b. Le protocole est constant 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. c. Si on prend $q = 2^k$ on obtient une sécurité en un tour quand il en faudrait k pour Fiat-Shamir.

0.47.a. On choisit au départ un paramètre de sécurité en 2^{-k} . On choisit k engagements distincts comme si on allait faire k tours. Ces engagements sont concaténés au message à signer. On hache alors le tout et 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 calculé convenablement et on vérifie le protocole. Comme les défis sont choisis aléatoirement (à cause de la fonction de hachage) après le choix des engagements, on est sur qu’ils n’ont pas été anticipés. b. 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.

0.48.Soit la valeur secrète r . On choisit un corps finis tel que le secret soit de la taille d’un élément du corps. Par exemple, un secret de 128 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, par interpolation ils sont capables de reconstruire f car f est de degré k . Si moins de k personnes collaborent, ils retrouveront un polynômes à une indéterminée près. Comme le secret est justement une constante, ils ne peuvent retrouver d’information supplémentaires sur le secret.

0.49.a. A choisit une clé de session k puis la casse en deux morceaux k_1 et k_2 tels que $k_1 + k_2$. Elle envoie un message commençant par $E_{K_{AB}}(k_1)$ et $E_{K_{AB}}(k_2)$, puis $E_k(M)$. Pour déchiffrer B et C auront à collaborer

pour reconstruire k à partir de k_1 et k_2 . b. On écrit cette fois $k = k_1 + k_2 + k_3$ et on envoie chiffré 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 et retrouver k , un seul ne peut pas. c. On peut soit généraliser un point de vue combinatoire comme au b. soit appliquer le schéma de partage de secret de Shamir de l'exercice précédent.

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

0.51.a. 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 nbre de mots du code fois nombre de mots dans la boule divisé par nombre de mots de l'espace, soit $\frac{\binom{n}{t} 2^{m-t}}{2^m}$. On peut approximer $\binom{n}{t}$ par $\frac{n^t}{t!}$, ce qui en remplaçant dans l'égalité de la densité une densité en $1/t!$. b. La question a nous dit que si 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. A partir du message à signer m on peut donc mettre en place un compteur et decoder un mot de l'espace \mathbb{F}_2^m dérivant de $h(m \oplus i)$ (pour h une fonction de hachage) en augmentant i jusqu'à qu'on obtienne un mot décodable (en moyenne au bout de $t!$ essais). Le secret est donc le fait d'être capable de decoder 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 decoder. 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. c. Comme il faut faire $t!$ décodage en moyenne avant de trouver le bon mot, on ne peut prendre t trop grand. En même temps pour t petit on peut attaquer plus facilement. Si on se base 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 8Megabits. C'est beaucoup! Mais cela donne un schéma de signature avec les codes.