

Damien Vergnaud

Préface de Jacques Stern

Exercices et problèmes de cryptographie

2^e édition

DUNOD

Illustration de couverture :
Energy of fractal realms © agsandrew-Fotolia.com

Le pictogramme qui figure ci-contre mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit, particulièrement dans le domaine de l'édition technique et universitaire, le développement massif du photocollage. Le Code de la propriété intellectuelle du 1^{er} juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établissements d'enseignement supérieur, provoquant une baisse brutale des achats de livres et de revues, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée. Nous rappelons donc que toute reproduction, partielle ou totale, de la présente publication est interdite sans autorisation de l'auteur, de son éditeur ou du Centre français d'exploitation du droit de copie (CFC, 20, rue des Grands-Augustins, 75006 Paris).



© Dunod, 2012, 2015

5 rue Laromiguière, 75005 Paris

www.dunod.com

ISBN 978-2-10-072281-5

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5, 2^e et 3^e a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

PRÉFACE

« Pour devenir habile en quelque profession que ce soit, il faut le concours de la nature, de l'étude et de l'exercice ». Cette maxime d'Aristote semble bien mal s'appliquer à la cryptologie tant l'exercice y est absent. Il existe de multiples ouvrages de référence de qualité mais, pour la plupart, ils sollicitent très peu l'initiative des étudiants. Et même ceux – rares – qui sont accompagnés d'un véritable choix de problèmes à résoudre, par exemple sous forme d'un livre compagnon, ne couvrent pas totalement une discipline qui connaît une évolution rapide. C'est donc un réel manque que vient combler le recueil que propose Damien Vergnaud.

Le livre que j'ai le plaisir de présenter est issu d'un vrai travail de terrain puisqu'il est le résultat de plusieurs années d'enseignement de la cryptologie à l'École normale supérieure. À l'évidence, l'auteur a beaucoup de talent pour éveiller l'intérêt des étudiants et les conduire, pas à pas, à s'approprier les concepts et les méthodes de la science du secret. Beaucoup de culture également, puisque les sujets choisis sont extrêmement variés à l'image d'une science qui emprunte à l'algèbre, à la théorie des probabilités, à l'algorithmique, à la théorie de l'information. D'ailleurs, ils débordent largement le cadre strict de la cryptographie. Ce talent et cette culture conduisent à un choix d'exercices qui ne demandent pas simplement à l'étudiant de faire des gammes mais lui proposent de s'attaquer à de véritables compositions : ici un effort raisonnable de programmation illustre des cryptanalyses célèbres comme celle de l'Enigma ou celle du programme Venona qui a permis l'interception de communications où les services russes mettaient incorrectement en œuvre le chiffrement jetable ; là une invitation à « mettre la main à la pâte » permet d'entrer de plain-pied dans les méthodes modernes de cryptanalyse – différentielle et linéaire – des algorithmes conventionnels tels que le DES ou l'AES ; là encore, une initiation progressive aux méthodes de factorisation d'entiers, intimement liées à la sécurité du RSA est proposée.

Présenter un tel ouvrage comme un simple livre d'exercices est le reflet de la modestie de son auteur. Certes, il permet la pratique nécessaire à l'acquisition des éléments essentiels de la cryptologie. Mais il va au-delà de cet objectif : chaque chapitre inclut une présentation qui est un véritable cours d'introduction et l'ensemble constitue de fait une forme d'ouvrage d'enseignement avancé fondé sur la pratique. En d'autres termes, le lecteur qui va au terme de tous les exercices proposés est déjà un

Exercices et problèmes de cryptographie

véritable spécialiste, capable de se confronter aux multiples concepts que la cryptologie moderne a développés ces trente dernières années. À un moment où la cryptologie est au cœur de la société de l'information, de l'internet aux moyens de paiement en passant par les téléphones portables, une telle expertise est indispensable et il faut souhaiter au livre de Damien Vergnaud des lecteurs à la fois nombreux et actifs.

Jacques STERN
Professeur à l'École normale supérieure

TABLE DES MATIÈRES

Préface	V
Avant-propos	XIII
Notations	XV
Chapitre 1. Cryptographie classique	1
1.1 Chiffrement par substitution mono-alphabétique	1
Exercice 1.1 (avec programmation). Chiffrement de César	3
Exercice 1.2 (avec programmation). Chiffrement affine	4
Exercice 1.3 (avec programmation).	
Chiffrement par substitution mono-alphabétique	5
1.2 Chiffrement par substitution poly-alphabétique	8
Exercice 1.4 (avec programmation).	
Chiffrement de Vigenère – test de Kasiski	9
Exercice 1.5 (avec programmation).	
Chiffrement de Vigenère – indice de coïncidence	11
Exercice 1.6. Chiffrement de Hill – nombre de clés	12
Exercice 1.7. Chiffrement de Hill – attaque à clair connu	13
1.3 Chiffrement par transposition	14
Exercice 1.8 (avec programmation). Scytale	15
Exercice 1.9 (avec programmation).	
Chiffrement par transposition par colonnes	16
1.4 Chiffrement parfait	17
Exercice 1.10. Carré latin	18
Exercice 1.11 (avec programmation).	
Mauvaise utilisation du chiffrement jetable	20
Problème 1.12. Algorithme de Viterbi	20
1.5 La machine Enigma	22
Exercice 1.13. Enigma – Nombre de clés	24
Exercice 1.14 (avec programmation). Enigma – Tableau de connexions	25
Problème 1.15. Enigma – Indice de coïncidence	27
Chapitre 2. Chiffrement par bloc	31
2.1 Modes opératoires	32
Exercice 2.1. Modes opératoires et propriétés de sécurité	34
Exercice 2.2. Mode opératoire CBC *	36
Problème 2.3. Attaque sur le mode CBC avec le processus de bourrage RFC2040	38
2.2 Schémas de Feistel	39
Exercice 2.4. Schéma de FEISTEL à un ou deux tours	40
Exercice 2.5. Sécurité du schéma de FEISTEL à trois tours	42
Exercice 2.6. Distinguiseur pour le schéma de FEISTEL à trois tours*	43

Exercices et problèmes de cryptographie

2.3 Chiffrement DES	45
Exercice 2.7. Clés faibles et semi-faibles du chiffrement DES	46
Exercice 2.8. Propriété de complémentation du chiffrement DES	48
Exercice 2.9. Chiffrement DES avec blanchiment	49
Exercice 2.10. Construction de Even-Mansour	50
Exercice 2.11. Double chiffrement	51
Exercice 2.12. Chiffrement Triple-DES avec deux clés indépendantes	52
Exercice 2.13. Mode opératoire CBC-CBC-ECB	53
2.4 Chiffrement AES	55
Exercice 2.14 (avec programmation). S-Boîte de l'AES	57
Exercice 2.15 (avec programmation). Opération MixColumns	59
Exercice 2.16. Propriétés de l'opération MixColumns	61
Exercice 2.17 (avec programmation). Diversification de clé de l'AES	63
Chapitre 3. Fonctions de hachage – Techniques avancées de cryptanalyse	65
3.1 Généralités sur les fonctions de hachage	66
Exercice 3.1. Résistance à la pré-image et aux collisions	66
Exercice 3.2. Construction de Merkle-Damgaard	67
Exercice 3.3 (avec programmation). Collisions sur la fonction MD5 tronquée	71
3.2 Chiffrement par bloc et fonction de compression	72
Exercice 3.4. Chiffrement par bloc et fonction de compression	72
Exercice 3.5. Sécurité de la construction de Matyas-Meyer-Oseas avec le DES	73
Exercice 3.6. Attaque en pré-image pour la construction de M. O. RABIN	74
3.3 Attaques génériques sur les fonctions de hachage itérées	76
Exercice 3.7. Multi-collisions pour les fonctions de hachage itérées	76
Exercice 3.8. Attaque en collision contre fonctions de hachage concaténées	78
Problème 3.9. Attaque de Kelsey-Schneier	79
3.4 Cryptanalyse différentielle	82
Exercice 3.10 (avec programmation). Table des différences du DES	83
Problème 3.11. Cryptanalyse différentielle de FEAL-4	85
3.5 Cryptanalyse différentielle impossible	89
Exercice 3.12. Attaque par différentielle impossible contre DEAL	89
Problème 3.13. Attaque par différentielle impossible contre l'AES	92
3.6 Cryptanalyse linéaire	96
Exercice 3.14 (avec programmation). Table d'approximation linéaire du DES	96
Exercice 3.15. Approximation linéaire de l'addition	98
Problème 3.16. Cryptanalyse linéaire de SAFER	100
Exercice 3.17. Biais de la parité d'une permutation	102
3.7 Attaques par saturation	104
Problème 3.18. Attaque par saturation contre l'AES	104
Exercice 3.19. Attaque par distinguiseur sur Ladder-DES	108

Chapitre 4. Chiffrement par flot	111
4.1 Registres à décalage à rétroaction linéaire	111
Exercice 4.1. LFSR et polynômes de rétroaction	113
Exercice 4.2. Propriétés statistiques d'une suite produite par un LFSR	115
Exercice 4.3. Reconstruction du polynôme de rétroaction minimal	115
4.2 Chiffrement par flot par registres à décalage irrégulier	116
Exercice 4.4 (avec programmation). Distinguier sur le générateur à signal d'arrêt	117
Problème 4.5. Propriétés du générateur par auto-rétrécissement	119
4.3 Chiffrement par flot par registre filtré	120
Exercice 4.6. Attaque « deviner et déterminer » sur Toyocrypt	121
Exercice 4.7. Attaque algébrique sur Toyocrypt*	122
4.4 Chiffrement par flot par registres combinés	124
Exercice 4.8. Attaque par corrélation sur le générateur de Geffe	125
Exercice 4.9. Attaque « deviner et déterminer » sur le générateur de Geffe	127
Exercice 4.10. Attaque algébrique sur le générateur de Geffe	127
4.5 Le chiffrement par flot A5/1	128
Exercice 4.11. États internes de A5/1	129
Exercice 4.12. Attaque par compromis temps-mémoire sur A5/1	131
Problème 4.13. Attaque « deviner et déterminer » sur A5/1	132
4.6 Le chiffrement par flot RC4	135
Exercice 4.14. Cryptanalyse de RC4 sans opération d'échange*	136
Exercice 4.15. Biais de la suite chiffrante produite par RC4	137
Problème 4.16. Attaque par recouvrement de clé sur RC4	139
Chapitre 5. Problème du logarithme discret	143
5.1 Logarithme discret dans un groupe générique	143
Exercice 5.1. Multi-exponentiation	145
Exercice 5.2. Algorithme de Shanks	146
Exercice 5.3. Algorithme ρ de Pollard	148
Exercice 5.4. Algorithme de Pohlig-Hellman	151
Exercice 5.5 (avec programmation). Application de l'algorithme de Pohlig-Hellman	153
5.2 Problème du logarithme discret dans $(\mathbb{Z}/p\mathbb{Z})^*$	154
Exercice 5.6. Entiers friables	155
Problème 5.7. Méthode de Kraitchik – Calcul d'indice *	157
5.3 Problèmes algorithmiques liés au logarithme discret	161
Exercice 5.8. Auto-réductibilité du problème du logarithme discret	161
Exercice 5.9. Algorithme λ de Pollard	163
Problème 5.10. Logarithme discret de petit poids de Hamming	166
5.4 Interpolation polynomiale de logarithme discret	169
Exercice 5.11. Polynôme d'interpolation du logarithme discret	169
Exercice 5.12. Interpolation polynomiale de logarithme discret – Borne inférieure	170

Exercices et problèmes de cryptographie

Chapitre 6. Factorisation des entiers et primalité	173
6.1 Tests de primalité	173
Exercice 6.1. Certificats de primalité de Pratt	174
Exercice 6.2. Nombres pseudo-premiers de Fermat en base a	175
Problème 6.3. Nombres de Carmichael – Critère de Korselt	176
Exercice 6.4 (avec programmation). Recherche de nombres de Carmichael	178
Exercice 6.5. Test de primalité de Solovay-Strassen	181
Problème 6.6. Test de primalité de Miller-Rabin	183
Exercice 6.7. Identité de Agrawal-Kayal-Saxena	185
Exercice 6.8. Nombres de Fermat et test de primalité de Pépin	186
6.2 Méthodes exponentielles de factorisation	188
Exercice 6.9 (avec programmation). Factorisation par divisions successives	188
Exercice 6.10 (avec programmation). Factorisation par la méthode Fermat	189
Exercice 6.11. Algorithme de Lehman *	189
Exercice 6.12. Méthode $p - 1$ de Pollard	192
Exercice 6.13 (avec programmation). Factorisation par la méthode $p - 1$ de Pollard	193
Exercice 6.14. Algorithme ρ de Pollard	194
6.3 Multi-évaluation de polynômes et algorithme de Pollard-Strassen	195
Exercice 6.15. Division euclidienne rapide par la méthode de Newton	196
Exercice 6.16. Multi-évaluation d'un polynôme univarié	198
Exercice 6.17. Algorithme de Pollard-Strassen	200
6.4 Racine carrée modulaire et factorisation	202
Exercice 6.18. Extraction de racine carrée modulo p	202
Exercice 6.19. Extraction de racine carrée modulo p^ℓ	204
Exercice 6.20. Extraction de racine carrée modulo N	205
Problème 6.21. Carrés modulaires friables	207
Exercice 6.22. Factorisation et logarithme discret	211
Chapitre 7. Chiffrement à clé publique	213
7.1 Fonction RSA	213
Exercice 7.1. Fonction RSA et factorisation	214
Exercice 7.2. Auto-réducibilité du problème RSA	215
Problème 7.3. Sécurité des bits de la fonction RSA	217
7.2 Chiffrement RSA	219
Exercice 7.4. Sécurité du protocole de chiffrement RSA naïf	220
Exercice 7.5. RSA avec module commun	220
Exercice 7.6 (avec programmation).	
Diffusion de données chiffrées avec RSA	221
Exercice 7.7. Attaque de Wiener	223
Exercice 7.8 (avec programmation). Attaque de Wiener	224
Exercice 7.9 (avec programmation). RSA et clairs liés	226
Exercice 7.10. RSA et petits textes clairs	227
Problème 7.11. Implantation du chiffrement RSA et théorème chinois des restes	228
7.3 Mise en accord de clé de Diffie-Hellman	230
Exercice 7.12. Attaque par le milieu	231
Problème 7.13. Logarithme discret et Diffie-Hellman *	232

Table des matières

7.4 Chiffrement d'ElGamal et variantes	235
Exercice 7.14. Sécurité du chiffrement d'ElGamal naïf	235
Exercice 7.15. Sécurité des bits du logarithme discret	237
Exercice 7.16. Attaque sur le chiffrement d'ElGamal par petit sous-groupe	239
Chapitre 8. Signatures numériques	241
8.1 Signatures basées sur la primitive RSA	241
Exercice 8.1. Sécurité du protocole de signature RSA naïf	242
Exercice 8.2. Sécurité des protocoles de signature de De Jonge et Chaum	243
Exercice 8.3. Sécurité de \mathcal{F} -RSA et propriétés de \mathcal{F}	245
Exercice 8.4. Sécurité de \mathcal{F} -RSA pour la recommandation CCITT	247
Exercice 8.5. Sécurité de \mathcal{F} -RSA avec encodage PKCS #1 v1.5	249
Exercice 8.6. Contrefaçon existentielle de \mathcal{F} -RSA avec redondance linéaire	252
Exercice 8.7. Contrefaçon universelle de \mathcal{F} -RSA avec redondance linéaire *	253
Problème 8.8. Sécurité du protocole de signature de Boyd	254
8.2 Signatures d'ElGamal et variantes	258
Exercice 8.9. Contrefaçon existentielle du schéma de signature d'ElGamal naïf	258
Exercice 8.10. Contrefaçon universelle du schéma de signature d'ElGamal naïf	259
Exercice 8.11. Vérification des signatures d'ElGamal	260
Exercice 8.12. Fonction de hachage et sécurité des signatures de Schnorr	261
Exercice 8.13 (avec programmation). Paramètres publics dans le protocole DSA	262
Exercice 8.14. Clé temporaire et sécurité des signatures d'ElGamal	263
8.3 Signatures de Lamport et variantes	265
Exercice 8.15. Sécurité et efficacité des signatures de Lamport	265
Exercice 8.16. Espace de message de la signature de Lamport	266
Exercice 8.17. Extension de l'espace des messages des signatures de Lamport *	267
Exercice 8.18. Arbres de Merkle	269
Problème 8.19. Sécurité du protocole de signature de Groth	271
Bibliographie	275
Index	281

AVANT-PROPOS

La cryptologie est un ensemble de techniques permettant d'assurer la sécurité des systèmes d'information. Cette discipline permet notamment de conserver aux données leur caractère de confidentialité, de contrôler leur accès ou d'authentifier des documents. L'utilisation de la cryptographie est de plus en plus répandue et les utilisateurs des systèmes cryptographiques doivent être en mesure non seulement de comprendre leur fonctionnement mais aussi d'en estimer la sécurité.

Cet ouvrage s'adresse aux étudiants de second cycle d'informatique ou de mathématiques. Il s'est développé à partir de textes de travaux dirigés et de travaux pratiques proposés à des étudiants du Master parisien de recherche en informatique (MPRI) et aux élèves de première année de l'École normale supérieure. Il a été conçu pour aider à assimiler les connaissances d'un cours d'introduction à la cryptologie et à se préparer aux examens. Il présente les outils mathématiques et algorithmiques utiles en cryptographie et les fonctionnalités cryptographiques de base dans le cadre de la cryptographie symétrique et asymétrique.

Les exercices destinés aux étudiants de « master 1 » pourront cependant être abordés par un étudiant motivé de licence ayant un goût pour l'algorithmique dans ses aspects mathématiques et pratiques. À l'intention des étudiants plus avancés, nous avons inclus des énoncés plus difficiles qui sont alors signalés par un astérisque. Enfin, l'ouvrage sera utile aux enseignants de cryptologie qui y trouveront un support pour leurs travaux dirigés.

La cryptologie est liée à d'autres disciplines mathématiques et informatiques comme l'arithmétique, l'algèbre, l'algorithmique, ou la théorie de la complexité. Le bagage informatique et mathématique requis pour aborder ce livre est celui que l'on acquiert lors des deux premières années de licence ou en classes préparatoires scientifiques augmenté de quelques notions de théorie des nombres de niveau troisième année. Ces notions plus avancées font l'objet de brefs rappels qui n'ont cependant pas pour ambition de remplacer un livre de cours.

Le but de cet ouvrage est de permettre à ceux qui le souhaitent de s'initier à la cryptographie par l'exemple. Il propose plus d'une centaine d'exercices et problèmes entièrement utilisés dans le cadre de travaux dirigés, de travaux pratiques ou d'examens. Ces exercices sont entièrement corrigés mais le lecteur ne tirera profit de ce livre que s'il cherche des solutions personnelles avant d'en étudier les corrections. L'étude de la cryptologie moderne ne peut se concevoir sans un ordinateur à portée

Exercices et problèmes de cryptographie

de main et le livre propose de nombreux exercices de programmation qui ont pour but notamment d'acquérir une pratique de la cryptanalyse. Les données numériques de ces exercices sont disponibles en ligne sur :

www.dunod.com/contenus-complementaires/9782100721450

Le lecteur pourra recopier les énoncés des exercices avant de les traiter. Il trouvera également une vingtaine d'exercices supplémentaires et des références complémentaires.

Cette **deuxième édition** a été inspirée par les nombreuses demandes et remarques que m'ont envoyées des étudiants et collègues, utilisateurs de la première édition. Cette nouvelle édition m'a donné l'occasion de modifier et réécrire des parties importantes du texte en suivant ces remarques sur le contenu, le style et l'organisation de l'ouvrage. En plus d'épurer le texte de ses inévitables erreurs typographiques et coquilles, j'ai simplifié et clarifié une grande partie des énoncés des exercices et de leurs solutions. J'ai notamment ajouté des questions intermédiaires pour simplifier la résolution de certains exercices et détaillé certains points techniques dans des solutions d'exercices complexes. J'ai supprimé certains exercices jugés trop difficiles et j'en ai également ajouté de nouveaux. Enfin, les compléments en ligne qui accompagnent l'ouvrage ont été enrichis d'une vingtaine d'exercices supplémentaires (avec leurs solutions complètes) et d'autres exercices et compléments de cours seront ajoutés progressivement.

Remerciements

J'adresse un chaleureux merci à DAVID NACCACHE et JACQUES STERN avec qui j'ai eu le plaisir d'enseigner le cours d'*Introduction à la cryptologie*. Ma gratitude va également aux étudiants du MPRI et aux élèves normaliens de ces dernières années qui ont testé, malgré eux, la majorité des exercices présentés dans ce livre. Ce texte doit beaucoup à des conversations de couloirs et je tiens également à remercier les doctorants, post-doctorants et membres permanents de l'équipe Cryptographie de l'ENS – et particulièrement PIERRE-ALAIN FOUCHE – pour toutes les discussions que nous avons pu avoir. Pour terminer, je voudrais remercier AURÉLIE BAUER, GUILHEM CASTAGNOS, CÉLINE CHEVALIER, AURORE GUILLEVIC, PIERRE-ALAIN FOUCHE, FABIEN LAGUILLAUMIE, ROCH LESCUYER et JULIETTE VERGNAUD-GAUDUCHON pour la rigueur et la pertinence de leurs nombreux commentaires.

NOTATIONS

Les conventions et notations suivantes sont utilisées dans cet ouvrage :

a) Ensembles

Nous utilisons les notations ensemblistes classiques : \emptyset désigne l'ensemble vide ; $x \in A$ signifie que x est un élément de l'ensemble A . Pour deux ensembles A et B , $A \subseteq B$ indique que A est un sous-ensemble de B (alors que $A \subset B$ indique que A est un sous-ensemble strict de B). De plus, $A \cup B$ désigne la réunion de A et B , $A \cap B$ désigne l'intersection de A et B , $A \setminus B$ l'ensemble des éléments de A qui ne sont pas dans B et $A \times B$ le produit cartésien des ensembles A et B . Le cardinal d'un ensemble A est noté $\#A$. Nous utilisons les notations classiques suivantes pour désigner certains ensembles :

\mathbb{N}	ensemble des entiers naturels
\mathbb{P}	ensemble des nombres premiers
\mathbb{Z}	anneau des entiers relatifs
\mathbb{Q}	corps des nombres rationnels
\mathbb{R}	corps des nombres réels
\mathbb{C}	corps des nombres complexes
$(\mathbb{Z}/N\mathbb{Z})$	anneau des résidus modulo un entier $N \geq 1$
\mathbb{F}_q	corps fini à q éléments
\mathfrak{S}_A	groupe de permutations de l'ensemble A
A^*	groupe des éléments inversibles d'un anneau A
$\mathcal{M}_\ell(A)$	anneau des matrices carrées $\ell \times \ell$ à coefficients dans un anneau A
$A[X]$	anneau des polynômes à une indéterminée X à coefficients dans un anneau A

La lettre p désigne le plus souvent un nombre premier $p \in \mathbb{P}$ et nous notons $(p_n)_{n \geq 1}$ la suite croissante des nombres premiers (avec $p_1 = 2, p_2 = 3, \dots$). Pour un polynôme $P \in A[X]$, nous notons $\deg P$ le degré de P . La notation \mathbb{G} désigne un groupe dont la loi est notée multiplicativement. L'élément neutre pour la multiplication dans \mathbb{G} est noté $1_{\mathbb{G}}$. L'ordre d'un groupe \mathbb{G} est noté $|\mathbb{G}| = \#\mathbb{G}$ et $\langle g \rangle$ désigne le sous-groupe de \mathbb{G} engendré par $g \in \mathbb{G}$.

b) Fonctions

Nous notons $f : A \longrightarrow B$ pour indiquer que f est une fonction d'un ensemble A dans un ensemble B . Pour un sous-ensemble $A' \subseteq A$, nous notons $f(A') = \{f(a), a \in A'\} \subseteq$

Exercices et problèmes de cryptographie

B. Pour un sous-ensemble $B' \subseteq B$, nous notons $f^{-1}(B') = \{a \in A, f(a) \in B'\} \subseteq A$. La composition de fonctions est notée \circ . Nous utilisons les notations classiques suivantes pour désigner certaines fonctions :

$\lfloor x \rfloor$	partie entière par défaut de $x \in \mathbb{R}$ ($\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$)
$\lceil x \rceil$	partie entière par excès de $x \in \mathbb{R}$ ($\lceil x \rceil - 1 < x \leq \lceil x \rceil$)
$\ln(x)$	logarithme népérien de $x \in \mathbb{R}$ ($x > 0$)
$\log(x)$	logarithme en base 2 de $x \in \mathbb{R}$ ($x > 0$)
$\log_g(h)$	logarithme discret de $h \in \langle g \rangle$ en base $g \in \mathbb{G}$
$\pi(x)$	nombre de nombres premiers inférieurs ou égaux à x ($\#\{p \in \mathbb{P}, p \leq x\}$)
$\Psi(x, y)$	fonction de Dickman-De Bruijn ($\#\{n \in \mathbb{N}, n \leq x \text{ et } n \text{ est } y\text{-friable}\}$)
$\binom{n}{m}$	coefficient binomial $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ pour $0 \leq m \leq n$
$\left(\frac{x}{m}\right)$	symbole de Jacobi
$\text{pgcd}(a, b)$	plus grand commun diviseur de $a, b \in \mathbb{Z}$
$\text{ppcm}(a, b)$	plus petit commun multiple de $a, b \in \mathbb{Z}$
$\Pr(E)$	probabilité d'un événement E

c) Chaînes binaires

Nous utilisons les notations classiques suivantes sur les chaînes binaires :

$\{0, 1\}^n$	ensemble des chaînes binaires de longueur n
$\{0, 1\}^*$	ensemble des chaînes binaires de longueur finie
\wedge	ET logique (bit à bit pour deux chaînes de même longueur)
\vee	OU logique (bit à bit pour deux chaînes de même longueur)
\neg	NON logique (bit à bit pour deux chaînes de même longueur)
\oplus	« OU exclusif » (bit à bit pour deux chaînes de même longueur)
$ x $	longueur binaire d'une chaîne $x \in \{0, 1\}^*$
\overline{x}	chaîne binaire complémentaire de x ($\overline{x} = \neg x = x \oplus 1^n$ avec $n = x $)
$x y$	concaténation des chaînes x et y
x^n	concaténation de la chaîne x n fois ($\underbrace{x \dots x}_{n\text{fois}}$)
$\lll i$	rotation à gauche d'une chaîne de bits de i positions

Dans les chapitres 2, 3 et 4, nous utilisons une fonte de type « machine à écrire » pour représenter la valeur d'un octet avec deux chiffres hexadécimaux : **00** = 0, **01** = 1, ..., **0A** = 10, ..., **10** = 16, ..., **FF** = 255.

d) Notations algorithmiques

Les algorithmes sont présentés sous forme de pseudo-code simple (notamment en s'affranchissant des problèmes de mémoire). Les entrées et les sorties sont toujours précisées. Les structures de contrôle classiques sont notées en gras (**tant que** condition **faire** instructions **fin tant que**, **si** condition **alors** instructions **sinon** instructions **fin si**, ...). Les commentaires dans les algorithmes sont signalés par le symbole \triangleright . Le symbole $a \leftarrow b$ indique l'assignation algorithmique (*i.e.* a prend la valeur de b) et le symbole $a \xleftarrow{u.a.} A$ l'assignation d'un élément tiré uniformément aléatoirement (*i.e.* un élément est tiré uniformément aléatoirement dans l'ensemble A et la valeur obtenue est enregistrée dans a).

CRYPTOGRAPHIE CLASSIQUE

1

La cryptologie est une science très ancienne : les hommes ont toujours eu besoin de dissimuler des informations et de transmettre des messages en toute confidentialité. Le terme cryptologie vient du grec *kruptos* signifiant *secret, caché* et de *logos* signifiant *discours*. La cryptologie est donc la *science du secret*. Elle regroupe la cryptographie et la cryptanalyse : la première a pour but de concevoir des systèmes visant à assurer la sécurité des communications sur un canal public et la seconde vise à trouver des failles dans ces systèmes.

La cryptographie est traditionnellement utilisée pour dissimuler des messages aux yeux de certains utilisateurs et le chiffrement des communications militaires a depuis l'Antiquité été une préoccupation majeure des diverses forces armées. Le *chiffrement* regroupe les techniques mises en œuvre pour brouiller la signification d'un message qui est matériellement visible. Le contenu du message ne doit alors être retrouvé que par les personnes auxquelles le message est adressé. Le chiffrement fait appel à deux processus élémentaires impliquant la transformation des lettres d'un message pour satisfaire ces propriétés : la *substitution* qui consiste à remplacer, sans en bouleverser l'ordre, les symboles d'un texte clair par d'autres symboles et la *transposition* qui repose sur le bouleversement de l'ordre des symboles (mais pas leur identité).

Dans ce chapitre, nous allons étudier des systèmes de chiffrement relativement simples qui ont été utilisés de l'Antiquité (*chiffrement de César* ou *scytale*) jusqu'au début du XX^e siècle (*chiffrement de Vernam*, *chiffrement de Hill*, *machine Enigma*).

1.1 CHIFFREMENT PAR SUBSTITUTION MONO-ALPHABÉTIQUE

Le *chiffrement par substitution* consiste à remplacer dans un message un ou plusieurs symboles par un ou plusieurs symboles (généralement du même alphabet) tout en conservant l'ordre de succession des symboles du message. Dans cette section, nous considérons le chiffrement par substitution *mono-alphabétique* qui consiste à remplacer chaque symbole individuel du message par un autre symbole de l'alphabet. Nous allons étudier les techniques de cryptanalyse permettant d'attaquer un tel système.

Elles reposent sur l'analyse des fréquences des symboles utilisés dans le texte chiffré et utilisent le fait que, dans chaque langue, certains symboles ou combinaisons de symboles apparaissent plus fréquemment que d'autres. Les systèmes de chiffrement par substitution mono-alphabétique conservent la répartition des fréquences et

Chapitre 1 • Cryptographie classique

si le message chiffré est suffisamment long, la recherche d'un symbole ayant une fréquence élevée permettra parfois de retrouver tout ou partie du message clair associé.

La fréquence d'apparition des lettres varie bien évidemment en fonction de la langue et du type de texte considérés. Pour un texte rédigé en français, nous obtenons généralement les fréquences d'apparition (en pourcentage) proches des valeurs suivantes¹ (cf. Figure 1.1) :

a	b	c	d	e	f	g	h	i	j	k	l	m
8,46	1,02	3,21	3,78	17,60	1,11	1,12	1,07	7,40	0,48	0	6,05	2,70
n	o	p	q	r	s	t	u	v	w	x	y	z
6,38	5,19	2,68	1,21	6,56	7,56	7,26	6,63	1,65	0	0,03	0,03	0,01

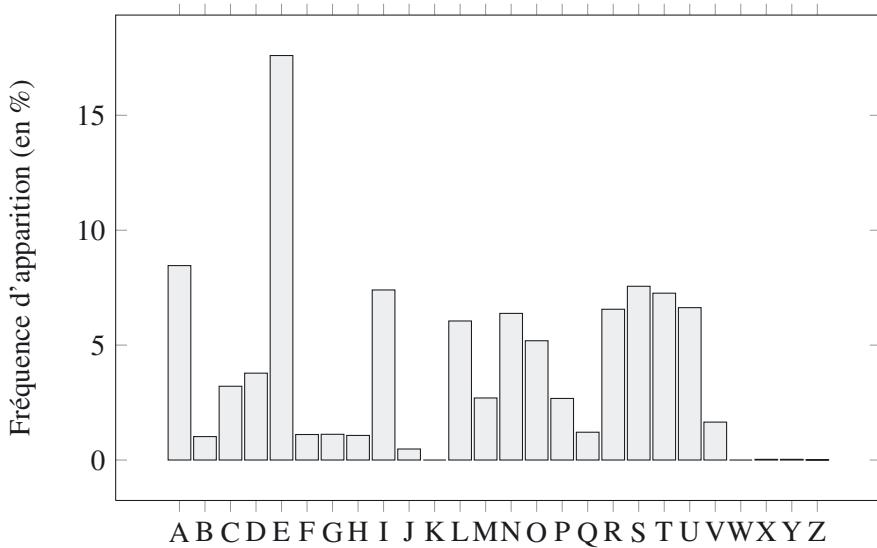


Figure 1.1 - Fréquence d'apparition des lettres en français

De même, certains couples de lettres (ou *bigrammes*) apparaissent plus souvent que d'autres dans une langue donnée. Les vingt bigrammes les plus fréquents de la langue française sont (du plus fréquent au moins fréquent) : *es, de, le, en, re, nt, on, er, te, el, an, se, et, la, ai, it, me, ou, em* et *ie*.

Le chiffrement par substitution mono-alphabétique le plus simple est le *chiffrement par décalage*, aussi connu sous le nom de *chiffrement de César*. Il consiste simplement à décaler les lettres de l'alphabet d'un nombre de positions constant vers la

1. Ces valeurs correspondent aux fréquences d'apparition des lettres dans le roman *Notre-Dame de Paris* de V. Hugo (en identifiant les lettres accentuées et les lettres non accentuées).

1.1. Chiffrement par substitution mono-alphabétique

droite ou la gauche. Par exemple, en décalant les lettres de trois rangs vers la gauche (comme le faisait J. CÉSAR), le texte clair *veni vidi vici* devient *yhql ylgl ylfl*.

Exercice 1.1 (avec programmation). Chiffrement de César

Décrypter le texte suivant qui a été obtenu en appliquant le chiffrement de César sur un texte en langue française dans lequel les espaces ont été supprimées :

```
vcfgrwqwfsbhfsntowbsobgfsbhfsnqvsnjcigsghqsoixcif  
rviwtshseicwbsgojsnjcigdogeisjcigoihfsgofhwgobgjc  
igbsrsjsnqwfqizsfrogzsgfisqzsgxcifgcijfopzsgeioj  
sqzsggwubsgrsjchfsdfctsggwcbdozfseiszsghhcbashwsf
```

Solution

Le chiffrement de César est un mode de chiffrement par substitution, il ne modifie donc pas la fréquence d'apparition des lettres. La lettre la plus fréquente dans un texte français étant le « e », le décalage entre la lettre la plus fréquente dans ce texte chiffré et la lettre « e » doit donc nous révéler la clé utilisée pour le chiffrement. La lettre qui apparaît le plus souvent dans le texte chiffré est le « s » avec 33 occurrences (puis vient la lettre « g » avec seulement 24 occurrences). Le décalage utilisé est donc vraisemblablement de 14 lettres vers la gauche et l'on obtient le message clair suivant :

```
horsdicurentrezfaineansrentrezchezvousestceaujour  
dhui fetequoinesavezvouspasquevousautresartisansvo  
usnedevezcirculerdanslesrueslesjoursouvrablesquav  
eclessignesdevotreprofessionparlequel este tonmetier
```

En ajoutant les espaces et la ponctuation, nous reconnaissons la première réplique de la pièce *Jules César* écrit par W. SHAKESPEARE en 1599 (dans la traduction de M. GUIZOT) :

« *Hors d'ici, rentrez, fainéans ; rentrez chez vous. Est-ce aujourd'hui fête ? Quoi ! Ne savez-vous pas que vous autres artisans vous ne devez circuler dans les rues les jours ouvrables qu'avec les signes de votre profession ? Parle, quel est ton métier ?* »

Le chiffrement affine est un système de chiffrement par substitution mono-alphabétique. La clé consiste en un couple d'entiers $(a, b) \in (\mathbb{Z}/26\mathbb{Z})^* \times (\mathbb{Z}/26\mathbb{Z})$. En remplaçant chaque lettre de l'alphabet par son rang (la lettre « a » est remplacée par 0, la lettre « b » est remplacée par 1, ... et la lettre « z » est remplacée par 25), une lettre du texte clair de rang $i \in \{0, \dots, 25\}$ est remplacée dans le chiffré par la lettre de rang $a \cdot i + b \bmod 26$. Puisque a est inversible dans $(\mathbb{Z}/26\mathbb{Z})$, cette transformation est bien une permutation de $(\mathbb{Z}/26\mathbb{Z})$.

Exercice 1.2 (avec programmation). Chiffrement affine

Décrypter le texte suivant qui a été obtenu en appliquant le chiffrement affine sur un texte en langue française dans lequel les espaces ont été supprimées :

ntjmpumgxpqtstgqpgtxpnchumtputgfsftgthnngxnchumwx
ootrtumhpyctgktjqtjchfooxujqhgzumxpotjxotfoqtohr
xumhzutwftgtopfmntjmpuatmfmshodpfrxpjjtqthgbxuj

Solution

Comme le chiffrement de César, le chiffrement affine ne modifie pas la fréquence d'apparition des lettres. La lettre la plus fréquente dans un texte français est le « e » et les lettres suivantes sont par ordre de fréquence le « a », le « i », le « n », le « s » et le « t » (avec des fréquences très variables d'un texte à l'autre). La lettre qui apparaît le plus souvent dans le texte chiffré est le « t » avec 19 occurrences puis vient la lettre « m » avec 12 occurrences.

En supposons que la lettre « t » correspond à la lettre « e » et que la lettre « m » correspond à l'une des lettres « a », « i », « n », « s » ou « t » (respectivement). Le couple (a, b) obtenu est testé en déchiffrant les premiers caractères du chiffré et nous obtenons les résultats suivants :

Lettre testée	« a »	« i »	« n »	« s »	« t »
ℓ	0	8	13	18	19
(a, b)	(8, 8)	(18, 0)	(21, 21)	(24, 16)	(9, 15)
Début du « clair » associé	iecamy	aegikw	iecnyz	qeysmc	cestun

La clé à utiliser pour le déchiffrement est donc vraisemblablement le couple (9, 15) et nous obtenons le message clair suivant :

cestuntroudeverdureouchanteuneriviereaccrochantfol
lementauxherbesdeshaillonsdargentoulesoleildelamon
tagnefiereluitcestunpetitvalquimoussederayons

Il s'agit bien sûr du premier quatrain du sonnet *Le Dormeur du val* écrit par A. RIMBAUD en 1870.

Les systèmes de chiffrement par substitution mono-alphabétique les plus généraux utilisent une permutation aléatoire des symboles de l'alphabet utilisé. Pour l'alphabet latin formé de 26 lettres, le nombre de clés possibles est donc égal à $26! \approx 4 \cdot 10^{26} \approx 2^{88}$. Cependant même si le nombre de clés rend toute recherche

1.1. Chiffrement par substitution mono-alphabétique

exhaustive impossible, les techniques d'analyse fréquentielle permettent de décrypter facilement un chiffré suffisamment long.

Exercice 1.3 (avec programmation).

Chiffrement par substitution mono-alphabétique

Le texte suivant résulte du chiffrement d'un texte français par une substitution mono-alphabétique.

v ubcfb osu ymoqsuu n cxqfj dqmfnu ub vjcfqu juz amqjmruz zmsscfusb bquflu
auoquz hfszbms zwfba ju wusbms qusbqu ncsz ju vmo z uddmqvcfb n uxfbuq ju
xusb wcoxc fz fj eczzc qcefnuwusb jc emqbu xfbqu no ijmv nuz wcfzmsz nu jc
xfvbmfqu ecz czzul qcefnuwusb vueusncsb emoq ueuvaug kou z usrmoddqu us
wuwu buwez kou jof os bmoqifjms nu emozzfuqu ub nu zciju
ju acjj zusbcfb ju vamo vofb ub ju xfuog bcefqz c j osu nu zuz ugbquwfbuz
osu cddfva nu vmojuoq bqme xczbu emoq vu nuejmfwusb fsbuqfuoq ubcfb
vjmouu co woq ujju quezuzusbcfb zfwejuwusb os usmqwu xfzcr u jcqru nu ejoz
n os wubqu ju xfzcr u n os amwwu n usxfqms kocqcsbu vfsk csz c j uecfz zu
wmozbcvau smfqu cog bqcfbz cvvusbouz ub iucog

hfszbms zu nfqfruc xuqz j uzvcjfuq fj ubcfb fsobfju n uzzcpuq nu
equsnqj j czvuszuoq wuwu cox wufjjuoqu uemkouz fj dmsvbfmsscfb qcquwusb
cvboujjuwusb n cfjjuoqz ju vmoqcsb ujuvbqfkou ubcfb vmoeu ncsz jc ymoqsuu
v ubcfb osu nuz wuzoquz n uvmsmwfj eqfzuz us xou nu jc zuwcfsu nu jc acfsu
zms ceecqbuwusb ubcfb co zuebfu hfszbms kof cxcfb bqusbu suod csz ub
zmoddqcfb n os ojvuqu xcqfkouog co nuzzoz nu jc vauxfjju nqmfu wmsbcfb
jusbuwusb fj z cqqubc ejozfuoqz dmfv us vauwfs emoq zu quemzuq c vackou
ecjfuq zoq osu cddfva vmmjuu co woq dcu c jc vcr u nu j czvuszuoq j
usmqwu xfzcr u xmoz dfgcfb no qurcq v ubcfb os nu vuz emqbqcfbz cqqcsruz
nu bujju zmqbu kou juz puog zuwijusb zofxqu vujof kof eczzu osu jurusnu
zmoz ju emqbqcfb nfzcfb ifr iqmbauq xmoz qurcqnu

c j fsbuqfuoq nu j ceecqbuwusb nu hfszbms osu xmfg zovquu dcfzcfb usbusnq
osu zuqfu nu smwiqz kof cxcfb bqcfb c jc eqmnovbfms nu jc dmsbu jc xmfg
eqmxuscfb n osu ejckou nu wubcj mijmsrou wfqmfq buqsu usvczbqu ncsz ju woq
nu nqmfu hfszbms bmoqsc os imobms ub jc xmfg nfwsoc nu xmjowu wcfz juz
wmbz ubcfusb usvmqu nfzbfsvbz ju zms nu j ceecqfj no bujuvqcs vmmwu ms
nfzcfb emoxcfb ubqu czzmoqnf wcfz fj s p cxcfb covos wmpus nu j ubufsnu
vmwejubuwusb hfszbms zu nfqfruc xuqz jc dusubqu fj ubcfb nu zbcboqu dquju
ejobmb eubfbu ub zc wcfrquoq ubcfb zmojfrsu ecq jc vmmifscfzms ijuou
osfdmqwu no ecqbf fj cxcfb juz vauxuog bquz ijmsnz ju xfzcr u scboqujjuwusb
zcsrofs jc euco noqvfj ecq ju zcxms rqmzzfuq juz jcwuz nu qczmfq uwmozzuuz
ub ju dqmfj nu j afxuq kof xuscfb nu equsnq dfs

Décrypter ce texte.

Solution

Le nombre d'occurrences de chaque caractère du texte chiffré est donné dans le tableau suivant :

a	b	c	d	e	f	g	h	i	j	k	l	m
17	141	150	23	48	146	11	6	13	106	14	2	95
n	o	p	q	r	s	t	u	v	w	x	y	z
71	113	4	129	20	129	0	329	51	57	35	2	128

Il est vraisemblable que le caractère « u » représente le caractère « e » dans le texte clair (nous utiliserons une fonte grasse pour indiquer les lettres appartenant au texte clair). Les autres lettres les plus fréquentes sont le « c » et le « f » mais leurs fréquences sont trop proches pour décider quels caractères elles représentent.

Les bigrammes les plus fréquents de la langue française sont « es », « de », « le », « en », « re » et « nt », et « on ». Les bigrammes commençant par « u » dans le texte chiffré ne sont pas assez fréquents pour décider quel caractère correspond à la lettre « s » dans le texte clair. Les bigrammes les plus fréquents du texte chiffré finissant par « u » sont « ju » (39 occurrences) et « nu » (35 occurrences) ce qui suggère que « j » représente « l » et « n » représente « d ». Ces trois substitutions donnent, pour le premier paragraphe, le texte suivant :

```
v ebcfb ose ymoqsee d cxqfl dqmfde eb vlcfqe lez amqlmrez zmsscfesb  
bqefle aeoqez hfszbms zwfba le wesbms qesbqe dcsz le vmo z eddmqvcfb  
d exfbeg le xesb wcoxcfz fl eczzc qcefdewesb lc emqbe xfbqee do ilmv  
dez wcfzmsz de lc xfvbmfqe ecz czzel qcefdewesb veeesdcsb emoq eweevaeq  
koe z esrmoddq es wewe bewez koe lof os bmoqifllms de emozzfeqe eb de  
zcile
```

Le mot « **do** » suggère que le caractère « o » représente « u ». Avec cette déduction, le mot « **ose** » devient « **use** » qui suggère que le caractère « s » représente « n ». Le premier paragraphe du texte devient alors :

```
v ebcfb une ymuqnee d cxqfl dqmfde eb vlcfqe lez amqlmrez zmnnncfenb  
bqefle aeuqez hfnzbmn zwfba le wenbmn qenbqe dcnz le vmu z eddmqvcfb  
d exfbeg le xenb wcuxcfz fl eczzc qcefdewenb lc emqbe xfbqee du ilmv  
dez wcfzmnz de lc xfvbmfqe ecz czzel qcefdewenb veeendcnb emuq eweevaeq  
kue z enrmuddqe en wewe bewez kue luf un bmuqifllmn de emuzzfeqe eb de  
zcile
```

Le mot « **kue** » qui apparaît deux fois suggère que la lettre « k » représente « q ». Les mots « **qcefdewenb** » et « **eb** » suggèrent que la lettre « b » représente « t ». Les mots « **luf** » et « **fl** » suggèrent que la lettre « f » représente « i » et nous obtenons :

```
v etcit une ymuqnee d cxqil dqmide et vlcique lez amqlmrez zmnnncient  
tqeile aeuqez hinztmn zwita le wentmn qentqe dcnz le vmu z eddmqvcit
```

1.1. Chiffrement par substitution mono-alphabétique

d exiteq le xent wcuxciz il eczzc qceidewent lc emqte xitqee du ilmv
dez wcizmnz de lc xivtmiqe ecz czzel qceidewent veeendcnt emuq eweevaeq
que z enrmuddqe en wewe tewez que lui un tmuqiillmn de emuzzieqe et de
zcile

Le mot « **etcit** » suggère que la lettre « c » représente « a ». Les mots « **wewe** » et « **qceidewent** » suggèrent que la lettre « w » représente « m ». Le premier paragraphe du texte chiffré devient :

v etait une ymuqnee d axqil dqmide et vlaique lez amqlmrez zmnnaient
tqeile aeuez hinzmn zmita le mentmn qentqe danz le vnu z eddmqvait
d exiteq le xent mauxaiz il eazza qaeidement la emqte xitqee du ilmv
dez maizmnz de la xivtmiqe eaz azzel qaeidement veeendant emuq emeevaeq
que z enrmuddqe en meme temez que lui un tmuqiillmn de emuzzieqe et de
zaile

Le mot « **mentmn** » suggère que la lettre « m » représente « o ». Le mot « **danz** » suggère que la lettre « z » représente « s ». Le mot « **mauxaiz** » suggère que la lettre « x » représente « v ». Avec cette déduction, le mot « **exiteq** » suggère que la lettre « q » représente « r ». L'expression « **en meme temez** » suggère que « e » représente « p ». Nous avons obtenu à ce stade la table de correspondance suivante :

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

et le texte suivant :

v etait une yournee d avril droide et vlaire les aorlores sonnaient
treile aeures hinston smita le menton rentre dans le vou s eddorvait
d eviter le vent mauvais il passa rapidement la porte vitree du ilov
des maisons de la vivtoire pas assel rapidement vependant pour empevaer
que s enrouddre en meme temps que lui un tourillon de poussiere et de
saile

le aall sentait le vaou vuit et le vieug tapis a l une de ses
egtremites une addivae de vouleur trop vaste pour ve deploiemnt
interieur etait vlouee au mur elle representait simplement un enorme
visare larre de plus d un metre le visare d un aomme d environ quarante
ving ans a l epaisse moustavae noire aug traits avventues et ieaug

hinston se dirirea vers l esvalier il etait inutile d essaper de
prendre l asveneur memo aux meilleures epoques il donvtionnait
rarement avtuellement d ailleurs le vourant elevtrique etait voupe
dans la yournee v etait une des mesures d evonomie prises en vue de la
semaine de la aaine

son appartement etait au septieme hinston qui avait trente neud ans et
souddrait d un ulvere variqueug au dessus de la vaeville droite montait

lentement il s arreta plusieurs dois en vaemin pour se reposer a vaaque palier sur une addivae vollee au mur dave a la vare de l asvenseur l enorme visare vous digait du rerard v etait un de ves portraits arranres de telle sorte que les peug semilent suivre velui qui passe une lerende sous le portrait disait iir irotaer vous rerarde

a l interieur de l appartement de hinstone voig survree daisait entendre une serie de nomires qui avaient trait a la produvtion de la donte la voig provenait d une plaque de metal oilonrue miroir terne envastre dans le mur de droite hinstone tourna un iouton et la voig diminua de volume mais les mots etaient envore distinvts le son de l appareil du televran vomme on disait pouvait etre assourdi mais il n p avait auvun mopen de l eteindre vromplettement hinstone se dirirea vers la denetre il etait de stature drele plutot petite et sa mairreur etait soulirnee par la vominaison ileue unidorme du parti il avait les vaeveug tres ilonds le visare naturellement sanruin la peau durvie par le savon rrossier les lames de rasoir emoussees et le droid de l aiver qui venait de prendre din

En terminant l’analyse de façon similaire, nous obtenons la table de correspondance :

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

et le texte complet (soit les premiers paragraphes du roman *1984* de G. ORWELL publié en 1949 dans la traduction de A. AUDIBERTI).

1.2 CHIFFREMENT PAR SUBSTITUTION POLY-ALPHABÉTIQUE

La vulnérabilité des systèmes de chiffrement par substitution mono-alphabétique a poussé les cryptologues à développer à partir du XV^e siècle des systèmes plus élaborés qui utilisent plusieurs alphabets de chiffrement. Le chiffrement par substitution *poly-alphabétique* consiste à remplacer un symbole du texte clair par un autre qui dépend de l’état du cryptosystème. Cette modification de la substitution dépend alors de la position du symbole dans le message.

L’utilisation de plusieurs alphabets pour effectuer le chiffrement a pour conséquence que des occurrences différentes d’une même lettre du texte clair ne sont pas toujours chiffrées par la même lettre et des occurrences différentes d’une même lettre dans le texte chiffré ne correspondent pas nécessairement à la même lettre du clair. En particulier, une cryptanalyse par analyse fréquentielle d’un chiffrement par sub-

1.2. Chiffrement par substitution poly-alphabétique

stitution poly-alphabétique doit utiliser des techniques plus évoluées que pour des substitutions mono-alphabétiques.

Le *chiffrement de Vigenère* est un système de substitution poly-alphabétique élaboré par B. DE VIGENÈRE en 1586. Ce procédé de chiffrement repose sur l'utilisation périodique de plusieurs alphabets de substitution déterminés par la clé (en général un mot). Pour pouvoir chiffrer un texte clair, à chaque caractère nous associons une lettre de la clé pour effectuer le décalage correspondant comme dans le chiffrement de César. Ainsi le texte clair « *vigenere* » chiffré avec la clé « *cle* » devient le chiffré « *xtkgyitp* ». En effet, la lettre *v* chiffrée avec la lettre *c* est décalée de deux positions, la lettre *i* est chiffrée avec la lettre *l* et la lettre *g* chiffrée avec la lettre *e*. Ensuite, la lettre *e* est chiffrée avec la lettre *c* et ainsi de suite de façon périodique.

En particulier, si la longueur de la clé ℓ est connue, retrouver le texte clair à partir du texte chiffré c peut se faire en appliquant une cryptanalyse du chiffrement de César (comme dans l'exercice 1.1) pour chaque sous-chiffré c_i de c formé uniquement des lettres dont les positions sont congrues à i modulo ℓ (pour $i \in \{0, \dots, \ell - 1\}$). La difficulté pour le cryptanalyste consiste donc à retrouver la longueur de la clé.

La première méthode pour déterminer la longueur de la clé est connue sous le nom de *test de Kasiski* (d'après F.W. KASISKI). Elle repose sur le fait que si deux groupes de lettres (ou *Polygrammes*) du chiffré sont égaux alors il s'agit probablement du même polygramme dans le texte clair chiffré avec la même partie de la clé. La taille de l'intervalle qui sépare ces deux polygrammes identiques dans le chiffré sera donc, dans la majorité des cas, un multiple du nombre de la longueur de la clé. S'il y a plusieurs répétitions de polygrammes, le plus grand diviseur commun des distances les séparant est très probablement la taille de clé.

Exercice 1.4 (avec programmation).

Chiffrement de Vigenère – test de Kasiski

Le texte suivant a été obtenu en appliquant le chiffrement de Vigenère sur un texte en langue française dans lequel les espaces ont été supprimées :

```
zbpuvepuqsd1zgllksousvpasfpddggaqwptdgpt zweemqzrdjtddefek  
eferdprrcyndgluaowcnbpzzrbvpsfpashpncothaeqrferdlrlw  
wertlussfikgoeuswotfdgqsyasrlnrzppdhitticfrciwurhcezrpthp  
uwiyenamrdbzyzwezelzucamrptqseqcfgdrfrhrpatsepzgfnaffisbpv  
blisrplzgnemswaqoxpdseehbeeksdptdtqsddgxurwnidbddplncsd
```

Utiliser le test de Kasiski pour déterminer la longueur de la clé utilisée et décrypter ce texte.

Solution

En cherchant les répétitions de chaînes de trois ou quatre caractères dans le texte chiffré, nous obtenons par exemple la répétition des motifs ferd, pas, ptz et ddd à distance 48, 68, 40 puis 156 et 12.

```
zbpuevpuqsd1zgllksousvpasfpddggaqwptdgptzweemqzrdjtddefek  
eferdprrcyyndgluaowcnbptzzrbvpssfpashpncoemhaeqrferdlrlw  
wertlussfikgoeuswotfdgqsyasrlnrzppdhtticfrciwrhcezrmpmhtp  
uwiyenamrdbzyzwelzucamrptzqseqcfgdrfrhrpatsepzgfnaffisbpv  
blisrplzgnemswaqoxpdseehbeeksdpdttsdddgxurwnidbdddplncsd
```

Le plus grand commun diviseur de ces distances est l'entier 4 et nous en déduisons que la clé est probablement de longueur 4. Nous obtenons les quatre sous-chiffrés c_0 , c_1 , c_2 et c_3 formés des 72 (ou 71 pour c_3) caractères de rang congru à 0, 1, 2 ou 3 modulo 4 (respectivement) :

$$\begin{aligned} c_0 &= \text{zeqzkssdqdzmdkrrdanbzscmqrlskutqsrdicrzhwndzzmzqdhtzasdszmqdhtqdrddc} \\ c_1 &= \text{bvsgsvfgwgwqjeedcgobzvfholrdwtsgfsrzchihrtabwurqcrsgfbbrgsosbsdsgwbps} \\ c_2 &= \text{ppdloppgppeztffpylpzppptaflwlflowdylptfwcppymzecpsffpeffplpnxeedtndxndl} \\ c_3 &= \text{cuulluadatterdeernuctrsaneeereueiogaptruemuerylategrapnivileapeptuidn} \end{aligned}$$

La lettre la plus fréquente dans le chiffré c_0 (*resp.* c_1 , *resp.* c_2 , *resp.* c_3) est le d (*resp.* le s, *resp.* le p, *resp.* le e). En supposant que ces lettres représentent la lettre « e » dans le texte clair, nous en déduisons que la clé utilisée pour le premier chiffré (*resp.* le second chiffré, *resp.* le troisième chiffré, *resp.* le quatrième chiffré) est probablement le z (*resp.* le o, *resp.* le l, *resp.* le a). Nous obtenons ainsi le texte clair

```
aneufheureslasalledutheatredesvarietesetaitencorevidequel  
quespersonnesaubalconetorchestreattentperduesparmi  
lesfauteuilsdeveloursgrenatdanslepetitjour dulustreademife  
uxuneombrenoyaitlagrandetacherougedurideauetpasunbruitnev  
enaitdelascenelarameeteintelespupitresdesmusiciensdebandes
```

En ajoutant les espaces et la ponctuation, nous retrouvons le texte

« À neuf heures, la salle du théâtre des Variétés était encore vide. Quelques personnes, au balcon et à l'orchestre, attendaient, perdues parmi les fauteuils de velours grenat, dans le petit jour du lustre à demi-feux. Une ombre noyait la grande tache rouge du rideau ; et pas un bruit ne venait de la scène, la rampe éteinte, les pupitres des musiciens débandés. »

qui débute le roman *Nana* écrit en 1880 par É. ZOLA.

Une méthode plus générale pour déterminer la longueur de la clé dans un système de chiffrement poly-alphabétique a été proposée par W. FRIEDMAN en 1920. Elle repose sur le calcul de l'indice de coïncidence qui détermine la probabilité de répétition des lettres dans un message chiffré. L'indice se calcule par la formule suivante :

$$I = \sum_{i=0}^{25} \frac{n_i(n_i - 1)}{n(n - 1)}$$

où n_i est le nombre de lettres de rang i dans le texte chiffré (pour $i \in \{0, \dots, 25\}$) et n la longueur du texte chiffré. Dans le cas d'un texte aléatoire (*i.e.* où les lettres sont tirées uniformément aléatoires dans l'alphabet $\{a, b, \dots, z\}$), l'indice de coïncidence est égal à 0,0385. Dans un langage structuré, l'indice de coïncidence ne dépend que de la distribution de probabilités des lettres de l'alphabet et n'est pas modifié par une substitution mono-alphabétique. En français, l'indice de coïncidence vaut environ 0,0746 et il est donc suffisamment éloigné de celui d'un texte aléatoire pour permettre de distinguer un texte chiffré par une substitution mono-alphabétique d'un texte aléatoire. Pour un chiffré produit par le chiffrement de Vigenère, si, étant donné un entier ℓ , chaque sous-chiffré – formé uniquement des lettres dont les positions sont congrues à une valeur fixée modulo ℓ – a un indice de coïncidence proche de 0,0746, alors ℓ est vraisemblablement un multiple de la longueur de la clé utilisée.

Exercice 1.5 (avec programmation).

Chiffrement de Vigenère – indice de coïncidence

Le texte suivant a été obtenu en appliquant le chiffrement de Vigenère sur un texte en langue française dans lequel les espaces ont été supprimées :

```
gmyxxocxziancxktanmyolupjrztgxwshctzluibuic
yzwxyqtvxzukibkotuxkagbknmimmzyajvjzampqyz
loinoiqknaumbknkvnkaiakgwtnilvvzvqydmvjcximr
vzkilxzqtomrgqmdjrzyazvzmmyjgkoaknkuiaivknvvy
```

Utiliser l'indice de coïncidence pour déterminer la longueur de la clé utilisée et déchiffrer ce texte.

Solution

Dans une première étape de cryptanalyse, nous calculons l'indice de coïncidence moyen obtenu à partir des sous-chiffrés en fonction de la longueur de la clé.

Longueur	1	2	3	4	5	6	7	8
Indice	0.0505	0.0517	0.0760	0.0476	0.0492	0.0760	0.0472	0.0456

Nous en déduisons que la clé est probablement de longueur 3 ce qui donne les trois sous-chiffrés c_0 , c_1 et c_2 formés des 59 caractères de rang congru à 0, 1 ou 2 modulo 3 (respectivement) :

```
c0 = gxoznkopzxhziizyvzioxgnmzjpziinmnvignvvdjivizogdzzmjonivv
c1 = mzcictmljtwclbcwqqubtkbmmyvaqlnqabkkawivqmcmlqmqjyvmgakaky
c2 = yoxaxayurgstuuuytxkkuakizajmyookuknaktlzyvxrkxtrmrasykuiny
```

La lettre la plus fréquente dans le chiffré c_0 (*resp.* c_1 , *resp.* c_2) est le **z** (*resp.* le **m**, *resp.* le **k**). En supposant que ces lettres représentent la lettre « e » dans le texte clair, nous en déduisons que la clé utilisée pour le premier chiffré (*resp.* le second chiffré, *resp.* le troisième chiffré) est probablement le **v** (*resp.* le **i**, *resp.* le **g**). Nous obtenons ainsi le texte clair

l'escriture au surplus est double la commune dont on use ordinairement et l'occulte secrète qu'on desguise d'infinies sortes, chacun selon sa fantaisie pour ne la rendre intelligible qu'entre soy et ses consachans

En ajoutant les espaces et la ponctuation, nous retrouvons la réflexion

« L'écriture au surplus est double ; la commune dont on use ordinairement ; et l'occulte secrète, qu'on desguise d'infinies sortes, chacun selon sa fantaisie pour ne la rendre intelligible qu'entre soy et ses consacchans. »

issue du *Traité des chiffres ou secrètes manières d'écrire* écrit par B. DE VIGENÈRE en 1585.

Dans les deux exercices suivants, nous allons considérer un système de chiffrement proposé par L.S. Hill en 1931. Il s'agit d'un chiffrement par substitution poly-alphabétique où l'on (dé)chiffre les lettres par paquets et non pas les unes après les autres. Il permet ainsi de rendre plus difficile l'analyse fréquentielle du système. Le *chiffrement de Hill* est une extension du chiffrement affine à un bloc. Comme pour le chiffrement affine, les lettres sont remplacées par leur rang dans l'alphabet (la lettre a est remplacée par 0, la lettre b est remplacée par 1, ... et la lettre z est remplacée par 25). La clé est une matrice carrée inversible M de taille ℓ dans $(\mathbb{Z}/26\mathbb{Z})$ et le (dé)chiffrement opère sur des blocs de ℓ caractères. Le chiffré correspondant à un texte clair $(m_1, \dots, m_\ell) \in (\mathbb{Z}/26\mathbb{Z})^\ell$ est le vecteur $(c_1, \dots, c_\ell) \in (\mathbb{Z}/26\mathbb{Z})^\ell$ obtenu par multiplication matricielle :

$$M \cdot \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_\ell \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_\ell \end{bmatrix}$$

Le déchiffrement s'effectue naturellement en multipliant le chiffré par l'inverse de la matrice M .

Exercice 1.6 Chiffrement de Hill – nombre de clés

Soient $\ell \geq 1$ un entier. Donner le nombre de clés différentes pour le chiffrement de Hill qui opère sur des blocs de taille ℓ . Donner la taille des clés en bits pour $\ell \in \{1, 2, 3, 4, 5, 6\}$.

Solution

Le nombre de matrices carrées inversibles de taille ℓ à coefficient dans $(\mathbb{Z}/26\mathbb{Z})$ peut être calculé en utilisant le théorème chinois des restes. En effet, une telle matrice est inversible si et seulement si elle est inversible modulo 2 et modulo 13 (et réciproquement étant donné un couple de matrices inversibles de $\mathcal{M}_\ell(\mathbb{Z}/2\mathbb{Z}) \times \mathcal{M}_\ell(\mathbb{Z}/13\mathbb{Z})$ nous pouvons construire une

1.2. Chiffrement par substitution poly-alphabétique

matrice inversible dans $\mathcal{M}_\ell(\mathbb{Z}/26\mathbb{Z})$). Le nombre de matrices inversibles de $\mathcal{M}_\ell(\mathbb{Z}/p\mathbb{Z})$ pour un nombre premier p est égal à

$$(p^\ell - 1) \cdots (p^\ell - p^{\ell-1})(p^\ell - p^{\ell-1})$$

En effet, la première colonne de la matrice peut être choisie comme un vecteur non nul de $(\mathbb{Z}/p\mathbb{Z})^\ell$ et il y a donc $p^\ell - 1$ choix possibles. La n -ième colonne (pour $n \in \{2, \dots, \ell\}$) peut être choisie en dehors de l'espace vectoriel engendré par les $(n - 1)$ premières colonnes et il y a donc $p^\ell - p^{n-1}$.

Par conséquent le nombre de clés possibles pour le chiffrement de Hill est égal à

$$\prod_{i=0}^{\ell-1} (2^\ell - 2^i)(13^\ell - 13^i)$$

En particulier, nous obtenons pour $\ell \in \{1, 2, 3, 4, 5, 6\}$, les tailles de clés (en bits) suivantes :

ℓ	1	2	3	4	5	6
Taille de clé (en bits)	4	18	41	74	116	168

Le chiffrement de Hill est linéaire ; l'exercice suivant montre que cette propriété le rend complètement vulnérable à une attaque à un clair connu.

Exercice 1.7 Chiffrement de Hill – attaque à clair connu

Décrypter le texte suivant qui a été obtenu en appliquant le chiffrement de Hill sur des blocs de taille 2 sur un mot de la langue française :

gzatzxjihvbreosu

sachant que le chiffrement du mot **chiffrer** avec la même clé donne le chiffré **jvfrtqnb**.

Solution

Posons

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathcal{M}_2(\mathbb{Z}/26\mathbb{Z})$$

la matrice à utiliser pour le déchiffrement. La matrice M transforme le bigramme **ju** en le bigramme **ch** et le bigramme **fr** en le bigramme **if**. Nous avons donc (en remplaçant les caractères par leur rang dans l'alphabet) l'équation matricielle :

$$M \cdot \begin{bmatrix} 9 & 5 \\ 21 & 17 \end{bmatrix} = \begin{bmatrix} 2 & 8 \\ 7 & 5 \end{bmatrix}$$

La matrice multipliée à M n'est pas inversible dans $\mathcal{M}_4(\mathbb{Z}/26\mathbb{Z})$ (son déterminant est pair et donc non inversible dans $(\mathbb{Z}/26\mathbb{Z})$), nous considérons donc le bigramme suivant : **tq** qui doit être déchiffré en **fr** et nous obtenons la nouvelle équation

$$M \cdot \begin{bmatrix} 9 & 19 \\ 21 & 16 \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 7 & 17 \end{bmatrix}$$

soit

$$\begin{aligned} M &= \begin{bmatrix} 2 & 5 \\ 7 & 17 \end{bmatrix} \cdot \begin{bmatrix} 9 & 19 \\ 21 & 16 \end{bmatrix}^{-1} \\ &= (5^{-1} \bmod 26) \begin{bmatrix} 2 & 5 \\ 7 & 17 \end{bmatrix} \cdot \begin{bmatrix} 16 & -19 \\ -21 & 9 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 17 \\ 3 & 4 \end{bmatrix} \end{aligned}$$

Le déchiffrement du bigramme **gz** donne alors

$$\begin{bmatrix} 1 & 17 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 6 \\ 25 \end{bmatrix} = \begin{bmatrix} 15 \\ 14 \end{bmatrix}$$

soit le bigramme **po** et le déchiffrement complet de **gzatzxjihvbreosu** nous donne le texte clair : polyalphabetique.

1.3 CHIFFREMENT PAR TRANSPOSITION

Un *chiffrement par transposition* est un système de chiffrement qui consiste à bouleverser l'ordre des données à chiffrer (de façon à les rendre incompréhensibles) sans pour autant remplacer les lettres du message par d'autres lettres ou symboles. Il s'agit généralement de réordonner géométriquement les données pour les rendre visuellement inexploitables. Comme un chiffrement par transposition ne modifie pas les lettres du message clair, le chiffré aura exactement la même fréquence de lettres que le texte clair original.

La *scytale* est un exemple de chiffrement par transposition et est considérée comme le plus ancien dispositif de cryptographie militaire connue, puisqu'elle a été utilisée à partir de 600 ans avant J.-C. en Grèce. Son principe consiste à enruler une bande de papyrus ou de cuir sur un cylindre appelé *scytale* ou *bâton de Plutarque*, et sur laquelle est écrit longitudinalement (en plaçant une lettre sur chaque circonvolution) le message à transmettre. Le texte, une fois déroulé, n'est plus compréhensible. Pour le déchiffrer, le destinataire devait posséder un cylindre d'un diamètre identique à

celui utilisé pour l'opération de chiffrement. Il lui suffit d'enrouler la scytale autour de ce bâton pour retrouver le texte clair.

La figure 1.2 illustre le fonctionnement de la scytale.

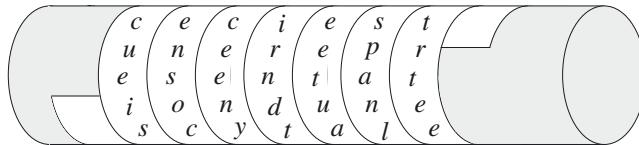


Figure 1.2 – Illustration du fonctionnement d'une scytale

- À sa surface, on place un papier enroulé sur lequel est écrit le message :

ceciestunerepresentationdunescytale

- Lorsque le papier est déroulé, les lettres sont dans le désordre et le message devient :

cueisensocceenyirndteetuaspanltrtee

Exercice 1.8 (avec programmation). Scytale

Décrypter le texte suivant qui a été obtenu en appliquant un chiffrement par transposition par scytale sur un texte en langue française dans lequel les espaces ont été supprimées :

**lelnracrsrunanatuvllerrmcnjeeaeseiaetancsagsgeemftqdnne
ntraraueneciliianeredofaesntdneenignpcdaishdcaoaeenede**

Solution

Pour retrouver le texte original, il suffit de deviner le nombre k de lettres qui apparaissent sur une circonférence de la scytale. Une fois cette valeur k trouvée, il suffit d'écrire le texte dans une grille rectangulaire formée de k lignes et $\lceil 109/k \rceil$ colonnes (le texte chiffré faisant 109 caractères). Nous pouvons donc écrire le texte dans des grilles de taille 1×109 , 2×55 , 3×37 , 4×28 , 5×22 , 6×19 , 7×16 , 8×14 , 9×13 , 10×11 ou leurs transposées. Par exemple, si le nombre k est égal à 7, nous obtenons le réarrangement 7×16 suivant :

l	s	t	r	a	t	g	t	t	n	a	f	n	p	d	n
e	r	u	m	e	a	s	q	r	e	n	a	e	c	c	e
l	u	v	c	s	n	g	d	a	c	e	e	d	a	d	
n	n	l	n	e	c	e	n	r	i	r	s	n	a	o	e
r	a	l	j	i	t	e	n	a	l	e	n	i	i	a	
a	n	e	e	a	s	m	e	u	i	d	t	g	s	e	
c	a	r	e	e	a	f	n	e	i	o	d	n	h	e	

En testant successivement le déchiffrement pour toutes les valeurs de $k \in \{2, \dots, 11\}$, nous obtenons finalement avec $k = 11$:

l	a	m	a	g	n	i	f	i	c
e	n	c	e	e	t	l	a	g	a
l	a	n	t	e	r	i	e	n	o
n	t	j	a	m	a	i	s	p	a
r	u	e	n	f	r	a	n	c	e
a	v	e	c	t	a	n	t	d	e
c	l	a	t	q	u	e	d	a	n
s	l	e	s	d	e	r	n	i	e
r	e	s	a	n	n	e	e	s	d
u	r	e	g	n	e	d	e	h	e
n	r	i	s	e	c	o	n	d	

soit la phrase

la magnificence et la galanterie n'ont jamais parue en France avec
tant de clarté dans les dernières années du règne de Henri II, second

qui débute le roman *La Princesse de Clèves* attribué à MME DE LA FAYETTE en 1678.

Comme avec le chiffrement par scytale, dans un *chiffrement par transposition par colonnes*, le message à chiffrer est écrit dans une grille rectangulaire formée de lignes de même longueur et il est lu par colonnes. La différence est que les colonnes sont choisies dans le désordre (la permutation choisie étant la clé de chiffrement). Dans un chiffrement par transposition par colonnes, si le message est trop court pour remplir la grille, les espaces vides restants sont remplis avec des lettres aléatoires.

Exercice 1.9 (avec programmation).

Chiffrement par transposition par colonnes

Déchiffrer le texte suivant qui a été obtenu en appliquant un chiffrement par transposition par colonnes sur un texte en langue française dans lequel les espaces ont été supprimées (en sachant que la clé utilisée est une permutation de longueur 8) :

ahcaaieqreeiecadapniieliouuxiusnlbocoretcilia
uintsefesetdletvseeeedeaunnsuuivntshnenlttvtl
ydsrttonsasruronndiicneijttestjliaeesaoleddrev
lriaaimxrpseruleaereautvorqhidoelutssglaneeslh

Solution

En réorganisant le texte en huit colonnes, nous obtenons les lignes suivantes :

	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8	
1	a	i	u	a	y	j	l	o		13	e	o	l	s	t	e	u	s
2	h	u	i	u	d	t	r	r		14	c	c	e	h	o	s	l	s
3	c	o	n	e	s	t	i	q		15	a	o	t	n	n	a	n	g
4	a	u	t	n	r	e	a	h		16	d	r	v	e	n	o	a	l
5	a	u	s	n	t	s	i	i		17	a	e	s	n	d	l	e	a
6	i	x	e	s	o	t	m	d		18	p	t	e	l	i	e	r	n
7	e	i	f	u	n	j	x	o		19	n	c	e	t	i	d	e	e
8	q	u	e	u	s	l	r	e		20	i	l	e	t	c	d	a	e
9	r	s	s	i	a	i	p	e		21	i	l	e	v	n	r	u	s
10	e	n	e	v	s	i	s	l		22	e	i	d	t	e	e	t	l
11	e	l	t	n	r	a	e	u		23	l	a	e	l	i	v	v	h
12	i	b	d	t	u	e	r	t										

et pour trouver la clé utilisée, il suffit de trouver une anagramme d'une ligne qui a un sens en français. En faisant une recherche informatique nous ne trouvons pas d'anagramme de neuf lettres par contre, nous trouvons que la ligne 21 contient une anagramme de huit lettres : *univers* laissant la lettre 1. En faisant la supposition que cette ligne correspond à *l'univers*, nous trouvons que la permutation à appliquée aux colonnes pour retrouver le texte clair est

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 1 & 6 & 5 & 3 & 7 & 2 & 8 \end{pmatrix} = (1 \ 4 \ 5 \ 3 \ 6 \ 7 \ 2)$$

La clé utilisée pour le chiffrement était donc $\sigma^{-1} = (1 \ 2 \ 7 \ 6 \ 3 \ 5 \ 4)$. En ajoutant les espaces et la ponctuation, nous obtenons le texte :

« *Il y a aujourd'hui trois cent quarante-huit ans six mois et dix-neuf jours que les Parisiens s'éveillèrent au bruit de toutes les cloches sonnant à grande volée dans la triple enceinte de la Cité, de l'Université et de la Ville.* »

(suivie des lettres vh), soit la première phrase du roman *Notre-Dame de Paris* écrit par V. HUGO en 1831.

1.4 CHIFFREMENT PARFAIT

Un système de chiffrement est dit *parfait* si la connaissance d'un message chiffré n'apporte aucune information sur le message clair même à un adversaire ayant des ressources calculatoires illimitées. Plus formellement, en notant M la variable aléatoire correspondant au texte clair et C la variable aléatoire correspondant au texte chiffré, un système de chiffrement est dit parfait si

$$\Pr[M = m | C = c] = \Pr[M = m]$$

pour tout couple clair/chiffré (m, c) (ou de façon équivalente si $\Pr[M = m, C = c] = \Pr[M = m] \cdot \Pr[C = c]$ pour tout couple clair/chiffré (m, c)).

Exercice 1.10 Carré latin

Soit $n > 0$ un entier. Un *carré latin* de rang n est un tableau de taille $n \times n$ contenant les entiers $\{1, \dots, n\}$ tel que chacun de ces n entiers apparaît une fois sur chaque ligne et chaque colonne.

1. Donner un exemple de carré latin de rang 5.

Étant donné un carré latin T de rang n , on lui associe un système de chiffrement où l'espace des clairs, des clés et des chiffrés est l'ensemble $\{1, \dots, n\}$ et le chiffrement du clair $m \in \{1, \dots, n\}$ avec la clé $k \in \{1, \dots, n\}$ est l'élément $T(k, m)$ placé à l'intersection de la k -ième ligne de la m -ième colonne.

2. Montrer que ce cryptosystème est un chiffrement parfait.

Solution

1. Il suffit de remarquer que la table de multiplication d'un groupe fini d'ordre n est nécessairement un carré latin de taille n (la réciproque est fausse en général). En particulier, pour construire un carré latin de taille 5, il suffit de considérer un groupe d'ordre 5. Il n'existe qu'un tel groupe à isomorphisme près et il s'agit d'un groupe $(\mathbb{Z}/5\mathbb{Z})^*$. Nous obtenons ainsi le carré latin suivant :

1	2	3	4	5
2	3	4	5	1
3	4	5	1	2
4	5	1	2	3
5	1	2	3	4

et toute permutation de ce carré latin convient également.

2. Soit M la variable aléatoire correspondant au texte clair, soit C la variable aléatoire correspondant au texte chiffré et soit K la variable aléatoire correspondant à la clé. Nous avons

$$\begin{aligned}\Pr[M = m, C = c] &= \sum_{k=1}^n \Pr[M = m, C = c \mid K = k] \cdot \Pr[K = k] \\ &= \frac{1}{n} \sum_{k=1}^n \Pr[M = m, C = c \mid K = k]\end{aligned}$$

puisque la clé k est uniformément distribuée. Il nous reste à calculer la probabilité $\Pr[M = m, C = c \mid K = k]$ pour toute clé $k \in \{1, \dots, n\}$. Comme pour une clé donnée et un texte clair donné, il n'existe qu'un texte chiffré associé, nous avons

$$\Pr[M = m, C = c \mid K = k] = \begin{cases} \Pr[M = m] & \text{si } c = T(k, m) \\ 0 & \text{sinon} \end{cases}$$

Puisque M et K sont des variables aléatoires indépendantes, nous avons

$$\Pr[M = m, C = c] = \frac{1}{n} \sum_{k=1}^n \Pr[M = m] \cdot \text{ind}(c, k, m)$$

où $\text{ind}(c, k, m)$ est la fonction booléenne qui vaut 1 si $c = T(k, m)$ et 0 sinon. Nous obtenons

$$\Pr[M = m, C = c] = \frac{\Pr[M = m]}{n} \sum_{k=1}^n \text{ind}(c, k, m) = \frac{\Pr[M = m]}{n}$$

car dans le carré latin T chaque chiffré apparaît une fois et une seule dans la colonne d'indice n . Par ailleurs, nous avons

$$\Pr[C = c] = \sum_{m=1}^n \Pr[M = m, C = c] = \sum_{m=1}^n \frac{\Pr[M = m]}{n} = \frac{1}{n}$$

et finalement

$$\Pr[M = m, C = c] = \Pr[M = m] \cdot \Pr[C = c]$$

pour tout couple $(m, c) \in \{1, \dots, n\}^2$.

Le système de *chiffrement de Vernam*, également appelé chiffrement par *masque jetable*, a été inventé en 1918 par G. VERNAM. Dans ce système de chiffrement sur un alphabet à n éléments, la clé de chiffrement est une suite de nombres aléatoires et indépendants, compris entre 0 et $n - 1$. Le message chiffré est obtenu comme dans le chiffrement de Vigenère en décalant chaque lettre du message clair par le nombre de positions donné par la clé. Pour que ce système de chiffrement soit parfait, il faut que la clé soit une suite de caractères au moins aussi longue que le message à chiffrer, qu'elle soit choisie de façon uniformément aléatoire et qu'elle ne soit utilisée qu'une seule fois. C. SHANNON a montré en 1949 que ces conditions sont nécessaires pour obtenir une sécurité parfaite.

Le risque que fait courir la réutilisation d'une clé est facile à montrer. Supposons que deux messages binaires m_1 et m_2 de longueur n sont chiffrés par le système de Vernam en utilisant la même clé k . Nous avons les chiffrés c_1 et c_2 définis par

$$c_1 = m_1 \oplus k \text{ et } c_2 = m_2 \oplus k$$

où l'opérateur \oplus désigne le « ou exclusif » bit-à-bit des chaînes de longueur n . En appliquant l'opération \oplus à c_1 et c_2 , un adversaire obtient

$$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$$

Il obtient ainsi le « ou exclusif » bit-à-bit des deux textes clairs en supprimant l'effet de masquage opéré par la clé. En particulier, si l'adversaire connaît l'un des messages

clairs m_1 , il peut trouver instantanément le second en calculant $c_1 \oplus c_2 \oplus m_1 = m_2$. Dans les deux exercices suivants, nous allons voir que, même sans connaître l'un des clairs, des méthodes plus complexes permettent souvent de retrouver m_1 et m_2 .

Exercice 1.11 (avec programmation).

Mauvaise utilisation du chiffrement jetable

Un utilisateur a chiffré deux mots (non accentués) de la langue française de sept lettres avec le chiffrement de Vernam mais il a été imprudent et a utilisé deux fois la même clé pour chiffrer ces deux messages. Sachant que les chiffrés obtenus sont les mots hqdtmap et onooiup, faire une recherche informatique dans un *corpus* de la langue française et trouver tous les couples de textes clairs susceptibles de produire ces chiffrés.

Solution

Notons $c_1 = \text{hqdtmap}$ et $c_2 = \text{onooiup}$ les deux chiffrés et m_1 et m_2 les deux textes clairs correspondants. En soustrayant ces deux chiffrés caractère par caractère (modulo 26), nous obtenons

$$m_1 - m_2 = c_1 - c_2 = \text{tdpfega},$$

et il suffit de rechercher parmi tous les couples de mots (non accentués) de la langue française, ceux dont la différence est égale à tdpfega. Une recherche informatique parmi les 8796 mots de sept lettres non accentués du dictionnaire french de la distribution Ubuntu 10.04.2 fournit un seul couple possible :

$$m_1 = \text{chiffre} \text{ et } m_2 = \text{jetable}$$

et la clé k est, dans ce cas, égale à fjvohjl.

Dans le prochain exercice, nous allons étudier un algorithme (dû à A.J. VITERBI) pour effectuer une estimation bayésienne. Il permet notamment d'obtenir de l'information sur des messages chiffrés par le chiffrement de Vernam lorsque la même clé est utilisée plusieurs fois.

Problème 1.12 Algorithme de Viterbi

Considérons un graphe orienté pondéré $G = (S, A)$ où les poids des arcs appartiennent à l'intervalle $[0, 1]$ et la somme des poids des arcs sortants d'un sommet de S est égale à 1 (*i.e.* les poids des arcs sont des *probabilités* de transition).

Nous supposons également que les arcs sont étiquetés par des symboles d'un alphabet Σ (plusieurs arcs peuvent avoir la même étiquette). L'étiquette d'un chemin est simplement la concaténation des arcs le constituant (dans l'ordre de parcours).

1. Décrire un algorithme de programmation dynamique efficace qui étant donnés un graphe orienté $G = (S, A)$ à arcs étiquetés, un sommet distingué v_0 et une suite finie $s = (\sigma_1, \dots, \sigma_k)$ de caractères de Σ détermine s'il existe un chemin de G qui commence à v_0 étiqueté par s .
2. Modifier cet algorithme pour qu'il retourne un tel chemin s'il existe.
3. Modifier cet algorithme pour que le chemin retourné soit un chemin le plus probable partant de v_0 et ayant l'étiquette s .
4. Analyser le temps d'exécution de l'algorithme.

Solution

1. Pour une approche dynamique du problème, il suffit de considérer le sous-problème correspondant : « pour tout nœud $v_i \in S$, existe-t'il un chemin étiqueté de v_0 à v_i avec $\sigma_1 \dots \sigma_j$? »

Notons $S = \{v_0, v_1, \dots, v_{n-1}\}$ et $S(i, j)$ la variable booléenne qui prend la valeur 1 s'il existe un chemin de v_0 à v_i étiqueté par le chemin $\sigma_1 \dots \sigma_j$ (et 0 sinon). Nous avons :

$$\begin{aligned} S(0, 0) &= 1 \\ S(i, 0) &= 0 \text{ pour } 1 \leq i \leq n - 1 \\ S(i, j) &= \bigvee_{\{k, (v_k, v_i) \in A \wedge \sigma(v_k, v_i) = \sigma_j\}} S(k, j - 1) \end{aligned}$$

Une approche dynamique pour calculer les valeurs de $S(i, j)$ nécessite un ordre pour calculer ces valeurs. Les valeurs initiales sont dans la première colonne et si l'on connaît les valeurs $S(i, j_0)$ pour tout entier $i \in \{1, \dots, n - 1\}$, nous pouvons calculer toutes les valeurs $S(i, j_0 + 1)$ pour tout entier $i \in \{1, \dots, n - 1\}$. Il suffit donc de remplir la table $S(i, j)$ colonne par colonne de la gauche vers la droite.

La réponse que doit retourner l'algorithme est simplement

$$\bigvee_{i=0}^{n-1} S(i, k)$$

2. Pour retourner un chemin avec la méthode précédente, il suffit de stocker dans la table $S(i, j)$ le numéro de sommet correspondant. Si l'on suppose que $S(i, j) = \ell$ signifie qu'il existe un chemin de v_0 à v_i étiqueté par $\sigma_1 \dots \sigma_j$ dont le dernier arc est (v_ℓ, v_i) . La récurrence est identique à celle de la question précédente sauf qu'au lieu de prendre le « ou » logique des valeurs $S(i, j)$, on choisira simplement un élément k de l'ensemble $\{k, (v_k, v_i) \in A \wedge \sigma(v_k, v_i) = \sigma_j\}$ tel que $S(k, j - 1) \neq 0$. Pour retourner le chemin, s'il existe, il suffira enfin de reconstruire le chemin en partant de la fin si $S(j, k) \neq 0$ pour un $j \in \{0, \dots, n - 1\}$.
3. Pour résoudre le problème par une approche dynamique, les sous-problèmes correspondants sont identiques à ceux des questions précédentes. Cependant, au lieu de stocker des valeurs booléennes ou entières, il faut conserver un réel correspondant à la

probabilité du chemin menant de v_0 à v_i étiqueté par $\sigma_1 \dots \sigma_j$. Si la probabilité est 0, cela signifie qu'un tel chemin n'existe pas. Pour extraire le chemin de probabilité maximale (s'il existe) à la fin de l'algorithme, nous ne pouvons pas appliquer l'astuce de la question précédente et nous devons utiliser une deuxième table pour conserver l'arc qui donne la probabilité maximale de transition. Le problème devient le calcul des valeurs définies par

$$S(0, 0) = 1$$

$$S(i, 0) = 0 \text{ pour } 1 \leq i \leq n - 1$$

$$S(i, j) = \max_{\{k, (v_k, v_i) \in A \wedge \sigma(v_k, v_i) = \sigma_j\}} S(k, j - 1) \cdot p(v_k, v_i)$$

$$R(i, j) = \operatorname{argmax}_{\{k, (v_k, v_i) \in A \wedge \sigma(v_k, v_i) = \sigma_j\}} S(k, j - 1) \cdot p(v_k, v_i)$$

et l'algorithme retourne le chemin pour lequel la valeur $S(j, k)$ est maximale (pour $j \in \{0, \dots, n - 1\}$) si une telle valeur strictement positive existe.

- La complexité de l'algorithme est facile à calculer : son temps d'exécution est majoré par la taille de la table multipliée par le temps nécessaire pour le remplissage d'une case de la table.

La taille de la table est nk où n est le nombre de sommets et k la longueur de l'étiquette du chemin. Le coût pour remplir une cellule $S(i, j)$ de la table dépend du nombre d'arcs qui arrivent au sommet v_i ; ce nombre est majoré par le nombre total d'arcs a . En réalité en effectuant une analyse amortie, on voit que dans le remplissage de la table, on traite chaque arc exactement une fois par colonne. Le coût de remplissage d'une colonne est donc $O(n + a)$ et le coût de remplissage de toute la table est $O((n + a)k)$.

Pour appliquer l'algorithme de Viterbi afin de retrouver des textes chiffrés c_1 et c_2 chiffrés avec la même clé k suivant le chiffrement de Vernam, il faut donc modéliser les langages dans lesquels les messages ont été rédigés sous la forme d'un graphe. Étant donnés ces modèles, l'algorithme de Viterbi retournera le déchiffrement à maximum de vraisemblance m_1 et m_2 des chiffrés c_1 et c_2 et le chemin à probabilité maximale dans le modèle du langage tel que $m_1 \oplus k = c_1$ et $m_2 \oplus k = c_2$. La difficulté principale est la modélisation des langages. L'approche qui consiste à modéliser le langage par les fréquences d'apparition de lettres, de bigrammes et de trigrammes donne des résultats partiels intéressants.

1.5 LA MACHINE ENIGMA

La période de l'entre-deux-guerres voit le début de la mécanisation de la cryptographie (cette période est parfois appelée l'*âge technique* de la cryptographie). Le prin-

cipe du chiffrement repose toujours sur des substitutions et des permutations mais les met en œuvre *via* des machines mécaniques ou électromécaniques.

La plus célèbre de ces machines est la *machine Enigma* inventée par A. SCHERBIUS et qui a été utilisée par l'armée allemande du début des années trente jusqu'à la fin de la seconde guerre mondiale. Il s'agit d'une machine électromécanique portable représentée sur la figure 1.3.

La partie mécanique de la machine Enigma est composée d'un clavier, d'un jeu de disques rotatifs adjacents appelés *rotors* arrangés le long d'un axe et d'un mécanisme provoquant la rotation d'un ou plusieurs des rotors à chaque fois qu'une touche est pressée. Quand on appuie sur une touche du clavier, un circuit électrique est fermé et une lampe s'allume pour indiquer quelle lettre doit être substituée. Le circuit électrique est constitué de plusieurs éléments en chaîne :

- Le *tableau de connexions* qui permet d'échanger des paires de lettres au moyen de *fiches*. Dans la version initiale de la machine Enigma, il y a six fiches qui permettent donc d'échanger douze lettres.
- Les *rotors* qui sont des permutations fixées de l'alphabet. La machine Enigma disposera, au gré de ses évolutions successives, de trois à six rotors qui sont associés pour former le circuit électrique. Parmi ces rotors, seuls trois sont utilisés pour les opérations de (dé)chiffrement et l'ordre dans lequel on les place constitue une partie de la clé.

Les rotors sont cylindriques et peuvent tourner autour de leur axe. À chaque fois que l'utilisateur tape une lettre, le premier rotor tourne d'un cran. Après 26 lettres, il est revenu à sa position initiale, et le second rotor tourne alors d'un cran et de même quand le second rotor a retrouvé sa position initiale, c'est le troisième rotor qui tourne d'un cran. Le mouvement continu des rotors permet l'obtention de transformations cryptographiques différentes à chaque pression sur une touche (faisant d'Enigma un système de chiffrement par substitution poly-alphabétique). La clé donne la position de départ des rotors (en plus de la configuration des fiches sur le tableau de connexions).

- Le *réflecteur* qui est situé après les trois rotors et a pour but d'effectuer une dernière permutation. Il permet de faire revenir en arrière le « signal » en retraversant les rotors et le tableau de connexions avant d'allumer la lampe correspondant à la lettre chiffrée. Le réflecteur est le produit de transpositions :

$$(A \ Y)(B \ R)(C \ U)(D \ H)(E \ Q)(F \ S)(G \ L)(I \ P)(J \ X)(K \ N)(M \ O)(V \ W)(T \ Z)$$

L'intérêt du réflecteur vient du fait que pour déchiffrer une lettre, il suffit de remettre la machine dans sa configuration initiale et d'appuyer sur le clavier pour que la lettre correspondante du texte clair s'allume. Les opérations de chiffrement et de déchiffrement sont donc parfaitement identiques.

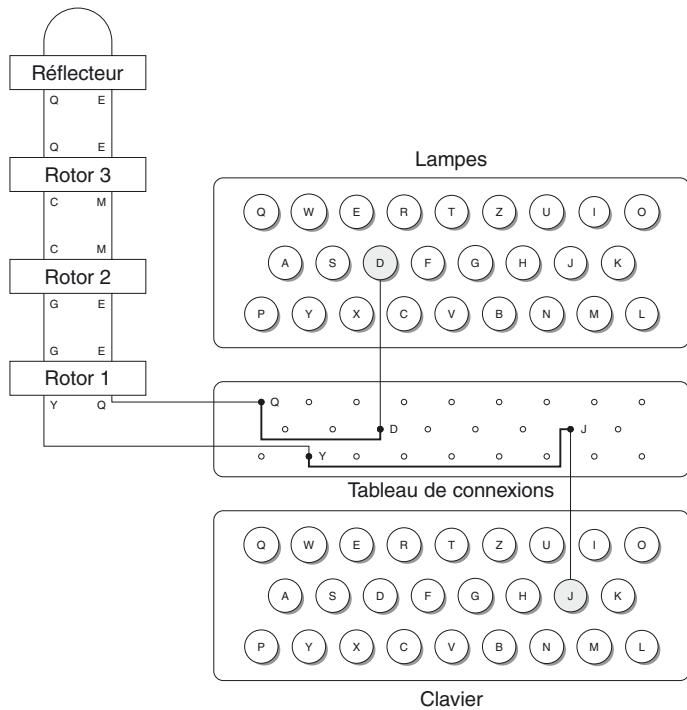


Figure 1.3 - Description schématique de Enigma

La figure 1.3 montre l'exemple du fonctionnement de la machine Enigma lorsque l'utilisateur appuie sur la lettre « J ». Le courant électrique traverse le tableau de connexions où une fiche relie la lettre « J » à la lettre « Y » ; en passant à travers les rotors, cette lettre est transformée successivement en un « G », un « C », puis un « Q ». À travers le réflecteur, le « Q » devient un « E » et en traversant les rotors dans le sens inverse, il se transforme en un « M », un « E » puis un « Q ». Enfin, le courant traverse le tableau de connexions et une fiche transforme la lettre en « D » qui s'allume.

Exercice 1.13 Enigma – Nombre de clés

Calculer le nombre de clés possibles de la machine Enigma qui utilise six fiches et cinq rotors.

Solution

Une clé de la machine Enigma consiste en les informations suivantes :

- Le choix des trois rotors parmi les cinq rotors possibles. Il y a 5 choix pour le premier, 4 pour le deuxième et 3 pour le troisième ce qui donne 60 possibilités.

- Le choix des positions initiales des rotors. Chaque rotor peut prendre 26 positions, ce qui donne 26^3 possibilités différentes.
- Le choix des positions des six fiches sur le tableau de connexions. Pour chaque choix, il faut déterminer les 14 lettres qui ne sont pas modifiées par le tableau de connexions et les 12 qui sont liées par des fiches. Le nombre de façons de grouper ces 12 lettres par paire est égal à $(12!)/(6!2^6)$. Il y a en effet 12! permutations des 12 lettres mais parmi celles-ci modifier l'ordre des paires ou l'ordre des lettres dans chaque paire produit la même configuration.

Nous obtenons donc

$$60 \cdot 26^3 \cdot \binom{26}{14} \cdot \frac{12!}{6! \cdot 2^6} \simeq 1,06 \cdot 10^{17} \approx 2^{56.5} \text{ clés possibles.}$$

Si l'ordre des rotors et les positions des rotors sont connus de l'attaquant, la machine Enigma effectue essentiellement un chiffrement par substitution mono-alphabétique. Il peut bien sûr s'attaquer comme dans l'exercice 1.3 mais comme la substitution est très particulière (certaines lettres ne sont pas modifiées et les autres le sont par couple), il est possible de trouver la substitution par approximations successives. H. WILLIAMS a proposé en 2000 d'appliquer une séquence de transpositions (à supports disjoints) au texte chiffré en prenant à chaque fois la transposition qui minimise l'*indice de Sinkov* défini comme

$$\left| \sum_{i=0}^{25} n_i \ln p_i \right|$$

où n_i désigne le nombre d'occurrences de la lettre de rang i dans le texte et p_i la fréquence de cette lettre dans un corpus de référence. Intuitivement, cet indice estime la « vraisemblance » du texte clair obtenu.

Exercice 1.14 (avec programmation). Enigma – Tableau de connexions

Le texte suivant a été obtenu en appliquant une substitution mono-alphabétique égale à un produit de six transpositions sur un texte en langue française dans lequel les espaces ont été supprimées :

epitrmcorympkzkrknpryrmamrcjmtmnpitmnzonymcknfroepgme
 pitrmmrnprdadprlodmrkprllmcjmikitintpamkarmzcmplpngpgmjm
 yonhokreonzimkrdkcorympkqkmvokzmtmzholiqkmvokzemzmeyle
 msympkzpnzemntirzivotrmrprpepgmzmrpaortmpvotrmaalkepgmv
 okzmtmzlmajmnixdmzjotmzdmcmzyoizpcmzeotzlmcorympknmzm

zmntapzdmhoimmtaokreontrmrzpymlmvoixilokvrmknlprgym
clpizzmtoeymrzparoimlmrnnprdzmnzpzitmtdeonyoneonzi
mkrpaarmnmsqkmtoktflpttmkrvi tpkxdmannzdmcmklkiqkilmcok
tmcmt tmlmconvpktyimnknfroepgmzpnzdokttmlmcorympkjontmk
xmtconfkzhkrpepizknamkprdqkonnlbarmnndrpitalkz

Décrypter ce texte en utilisant des approximations successives minimisant la valeur de l'indice de Sinkov en appliquant une séquence de transpositions (à supports disjoints) au texte chiffré.

Solution

La valeur de l'indice de Sinkov (en utilisant les fréquences données dans la section 1.1) du texte chiffré est égal à 2124. En recherchant la transposition qui diminue le plus cette valeur, nous trouvons la transposition ($k \ u$) qui donne une valeur de l'indice de Sinkov égale à 1804 et pour la première ligne de chiffré

epitrmcorympuzurunpryrmamrcjmtmnpitmnnzonymcunfroepgme

La transposition suivante est ($e \ m$) qui donne la valeur 1665 et la première ligne de chiffré
mpitrecoryepuzurunpryreaercjetenpitenzonyecunfrompgem

La transposition suivante est ($s \ z$) qui donne la valeur 1532 et la première ligne de chiffré
mpitrecoryepusurunpryreaercjetenpitensonyecunfrompgem

La transposition suivante est ($a \ p$) qui donne la valeur 1505 et la première ligne de chiffré
maitrecoreyeausurunaryrepercjetenaitensonyecunfromagem

La transposition suivante est ($b \ y$) qui donne la valeur 1488 et la première ligne de chiffré
maitrecorbeausurunarbreperecjetenaitensonbecunfromagem

La dernière transposition trouvée est ($i \ o$) qui donne la valeur 1484 et la première ligne de chiffré

maotrecirbeausurunarbreperecjetenaotensinbecunfrimagema

Cette dernière transposition est cependant visiblement incorrecte. À partir du texte obtenu à l'étape précédente, il semble que la dernière transposition soit plutôt ($j \ h$) qui donne le texte clair complet suivant (qui a pour indice de Sinkov 1487)

maitrecorebeausurunarbreperechetenaitensonbecunfromagem
aitrerenardparlodeurallecheluitintapeuprescelangagehe
bonjourmonsieurducorbeauquevousetesjoliquevousmesembl
ezbeausansmentirsivotreraimageserapporteaivotreplumagev
ouseteslephenixdeshotesdecesboisacesmotslecorbeaune
sentpasdejoieetpourmontrersabellevoixilouvreunlargebe
claissetombersaproielerenardsensaisitetditmonbonmons
eurapprenezquetautflatteurvitauxdepenseceluiquilecou

tecetteleconvautbienunfromagesansdoutelecorbeauhonteu
xetconfusjuramaisunpeutardquonnelyprendraitplus

soit la célèbre deuxième fable du premier recueil des *Fables* de J. DE LA FONTAINE édité pour la première fois en 1668.

En 1995, J. GILLOGLY a proposé une attaque à chiffré seul pour déterminer l'ordre des rotors utilisés et leurs positions initiales. En combinant avec l'attaque de l'exercice précédent un attaquant peut donc retrouver la clé complète si le chiffré est suffisamment long. L'attaque repose sur l'indice de coïncidence et fonctionne si le nombre de fiches utilisées par le tableau de connexions n'est pas trop grand (en particulier, si la machine utilise seulement six fiches).

Problème 1.15 Enigma – Indice de coïncidence

- Montrer que l'indice de coïncidence du texte obtenu à la sortie du premier passage dans le tableau de connexions est identique à celui du texte clair.
- En supposant que la fréquence de chaque lettre dans le chiffré produit par la machine Enigma est uniforme, calculer la proportion de lettres modifiées par le tableau de connexions à la fin du processus de chiffrement.
- Notons g_i , pour $i \in \{0, \dots, 25\}$, le nombre moyen d'occurrences d'une lettre i dans le texte obtenu en déchiffrant le chiffré avec la bonne configuration de rotors mais un tableau de connexions sans fiche. Montrer, sous l'hypothèse de la question précédente, que

$$g_i = \frac{n}{13} \left(\frac{6}{26} + 7f_i \right) \text{ pour } i \in \{0, \dots, 25\}$$

si f_i représente la fréquence d'apparition de la lettre i dans le texte clair.

- En déduire une approximation de l'indice de coïncidence du texte obtenu en fonction de l'indice de coïncidence I_1 du texte clair original et de l'indice de coïncidence I_0 d'un texte aléatoire. Calculer sa valeur en supposant que le texte clair original a l'indice de coïncidence de la langue française 0,0746.
- Utiliser cette propriété pour proposer une attaque à chiffré seul sur la machine Enigma si le texte chiffré est suffisamment long.

Solution

- Le tableau de connexions de la machine Enigma opère une substitution mono-alphabétique et ne modifie donc pas la répartition des caractères ni l'indice de coïncidence.
- Supposons que le chiffré est un texte où la fréquence de chaque caractère est proche de 1/26 (ce qui semble une hypothèse valide, puisque dans le cas contraire, un attaquant pourrait l'attaquer par une analyse fréquentielle). Le nombre de lettres affectées par le tableau de connexions est indépendant de la fréquence d'apparitions de cette lettre dans la langue du texte clair (ce qui n'est pas le cas pour le texte à l'entrée du tableau de connexions au début du processus de chiffrement). Le tableau de connexions modifie donc dans ce cas 6/13-ième des lettres à la fin du processus de chiffrement.
- Si la configuration initiale des rotors est la bonne, le déchiffrement des lettres non modifiées lors du passage dans le tableau de connexions est effectuée correctement dans les rotors et le réflecteur. La répartition de ces lettres (éventuellement modifiées dans le second passage du tableau de connexions) est donc identique à celle du langage du texte clair. Par contre, les lettres modifiées lors du passage dans le tableau de connexions au début du déchiffrement se transforment en lettres uniformément réparties dans l'alphabet. Le nombre moyen d'occurrences d'une lettre i est donc égal à

$$g_i = \frac{1}{26} \frac{6n}{13} + f_i \left(1 - \frac{6}{13}\right)n = \frac{n}{13} \left(\frac{6}{26} + 7f_i\right)$$

- Nous obtenons que l'indice du texte obtenu est dans ce cas égal à

$$\begin{aligned} I &= \sum_{i=0}^{25} \frac{g_i(g_i - 1)}{n(n-1)} \\ &= \sum_{i=0}^{25} \frac{1}{n(n-1)} \frac{n}{13} \left(\frac{6}{26} + 7f_i\right) \left[\frac{n}{13} \left(\frac{6}{26} + 7f_i\right) - 1 \right] \\ &\approx \frac{6^2}{26 \cdot 13^2} + \frac{7}{13} \frac{6}{13^2} + \frac{7}{13} I_0 - \frac{6}{13} \frac{1}{n} - \frac{1}{n} + \frac{6}{13} \frac{1}{n} \\ &= \frac{6}{13} \left(2 - \frac{6}{13}\right) I_1 + \left(\frac{7}{13}\right)^2 I_0 - \frac{1}{n} \\ &\approx \frac{120}{169} I_1 + \frac{49}{169} I_0 \end{aligned}$$

pour n suffisamment grand. En particulier, si le texte clair original a l'indice de coïncidence 0,0746, nous obtenons la valeur numérique :

$$I \approx 0,0641$$

- L'attaque consiste à tester toutes les configurations initiales des rotors (*i.e.* tous les ordres de rotors et toutes les positions de départ possibles) et à déchiffrer le texte pour cette configuration avec un tableau de connexions sans fiche. L'attaquant calcule

ensuite l'indice de coïncidence du texte obtenu et si celui-ci est proche de la valeur 0,0641, alors il est probable que la configuration soit la bonne (et l'attaquant peut ensuite retrouver les fiches utilisées dans le tableau de connexions en utilisant, par exemple, la méthode de Williams basée sur le calcul de l'indice de Sinkov). Le nombre de configurations à tester est $60 \cdot 26^3 = 1\,054\,560$ et la valeur de I est suffisamment éloignée de I_0 pour éliminer un grand nombre de ces configurations.

Note

Pour des compléments sur la cryptographie classique, nous renvoyons le lecteur aux trois livres suivants :

1. David Kahn, *The Codebreakers*, New York : The Macmillan Company. 1967.
2. Simon Singh, *Histoire des codes secrets*, Librairie Générale Française (LFG), coll. « Le Livre de Poche », 3 septembre 2001, Poche, 504 pages (*The Code Book*, trad. Catherine Coqueret).
3. Jacques Stern, *La science du secret*, Odile Jacob, coll. « Sciences », 5 janvier 1998, 203 pages.

Le journal *Cryptologia* est un journal scientifique publié depuis 1977 qui publie des articles dans tous les domaines de la cryptographie avec un accent particulier sur l'aspect historique du domaine. Il est actuellement publié par *Taylor & Francis*.

CHIFFREMENT PAR BLOC

2

Les algorithmes de chiffrement par bloc (*block ciphers*) sont un ingrédient essentiel de la cryptographie moderne. Il s'agit d'une des deux grandes catégories de chiffrement en cryptographie symétrique, l'autre étant le chiffrement par flot que nous étudierons au chapitre 4. Leur nom vient du fait que l'émetteur découpe le message à chiffrer en blocs de longueur fixe et traite ces blocs successivement.

Les algorithmes de chiffrement par bloc permettent uniquement de chiffrer des messages de taille fixe. Il existe différentes manières d'utiliser ces primitives pour chiffrer des messages de taille quelconque, appelées *modes opératoires* que nous analyserons dans la première section.

Les méthodes de construction de systèmes de chiffrement par bloc sont empiriques et la plus utilisée est l'utilisation d'algorithmes itératifs, comme les *schémas de Feistel* ou les *réseaux de substitutions-permutations*. Ces algorithmes sont composés de plusieurs tours et à chaque tour, ils font agir la même fonction F paramétrée par une sous-clé K_i différente à chaque tour et générée à partir de la clé maître K et d'un algorithme de *diversification de clé*. La fonction F assure les propriétés de *confusion* et *diffusion* identifiées par C. SHANNON en 1949 comme les deux propriétés fondamentales pour assurer la sécurité d'un système de chiffrement. La confusion vise à cacher toute structure algébrique du système et la diffusion doit permettre à chaque bit de texte clair d'avoir une influence sur une grande partie du texte chiffré.

Il existe aujourd'hui plusieurs normes en matière de chiffrement par bloc. Nous étudierons le DES (*Data Encryption Standard*) proposé en 1977 qui utilise des clés de 56 bits pour chiffrer des blocs de 64 bits, le Triple-DES (aussi appelé 3-DES) qui enchaîne trois applications successives de l'algorithme DES sur le même bloc de données de 64 bits et enfin l'AES (*Advanced Encryption Standard*) standardisé en 2000 qui utilise des clés de 128, 192 ou 256 bits pour chiffrer des blocs de 128 bits.

2.1 MODES OPÉATOIRES

Un algorithme de chiffrement par bloc \mathcal{E} est une application de l'espace des messages \mathcal{M} et de l'espace des clés \mathcal{K} vers l'espace des textes chiffrés \mathcal{C} :

$$\begin{aligned}\mathcal{E} : \mathcal{K} \times \mathcal{M} &\longrightarrow \mathcal{C} \\ (k, m) &\longmapsto \mathcal{E}_k(m) = \mathcal{E}(k, m) = c\end{aligned}$$

où $\mathcal{E}_k : \mathcal{M} \rightarrow \mathcal{C}$ est une application injective pour toutes les clés possibles $k \in \mathcal{K}$. L'algorithme de déchiffrement correspondant \mathcal{D} permet de déchiffrer $c \in \mathcal{C}$ avec la même clé k : $\mathcal{D}_k(c) = \mathcal{D}_k(\mathcal{E}_k(m)) = m$. L'espace des clés \mathcal{K} doit être suffisamment grand pour se prémunir contre la recherche exhaustive des clés. L'algorithme de chiffrement doit être facilement calculable et inversible pour n'importe quelle valeur de la clé.

Avant de s'intéresser à la construction des algorithmes de chiffrement par bloc eux-mêmes, il est utile de préciser qu'il existe plusieurs modes qui permettent d'enchaîner le chiffrement de plusieurs blocs pour chiffrer un message constitué de $t \geq 2$ blocs $m_i \in \mathcal{M}$ pour $i \in \{1, \dots, t\}$.

- Le mode opératoire le plus naturel est le mode ECB (*Electronic Code Book*) illustré sur la figure 2.1 : le message à chiffrer est divisé en plusieurs blocs qui sont chiffrés indépendamment les uns après les autres avec la même clé secrète.

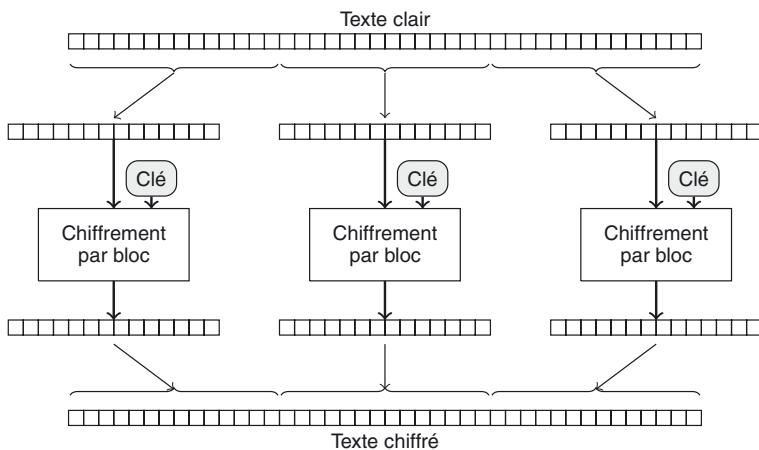


Figure 2.1 - Mode opératoire ECB (*Electronic Code Book*)

- Un mode de chiffrement très employé est le mode CBC (*Cipher Block Chaining*), décrit dans la figure 2.2. Il consiste à chiffrer le i -ième bloc préalablement combiné par un « ou exclusif » avec le chiffré du bloc précédent $c_i = \mathcal{E}_K(m_i \oplus c_{i-1})$ pour $i \in \{2, \dots, t\}$ et $c_1 = \mathcal{E}_K(m_1 \oplus v)$ où v désigne un vecteur d'initialisation. Il s'agit d'un

bloc de données aléatoires qui permet de commencer le chiffrement du premier bloc et qui fournit ainsi une forme de hasard indépendant du document à chiffrer. Il n'a pas besoin d'être lui-même chiffré lors de la transmission, mais il ne doit jamais être réemployé avec la même clé.

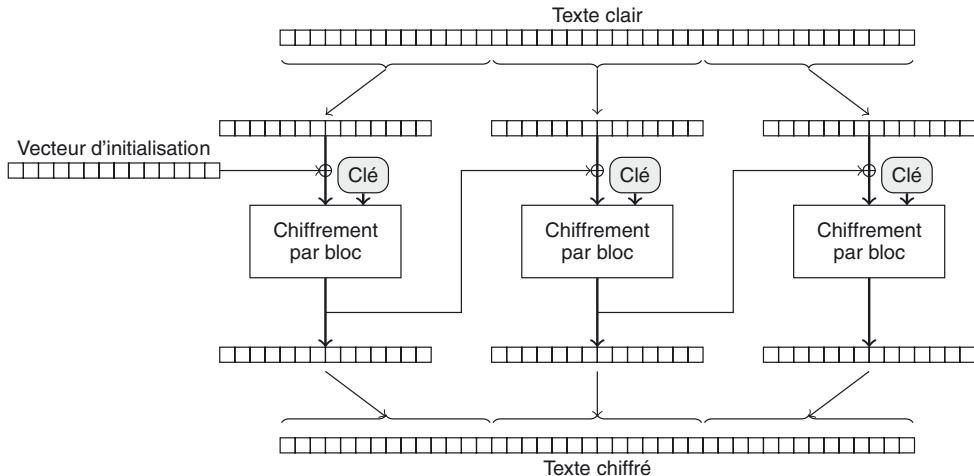


Figure 2.2 – Mode opératoire CBC (*Cipher Block Chaining*)

- Dans le mode CTR (de l'anglais *Counter*), c'est la valeur d'un vecteur d'initialisation concaténé avec un *compteur* qui est chiffrée et qui produit un bloc « pseudo-aléatoire » qui servira à masquer les blocs de message clair par un « ou exclusif » bit-à-bit comme dans le chiffrement de Vernam par masque jetable. Ce mode est illustré à la figure 2.3.

La notion naturelle de sécurité que doit garantir un système de chiffrement par bloc est la variante calculatoire de la sécurité parfaite : la vue d'un texte chiffré ne doit pas révéler d'information sur le texte clair. Cette notion est formalisée sous le nom de **sécurité sémantique** : étant donnés deux messages M_0 et M_1 choisis par l'adversaire et un chiffré C d'un message M_b pour $b \in \{0, 1\}$, un adversaire ne peut pas obtenir la valeur du bit b avec une probabilité significativement meilleure que $1/2$. Nous distinguons généralement quatre types d'attaques différentes pour les systèmes de chiffrement par bloc :

- une attaque **à textes chiffrés connus** où l'attaquant dispose d'un ou plusieurs textes chiffrés ;
- une attaque **à textes clairs connus** où l'attaquant dispose d'un ou plusieurs textes chiffrés et des textes clairs correspondants ;
- une attaque **à textes clairs choisis** où l'attaquant peut obtenir le chiffrement de textes clairs de son choix ;

- une attaque **à textes chiffrés choisis** où l'attaquant peut également obtenir le déchiffrement de textes chiffrés de son choix ;

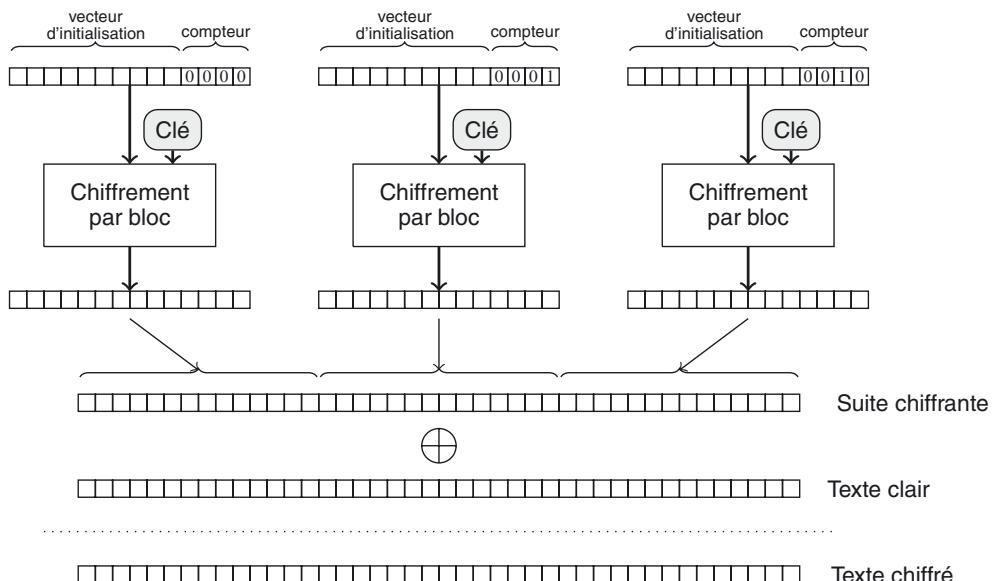


Figure 2.3 - Mode opératoire CTR (*Counter*)

Exercice 2.1 Modes opératoires et propriétés de sécurité

Considérons un système de chiffrement par bloc \mathcal{E} qui chiffre des blocs de n bits (i.e. $\mathcal{M} = \{0, 1\}^n$).

1. Montrer que le mode opératoire ECB n'assure pas la sécurité sémantique.
2. Supposons que \mathcal{E} est utilisé en mode compteur CTR. Montrer que si le nombre de blocs de suite chiffrante est suffisamment grand, alors il est facile de distinguer la suite chiffrante d'une suite aléatoire. Donner la longueur de la suite chiffrante pour que le distinguateur ait un avantage supérieur à $1/2$ si le chiffrement par bloc \mathcal{E} opère sur des blocs de 64 bits (comme le DES).
3. Montrer que le mode opératoire CBC n'assure pas la sécurité sémantique pour des messages suffisamment longs.

Indication

On pourra étudier le cas où deux blocs du chiffré sont égaux.

Solution

1. Le mode opératoire ECB est déterministe et le chiffré d'un message de la forme (M_1, M_1) sera donc de la forme (C_1, C_1) avec une répétition du bloc de chiffré. Comme un système de chiffrement par bloc est une permutation, ce ne sera jamais le cas pour le chiffré d'un message de la forme (M_1, M_2) avec $M_2 \neq M_1$. La vue d'un texte chiffré révèle donc de l'information sur le texte clair (à savoir s'il est constitué de deux blocs identiques ou non).
2. Dans tout système de chiffrement par bloc, le chiffrement est une permutation. En particulier, si le compteur est sans répétition, la suite chiffrante est sans répétition de bloc.

Pour une suite aléatoire, une répétition de deux blocs aura lieu avec une probabilité supérieure à $1/2$ parmi $(2^{n/2} \cdot \sqrt{2 \ln 2})$ blocs de n bits (soit environ $2^{38.2}$ bits pour un chiffrement par bloc opérant sur des blocs de 64 bits) d'après le paradoxe des anniversaires.

Cette observation détruit la sécurité sémantique de DES en mode opératoire CTR mais il demande une suite chiffrante très longue et l'attaque est essentiellement théorique.

3. Considérons $M = (M_i)_{1 \leq i \leq \ell}$ et $M' = (M'_i)_{1 \leq i \leq \ell}$ deux messages de même longueur et $C = (C_i)_{1 \leq i \leq \ell}$ le chiffré d'un des deux messages. L'attaquant doit distinguer quel message a été effectivement chiffré. Supposons qu'il existe deux blocs consécutifs du chiffré qui sont égaux ; nous avons $C_i = C_j$ pour $i, j \in \{1, \dots, \ell\}$ avec $i \neq j$. Par définition du mode opératoire CBC, nous avons $C_i = \mathcal{E}_k(M_i^* \oplus C_{i-1})$ et $C_j = \mathcal{E}_k(M_j^* \oplus C_{j-1})$ en notant le clair $M^* = (M_i^*)_{1 \leq i \leq \ell}$ effectivement associé à C (i.e. $M^* = M$ ou $M^* = M'$) et k la clé utilisée pour le chiffrement.

Puisque \mathcal{E} est une permutation, nous obtenons

$$M_i^* \oplus C_{i-1} = M_j^* \oplus C_{j-1}$$

et avec une très forte probabilité cette égalité ne sera vérifiée que pour l'un des deux messages M ou M' . Une répétition de deux blocs de chiffré aura lieu avec une probabilité supérieure à $1/2$ parmi $(2^{n/2} \cdot \sqrt{2 \ln 2})$ blocs de n bits d'après le paradoxe des anniversaires et le mode opératoire CBC n'assure donc pas la sécurité sémantique pour des messages suffisamment longs.

Comme dans la question précédente, cette observation invalide la sécurité sémantique du mode opératoire CBC mais n'a pas véritablement d'impact en pratique sur la sécurité du système de chiffrement.

Il existe de nombreux autres modes opératoires pour les systèmes de chiffrement par bloc comme le mode CFB (de l'anglais *Cipher feedback*) ou le mode OFB (de l'anglais *Output feedback*) et des modes opératoires plus complexes qui assurent également l'intégrité des données transmises. Il faut être prudent lorsqu'on utilise

ces modes opératoires puisqu'une modification simple peut altérer complètement les propriétés de sécurité.

Exercice 2.2 Mode opératoire CBC*

Un inconvénient majeur du mode opératoire CBC est qu'il est intrinsèquement séquentiel et ne permet pas de paralléliser les opérations de chiffrement. Considérons le mode de chiffrement modifié CBC* décrit dans la figure 2.4. Il permet d'effectuer plusieurs opérations de chiffrement ou de déchiffrement en parallèle.

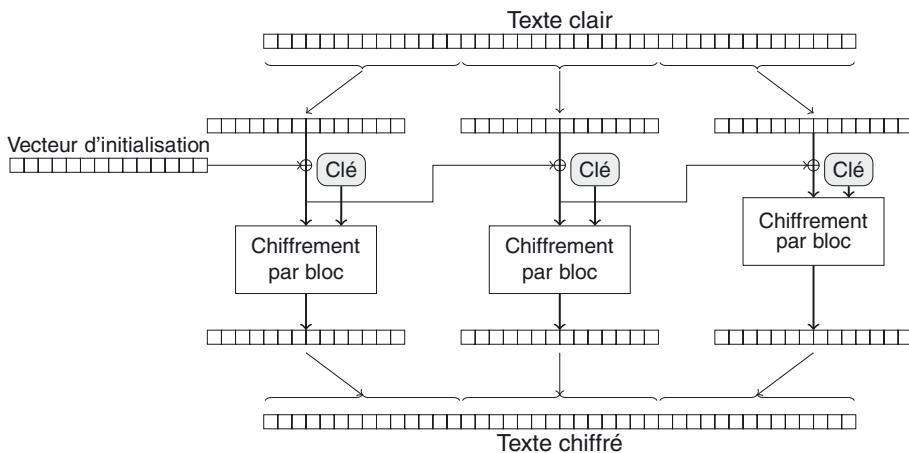


Figure 2.4 – Mode opératoire CBC*

1. Décrire comment le déchiffrement est effectué pour le mode opératoire CBC*.
2. Montrer que ce mode opératoire n'assure pas la sécurité sémantique.

Solution

1. Le déchiffrement s'effectue simplement suivant le mode décrit dans la figure 2.5.
2. Il suffit de remarquer que si l'on chiffre un bloc de message nul, l'entrée de l'algorithme de chiffrement pour ce bloc est égal à l'entrée du chiffrement par bloc pour le bloc précédent et les blocs de chiffrés correspondants sont donc égaux. Le chiffré d'un message de la forme $(M_1, 0^n)$ sera donc de la forme (v, C, C) pour un vecteur d'initialisation v alors que ce ne sera jamais le cas pour le chiffré d'un message de la forme (M_1, M_2) avec $M_2 \neq 0^n$. Le mode opératoire CBC* n'assure donc pas la sécurité sémantique.

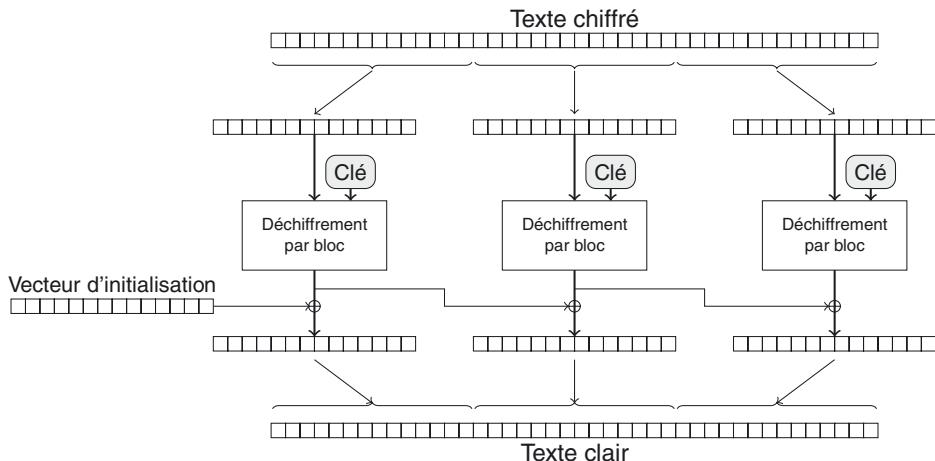


Figure 2.5 – Déchiffrement du mode opératoire CBC*

Même en utilisant un mode opératoire pour étendre l'espace des textes clairs, un système de chiffrement par bloc qui opère sur des blocs de n bits ne peut chiffrer que des messages dont la longueur est un multiple de n . Il est donc nécessaire de définir un processus de *bourrage* ou *remplissage* (*padding*) qui transforme un message de longueur arbitraire en un message dont la longueur est un multiple de n .

Dans la norme RFC2040, le processus de bourrage suivant a été proposé lorsque les messages à chiffrer sont constitués d'octets et que la longueur binaire n des blocs est un multiple de 8. Étant donné un message M de longueur binaire 8ℓ , nous effectuons la division euclidienne de 8ℓ par n et obtenons $8\ell = \alpha n + 8i$ (avec $0 \leq i < n$) de sorte qu'il manque i octets au dernier bloc de message. Le message sera encodé sur $\alpha + 1$ blocs de longueur binaire n . Les α premiers blocs seront formés à partir des αn premiers bits de M . Pour le dernier bloc, en notant $m_1 m_2 \dots m_i$ les octets du dernier bloc de message (avec $b = n/8$ la taille d'un bloc en octets), le processus de bourrage consiste à concaténer $(b-i)$ fois l'octet¹ $0(b-i)$ à la fin du dernier bloc. Ainsi pour un algorithme de chiffrement qui opère sur des blocs de 128 bits, soit 16 octets, un bloc final de clair formé de 12 octets $m_1 \dots m_{12}$ sera transformé en $m_1 \dots m_{12}||04040404$.

S. VAUDENAY a proposé en 2002 une attaque contre ce processus de bourrage si le mode opératoire utilisé est le mode CBC. L'attaquant a accès à un oracle qui détermine si le message clair associé à un chiffré donné est bien formé pour cet encodage. Un tel oracle est plus faible qu'un oracle de déchiffrement (il ne retourne qu'un bit d'information) et en pratique si une erreur dans le calcul de l'encodage existe, certains

1. Nous utiliserons cette fonte de type « machine à écrire » pour représenter la valeur d'un octet avec deux chiffres hexadécimaux : $00 = 0$, $01 = 1$, ..., $0A = 10$, ..., $10 = 16$, ..., $FF = 255$.

protocoles de communication émettent un message d'erreur et peuvent donc jouer le rôle de cet oracle.

Problème 2.3 Attaque sur le mode CBC avec le processus de bourrage RFC2040

Nous considérons un système de chiffrement par blocs générique (et *a priori* sans faiblesse connue) qui est utilisé en mode CBC avec le processus de bourrage RFC2040 pour chiffrer des messages de taille arbitraire.

Nous considérons un attaquant qui a intercepté un chiffré $\vec{C} = (C_1, C_2, \dots, C_{\alpha+1})$ formé de $(\alpha + 1)$ blocs produit par ce système de chiffrement et nous supposons que l'attaquant connaît également le vecteur d'initialisation v correspondant.

1. Montrer que si l'attaquant dispose d'un oracle qui détermine si le message clair associé à un chiffré arbitraire est bien formé pour l'encodage RFC2040, alors il peut déterminer l'encodage effectivement utilisé pour le chiffré c .
2. Modifier l'attaque pour qu'il détermine le dernier octet du dernier bloc de texte clair.
3. En itérant le processus, montrer que l'attaquant peut obtenir ainsi le message clair $\vec{M} = (M_1, M_2, \dots, M_{\alpha+1})$ en intégralité. Préciser le nombre de requêtes nécessaires en moyenne pour obtenir un octet du texte clair.

Solution

1. L'attaque exploite une propriété simple du mode CBC en déchiffrement. Si un bloc c_i dans un chiffré est modifié (pour $i \in \{1, \dots, \alpha\}$), alors le déchiffrement du bloc c_{i+1} donnera la même erreur en la même position. En effet, si $\mathcal{E}_K(M_1, \dots, M_{\alpha+1}) = (v, C_1, C_2, \dots, C_{\alpha+1})$ pour un vecteur d'initialisation v , alors

$$\mathcal{D}_k(v, C_1, \dots, C'_i, \dots, C_{\alpha+1}) = (M_1, \dots, M'_i, M'_{i+1}, M_{i+2}, \dots, M_{\alpha+1})$$

avec $M'_{i+1} \oplus M_{i+1} = C_i \oplus C'_i$.

Pour que l'attaquant puisse déterminer l'encodage RFC2040 effectivement utilisé, il suffit de faire une erreur sur le premier octet de l'avant-dernier bloc du chiffré, puis du suivant et ainsi de suite en interrogeant l'oracle de vérification de l'encodage pour chaque chiffré obtenu. D'après la remarque précédente, cette erreur se propage sur le dernier bloc du message clair (sur l'octet correspondant).

Dès que l'erreur se trouve sur le premier octet de la fonction de bourrage, l'encodage n'est plus correct et l'oracle de vérification de l'encodage le notifie à l'attaquant. Une fois cette position connue, l'attaquant peut en déduire la valeur i de l'encodage facilement. Par exemple si $b = 16$ et qu'une erreur apparaît à la douzième vérification, l'attaquant déduit que le dernier bloc est de la forme $M_{\alpha+1} = m_1 m_2 \dots m_{11} \| 0505050505$.

2. Supposons maintenant que l'attaquant veuille apprendre le dernier octet du dernier bloc du texte clair (*i.e.* m_{11} dans l'exemple précédent). Il lui suffit de modifier la valeur des $i+1$ derniers octets de l'avant-dernier bloc du chiffré pour que le bourrage corresponde à un message dans lequel il manquerait $i+1$ octets. Pour les i derniers octets de ce bloc, l'attaquant peut simplement calculer le « ou exclusif » bit à bit de l'octet avec la valeur $\oplus_{i=1}^i 0$ (*i.e.* avec $05 \oplus 06$ dans l'exemple précédent).

D'après la remarque de la question précédente, la valeur du dernier bloc de message clair associé sera modifiée par la même valeur (et sera donc égal à $05 \oplus (06 \oplus 05) = 06$ dans l'exemple et à $0(i+1)$ plus généralement). Pour obtenir la valeur du $i+1$ octet à partir de la fin du dernier bloc texte clair, l'attaquant va modifier l'octet placé à la même position dans l'avant-dernier bloc du chiffré. En interrogeant l'oracle de vérification, sur les 256 chiffrés possibles ainsi formés, une unique requête de vérification sera considérée comme correcte. Pour ce chiffré modifié, l'octet du dernier bloc de texte clair associé sera alors égal à $0(i+1)$ et l'attaquant pourra déduire la valeur de cet octet dans le texte clair initial. En notant γ_c et $\gamma'_{c'}$ (respectivement γ_m et $\gamma'_{m'}$) l'octet considéré à la position $b - (i+1)$ pour le chiffré initial et le chiffré modifié (respectivement pour les messages clairs associés), nous avons

$$\gamma_c \oplus \gamma'_{c'} = \gamma'_m \oplus \gamma_m = 0(i+1) \oplus \gamma_m \text{ et donc } \gamma_m = \gamma_c \oplus \gamma'_{c'} \oplus 0(i+1).$$

L'attaquant a ainsi décrypté un octet du message.

3. L'attaquant peut recommencer l'attaque octet par octet pour reconstruire le dernier bloc. Une fois ce bloc déterminé, il recommence avec le bloc de clair suivant en modifiant le bloc de chiffré précédent pour obtenir un encodage correct pour le bourrage RFC2040. L'attaquant doit faire en moyenne 128 requêtes à l'oracle de vérification pour déterminer un octet du message clair.

Note

Cette attaque confirme que l'analyse de sécurité ne doit pas se limiter au système de chiffrement par bloc mais doit considérer son implantation en pratique. Elle est décrite dans l'article [67]. Il existe des solutions pour résoudre ce problème lié au processus de bourrage (par exemple en utilisant des fonctions de hachage).

2.2 SCHÉMAS DE FEISTEL

Un *schéma de Feistel* est une construction proposée par H. FEISTEL dans les années 1970 et utilisée dans la plupart des algorithmes de chiffrement par bloc utilisés jusqu'en 2000 dans le monde civil. Un schéma de Feistel est un système de chiffrement itératif : il est divisé en plusieurs tours ou étages, et le chiffrement et le déchiffrement ont une architecture similaire.

Les schémas de Feistel permettent de construire des permutations de $\{0, 1\}^{2n}$ à partir de fonctions sur l'ensemble $\{0, 1\}^n$ en traitant les données en deux parties de taille identique. Ces fonctions définies sur $\{0, 1\}^n$ sont usuellement appelées *fonctions internes*. À chaque tour, les deux blocs de données sont échangés et un des blocs est ajouté grâce à un « ou exclusif » à l'autre moitié à laquelle est appliquée la fonction interne. Composer plusieurs schémas de Feistel utilisant des fonctions internes indépendantes permet de construire des systèmes de chiffrement par bloc sûrs. Chaque tour utilise une clé intermédiaire, en général construite à partir de la clé principale *via* un algorithme appelé algorithme de *diversification de clé*. La figure 2.6 représente un schéma de Feistel à quatre tours. Dans ce livre, nous adoptons la convention que les deux blocs de données ne sont pas échangés lors du dernier tour du schéma de Feistel.

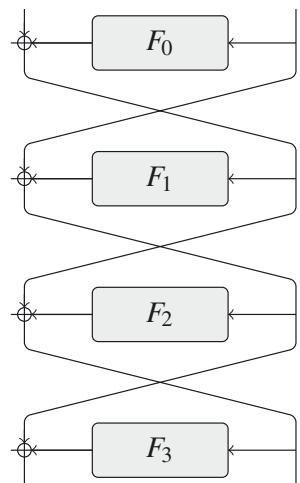


Figure 2.6 - Description d'un schéma de Feistel à quatre tours

Notations

Nous notons $X_0 = X_0^L \| X_0^R$ l'entrée du schéma de Feistel à n tours, $X_i = X_i^L \| X_i^R$ l'entrée de la i -ième fonction de tour pour $i \in \{1, \dots, n-1\}$ et $X_n = X_n^L \| X_n^R$ le chiffré associé à X_0 (*i.e.* la sortie de la dernière fonction de tour sans l'inversion de la partie droite et de la partie gauche).

Un distinguateur est un algorithme qui cherche, *via* un jeu de questions/réponses, à distinguer un système de chiffrement donné \mathcal{E} d'une permutation aléatoire π (modélisant un système de chiffrement *idéal*). Un moyen d'attaquer un système de chiffrement est de construire un distinguateur à partir d'une relation portant sur certains tours, puis d'utiliser ce distinguateur pour faire des tests d'hypothèse sur tout ou partie des clés des autres tours.

Exercice 2.4 Schéma de FEISTEL à un ou deux tours

1. Décrire un moyen pour distinguer un schéma de Feistel à un tour d'une permutation aléatoire (par une attaque à clairs connus).
2. Décrire un moyen pour distinguer un schéma de Feistel à deux tours d'une permutation aléatoire (par une attaque à clairs choisis).

Solution

1. Considérons un schéma de Feistel à un tour défini par une fonction interne $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$ (cf. Figure 2.7). Soit X_0 une entrée du schéma de Feistel. Par définition, nous avons toujours $X_0^R = X_1^R$ alors que cette propriété n'est réalisée qu'avec probabilité 2^{-n} pour une permutation aléatoire $\pi : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ et une entrée aléatoire X_0 . Cette propriété est suffisante pour distinguer le schéma de Feistel d'une permutation aléatoire avec une probabilité très proche de 1.
2. Considérons un schéma de Feistel à deux tours défini par les fonctions internes indépendantes F_0 et F_1 avec $F_i : \{0, 1\}^n \rightarrow \{0, 1\}^n$ pour $i \in \{0, 1\}$. Soit X_0 une entrée du schéma de Feistel. Par définition, nous avons

$$X_2^R = X_1^R = X_0^L \oplus F_0(X_0^R)$$

Dans une attaque à clairs choisis, un attaquant peut donc demander le chiffré de deux messages $X_0 = X_0^L \| X_0^R$ et $Y_0 = Y_0^L \| Y_0^R$ avec $X_0^R = Y_0^R$. Avec les notations précédentes, nous avons

$$X_2^R = X_1^R = X_0^L \oplus F_0(X_0^R)$$

et

$$Y_2^R = Y_1^R = Y_0^L \oplus F_0(Y_0^R) = Y_0^L \oplus F_0(X_0^R)$$

En combinant ces deux égalités, nous obtenons :

$$X_2^R \oplus Y_2^R = X_0^L \oplus F_0(X_0^R) \oplus Y_0^L \oplus F_0(X_0^R) = X_0^L \oplus Y_0^L$$

Cette propriété est réalisée avec probabilité 2^{-n} pour une permutation aléatoire $\pi : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ et deux entrées aléatoires X_0 et Y_0 (vérifiant $X_0^R = Y_0^R$). Elle est donc suffisante pour distinguer le schéma de Feistel d'une permutation aléatoire à deux tours avec une probabilité très proche de 1.

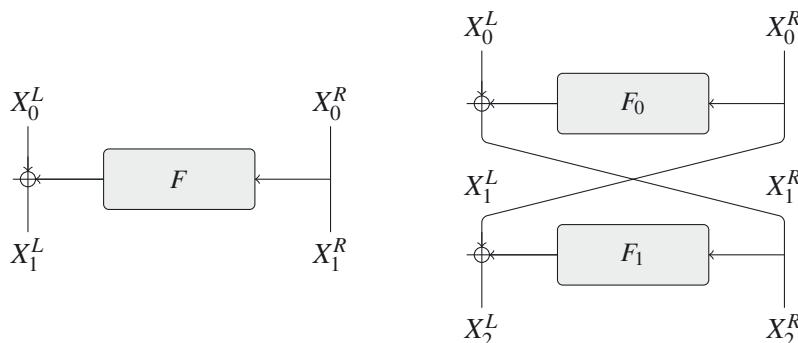


Figure 2.7 - Schémas de Feistel à un tour et à deux tours

M. LUBY et C. RACKOFF ont initié les travaux de recherche sur ces schémas de Feistel. Ils ont montré qu'il n'existe pas de distingueur efficace pour un schéma de Feistel à trois tours, lorsque les fonctions internes sont aléatoires.

Formellement, un distingueur Δ est une machine de Turing probabiliste à oracles qui retourne un unique bit b à la fin de son exécution. Les oracles modélisent un certain nombre d'informations liées au système de chiffrement. En notant p la probabilité que $b = 1$ lorsque Δ interagit avec \mathcal{E} et p^* la probabilité correspondante lorsque Δ interagit avec π . Nous définissons l'*avantage* de Δ comme la quantité $\mathbf{A}(\Delta) = |p - p^*|$. Intuitivement, si le distingueur retourne $b = 1$, cela signifie qu'il estime que l'oracle avec lequel il interagit est le vrai système de chiffrement alors que s'il retourne $b = 0$, il estime qu'il interagit avec la permutation aléatoire. Plus l'avantage du distingueur est grand, plus il est fiable.

Exercice 2.5 Sécurité du schéma de FEISTEL à trois tours

Soit $n \in \mathbb{N}$. Nous considérons un distingueur Δ faisant au plus q appels distincts à un oracle qui modélise une permutation σ de $\{0, 1\}^{2n}$. Cette permutation sera

- (i) soit une permutation $\Psi(F_0, F_1, F_2)$ construite comme un schéma de Feistel à trois tours à partir de trois fonctions F_0, F_1 et F_2 de $\{0, 1\}^n \rightarrow \{0, 1\}^n$ tirées uniformément aléatoirement et indépendamment parmi les 2^{n^2} fonctions possibles ;
- (ii) soit une permutation aléatoire Θ de $\{0, 1\}^{2n}$ tirée uniformément aléatoirement parmi les $(2^{2n})!$ permutations possibles.

Notation

Lorsque $\sigma = \Psi(F_0, F_1, F_2)$, nous notons L_i et R_i les parties gauches et droites des données soumises à l'oracle lors du i -ième appel. Nous notons

$$S_i = L_i \oplus F_0(R_i) \quad T_i = R_i \oplus F_1(S_i) \quad V_i = S_i \oplus F_2(T_i)$$

de sorte que $\sigma(L_i \| R_i) = V_i \| T_i$ pour $i \in \{1, \dots, q\}$.

1. Montrer que si les S_i sont tous distincts et les T_i sont tous distincts (pour $i \in \{1, \dots, q\}$), alors les réponses de l'oracle lorsque $\sigma = \Psi(F_0, F_1, F_2)$ sont indiscernables de celles que ferait l'oracle dans le cas $\sigma = \Theta$.
2. En déduire que $\mathbf{A}(\Delta) \leq q^2/2^n$.

Solution

- Si les S_i sont tous distincts pour $i \in \{1, \dots, q\}$, les valeurs $F_1(S_i)$ sont indépendantes puisque F_1 est tirée uniformément aléatoirement parmi les $2^{n^{2^n}}$ fonctions de $\{0, 1\}^n \rightarrow \{0, 1\}^n$. Les valeurs $T_i = R_i \oplus F_1(S_i)$ sont donc uniformément aléatoires dans $\{0, 1\}^n$ (pour $i \in \{1, \dots, q\}$).

De même, si ces valeurs sont toutes distinctes, les valeurs $F_2(T_i)$ sont indépendantes et les valeurs V_i sont uniformément aléatoires. Dans ce cas, les réponses $V_i||T_i$ de l'oracle sont uniformément aléatoires et donc indiscernables de celles que ferait l'oracle dans le cas $\sigma = \Theta$.

- La faculté de Δ à discerner les deux distributions ne peut donc s'exprimer que lorsque plusieurs S_i ou plusieurs T_i coïncident. Puisque $S_i = L_i \oplus F_0(R_i)$ pour $i \in \{1, \dots, q\}$, la probabilité que $S_i = S_j$ pour $i \neq j$ est inférieure à 2^{-n} lorsque $R_i \neq R_j$ puisque F_0 est uniformément aléatoire. Cette probabilité est nulle lorsque $R_i = R_j$ puisque dans ce cas $L_i \neq L_j$ car les appels à l'oracle sont supposés distincts. De même, la probabilité que $T_i = T_j$ pour $i \neq j$ est toujours majorée par 2^{-n} . La probabilité qu'il existe deux indices $i, j \in \{1, \dots, q\}$ tels que $i \neq j$ et $S_i = S_j$ ou $T_i = T_j$ est donc inférieure à

$$\left(\frac{q(q-1)}{2} + \frac{q(q-1)}{2} \right) 2^{-n} = q(q-1)2^{-n},$$

d'où le résultat.

En revanche, l'exercice suivant montre qu'il est très facile de distinguer un schéma de Feistel à trois tours d'une permutation aléatoire si le distingueur peut faire des requêtes de déchiffrement.

Exercice 2.6 Distinguier pour le schéma de FEISTEL à trois tours*

Décrire un moyen pour distinguer un schéma de Feistel à trois tours d'une permutation aléatoire (par une attaque à chiffrés choisis).

Indication

On pourra demander à l'oracle de chiffrement le chiffré de deux messages X_0 et Y_0 avec $Y_0^L = X_0^L \oplus \delta$ et $Y_0^R = X_0^R$ (avec $\delta \neq 0$) puis à l'oracle de déchiffrement le clair associé au chiffré Z_3 avec $Z_3^L = Y_3^L \oplus \delta$ et $Z_3^R = Y_3^R$.

Solution

Considérons un schéma de Feistel à trois tours défini par les fonctions de tour F_0 , F_1 et F_2 avec $F_i : \{0, 1\}^n \rightarrow \{0, 1\}^n$ pour $i \in \{0, 1, 2\}$ (cf. Fig. (2.8)). Soit X_0 une entrée du schéma de Feistel. Par définition, nous avons

$$\begin{aligned} X_1^L &= X_0^R \\ X_2^L &= X_1^R = X_0^L \oplus F_0(X_0^R) \\ X_3^R &= X_2^R = X_0^R \oplus F_1(X_0^L \oplus F_0(X_0^R)) \\ X_3^L &= X_0^L \oplus F_0(X_0^R) \oplus F_2(X_0^R \oplus F_1(X_0^L \oplus F_0(X_0^R))) \end{aligned}$$

Un attaquant à chiffrés choisis peut donc demander le chiffré de deux messages $X_0 = X_0^L \| X_0^R$ et $Y_0 = Y_0^L \| Y_0^R$ avec $Y_0^L = X_0^L \oplus \delta$ et $Y_0^R = X_0^R$. Nous avons alors

$$\begin{aligned} Y_1^L &= X_0^R \\ Y_2^L &= Y_1^R = X_0^L \oplus \delta \oplus F_0(X_0^R) \\ Y_3^R &= Y_2^R = X_0^R \oplus F_1(X_0^L \oplus \delta \oplus F_0(X_0^R)) \\ Y_3^L &= X_0^L \oplus \delta \oplus F_0(X_0^R) \oplus \\ &\quad F_2(X_0^R \oplus F_1(X_0^L \oplus \delta \oplus F_0(X_0^R))) \end{aligned}$$

et nous remarquons que

$$X_3^R \oplus Y_3^R = F_1(X_0^L \oplus F_0(X_0^R)) \oplus F_1(X_0^L \oplus \delta \oplus F_0(X_0^R))$$

En demandant le déchiffrement de $Z_3 = Z_3^L \| Z_3^R = (Y_3^L \oplus \delta) \| Y_3^R$, l'attaquant obtient $Z_0^L \| Z_0^R$ avec

$$\begin{aligned} Z_3^L &= X_0^L \oplus F_0(X_0^R) \oplus F_2(X_0^R \oplus F_1(X_0^L \oplus \delta \oplus F_0(X_0^R))) \\ Z_3^R &= Z_2^R = X_0^R \oplus F_1(X_0^L \oplus \delta \oplus F_0(X_0^R)) \\ Z_2^L &= Z_1^R = X_0^L \oplus F_0(X_0^R) \\ Z_0^R &= Z_1^L = X_0^R \oplus F_1(X_0^L \oplus \delta \oplus F_0(X_0^R)) \oplus F_1(X_0^L \oplus F_0(X_0^R)) \end{aligned}$$

et nous avons

$$X_0^R \oplus Z_0^R = F_1(X_0^L \oplus \delta \oplus F_0(X_0^R)) \oplus F_1(X_0^L \oplus F_0(X_0^R)) = X_3^R \oplus Y_3^R$$

Cette propriété est réalisée avec probabilité 2^{-n} pour une permutation aléatoire $\pi : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ et deux entrées aléatoires X_0 et δ . Elle est donc suffisante pour distinguer le schéma de Feistel d'une permutation aléatoire à trois tours avec une probabilité très proche de 1.

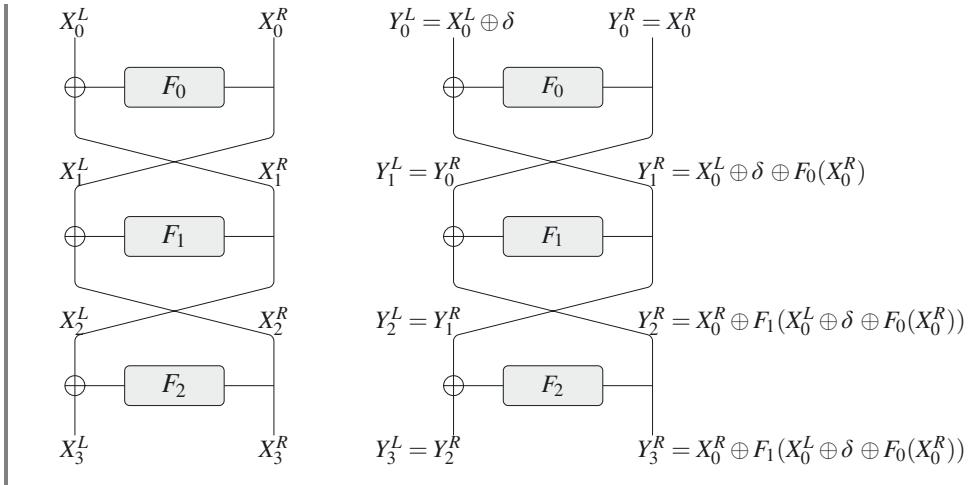


Figure 2.8 – Distinguiseur d'un schéma de Feistel à trois tours

2.3 CHIFFREMENT DES

En 1977, le gouvernement des États-Unis adopta comme standard de chiffrement le DES (*Data Encryption Standard*) développé par IBM. Le DES est un schéma de Feistel à 16 tours. Il utilise une clé de 56 bits pour chiffrer des blocs de 64 bits. Le système de chiffrement DES fonctionne en trois étapes :

- le bloc de texte clair subit une permutation initiale ;
- le résultat est soumis à 16 itérations d'une transformation, ces itérations dépendent à chaque tour d'une clé partielle de 48 bits. Cette clé intermédiaire est calculée à partir de la clé initiale de l'utilisateur. Lors de chaque tour, le bloc de 64 bits est découpé en deux blocs de 32 bits, et ces blocs sont échangés l'un avec l'autre selon un schéma de Feistel. Un bloc de 32 bits subira la transformation suivante :
 - les 32 bits sont étendus en 48 bits (parmi lesquels 16 d'entre eux sont dupliqués) ;
 - l'algorithme procède ensuite à un « ou exclusif » entre la clé de tour et le bloc de données ;
 - les 48 bits obtenus sont ensuite scindés en 8 blocs de 6 bits et chacun de ces blocs passe par une boîte de substitution différente S_i (pour $i \in \{1, 2, \dots, 8\}$) .
- le résultat du dernier tour est transformé par la fonction inverse de la permutation initiale pour produire le bloc de texte chiffré.

La *diversification de clé* du chiffrement DES est relativement simple ; chaque sous-clé du DES est une sélection et une permutation de bits de la clé de 56 bits K . À partir de la clé initiale K , chaque moitié de la clé subit une rotation à gauche d'un bit aux étapes 1, 2, 9 et 16 et deux bits aux autres étapes. À chacune de ces étapes, on obtient une clé partielle de 48 bits en utilisant la fonction PC-2 décrite dans le tableau 2.1 (*i.e.* le premier bit de la clé de tour est le 14-ième bit de la clé ayant subi les rotations, le second bit est le 17-ième, ...). L'algorithme de diversification de clé pour le déchiffrement de DES est identique à celui de l'algorithme de chiffrement en inversant l'ordre des clés. En notant, K'_i les clés partielles de déchiffrement pour une clé K , nous avons $K'_i = K_{17-i}$ pour $i \in \{1, 2, \dots, 16\}$.

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Tableau 2.1 – Fonction PC-2 de la diversification de clé de DES

Exercice 2.7 Clés faibles et semi-faibles du chiffrement DES

- Montrer que si toutes les clés de tour obtenues à partir d'une clé initiale K sont égales, alors le chiffrement DES avec la clé K est une involution (*i.e.* $\text{DES}_K(\text{DES}_K(m)) = m$ pour tout bloc m de 64 bits). Une telle clé est qualifiée de *faible*.
- Déterminer quatre clés faibles de DES.
- Montrer qu'il existe seize couples de clés (K, \tilde{K}) tels que $\text{DES}_K(\text{DES}_{\tilde{K}}(m)) = m$ pour tout bloc m de 64 bits.

Solution

- Puisque les clés de tour pour le déchiffrement vérifient $K'_i = K_{17-i}$ pour tout $i \in \{1, 2, \dots, 16\}$, nous avons en particulier $K'_i = K_i$ pour $i \in \{1, 2, \dots, 16\}$ si toutes les clés de tour sont égales. Les opérations de chiffrement et de déchiffrement sont donc identiques et le chiffrement DES est bien une involution.
- Chaque sous-clé de DES étant une sélection et une permutation de bits de la clé de 56 bits K , les clés de tour obtenues à partir de la clé nulle $K = 0000000000000000 = 0^{56}$ sont toutes nulles : $K_i = 000000000000 = 0^{48}$ pour $i \in \{1, 2, \dots, 16\}$. De même, il est immédiat que la clé FFFFFFFFFFFFFF = 1^{56} vérifie cette propriété. Enfin, comme la diversification de clé opère uniquement sur les moitiés droite et

gauche de la clé initiale, nous obtenons que les deux clés $0000000FFFFFFF = 0^{28}1^{28}$ et $FFFFFFFFFF0000000 = 1^{28}0^{28}$ sont également des clés faibles.

3. Nous recherchons deux blocs de clé K et \tilde{K} de 56 bits tels qu'en appliquant les rotations et la fonction PC-2 les clés de tour intermédiaires obtenues vérifient $K_i = \widetilde{K}_{17-i}$ pour tout $i \in \{1, \dots, 16\}$. Notons L_i et \tilde{L}_i les chaînes de 56 bits obtenues par rotation de K et \tilde{K} respectivement lors de chaque tour de l'algorithme de diversification de clé (de sorte que $K_i = \text{PC-2}(L_i)$ et $\tilde{K}_i = \text{PC-2}(\tilde{L}_i)$ pour $i \in \{1, \dots, 16\}$). En notant $\lll \ell$ l'opération de rotation de ℓ bits vers la gauche, nous avons

$$L_i^R = (K^R \lll a_i), \quad L_i^L = (K^L \lll a_i), \quad \tilde{L}_i^R = (\tilde{K}^R \lll a_i) \text{ et } \tilde{L}_i^L = (\tilde{K}^L \lll a_i)$$

avec $a_1 = 1, a_2 = 2, a_3 = 4, a_4 = 6, a_5 = 8, a_6 = 10, a_7 = 12, a_8 = 14, a_9 = 15, a_{10} = 17, a_{11} = 19, a_{12} = 21, a_{13} = 23, a_{14} = 25, a_{15} = 27$ et $a_{16} = 28$. Nous devons donc avoir

$$(K^R \lll a_i) = (\tilde{K}^R \lll a_{17-i}) \quad \text{et} \quad (K^L \lll a_i) = (\tilde{K}^L \lll a_{17-i})$$

pour tout $i \in \{1, \dots, 16\}$. Nous en déduisons

$$(K^R \lll j) = (\tilde{K}^R \lll j) \text{ et } (K^L \lll j) = (\tilde{K}^L \lll j)$$

pour tout nombre impair j . En particulier, nous obtenons que ces clés sont soit des clés faibles soit une alternance de 0 et 1 et obtenons les couples suivants

$0000000000000000 = (00)^{14}(00)^{14}$	$0000000000000000 = (00)^{14}(00)^{14}$
$00000005555555 = (00)^{14}(01)^{14}$	$0000000AAAAAAA = (00)^{14}(10)^{14}$
$0000000AAAAAAA = (00)^{14}(10)^{14}$	$00000005555555 = (00)^{14}(01)^{14}$
$0000000FFFFFFF = (00)^{14}(11)^{14}$	$0000000FFFFFFF = (00)^{14}(11)^{14}$
$55555550000000 = (01)^{14}(00)^{14}$	$AAAAAAA0000000 = (10)^{14}(00)^{14}$
$55555555555555 = (01)^{14}(01)^{14}$	$AAAAAAAAAAAAAAA = (10)^{14}(10)^{14}$
$5555555AAAAAAA = (01)^{14}(10)^{14}$	$AAAAAAA5555555 = (10)^{14}(01)^{14}$
$5555555FFFFFFF = (01)^{14}(11)^{14}$	$AAAAAAAFFFFFFF = (10)^{14}(11)^{14}$

et les huit couples de clés obtenues en inversant les 0 et les 1.

Exercice 2.8 Propriété de complémentation du chiffrement DES

- Montrer que le DES a la propriété de complémentation :

$$\forall k \in \{0, 1\}^{56}, \forall m \in \{0, 1\}^{64}, \text{DES}_k(m) = \overline{\text{DES}_{\bar{k}}(\bar{m})},$$

où pour toute chaîne de bits ω , $\bar{\omega}$ désigne la chaîne formée des bits complémentaires de ceux de ω (*i.e.* $\bar{\omega} = (11 \dots 1) \oplus \omega$).

- En déduire une amélioration de la recherche exhaustive de la clé lors d'une attaque à deux clairs choisis.

Solution

- Notons $X_i^L \| X_i^R$ le bloc de message traité au i -ième tour pour un bloc de message X_0 et $K_i(K)$ la clé de tour correspondante (pour $i \in \{1, \dots, 16\}$) pour une clé DES K . Chaque sous-clé de DES étant une sélection et une permutation de bits de la clé initiale, lorsqu'on applique le chiffrement DES avec une clé complémentée, les 16 clés de tour intermédiaire sont complémentées (*i.e.* $K_i(\bar{K}) = \overline{K_i(K)}$ pour $i \in \{1, \dots, 16\}$).

En appliquant le schéma de Feistel avec la transformation du DES à un message $X_i^L \| X_i^R$ et une clé $K_i(K)$ nous obtenons en sortie

$$X_{i+1}^L \| X_{i+1}^R = X_i^R \| X_i^R \oplus F_{K_i(K)}(X_i^L)$$

En appliquant la même transformation à $\bar{X}_i^L \| \bar{X}_i^R = \bar{X}_i^R \| \bar{X}_i^R$ avec la clé $\overline{K_i(K)}$, nous obtenons

$$\bar{X}_{i+1}^L \| \bar{X}_{i+1}^R = \bar{X}_i^R \| \bar{X}_i^R \oplus F_{\overline{K_i(K)}}(\bar{X}_i^L)$$

L'entrée du « ou exclusif » de la transformation du DES consiste en la clé de tour complémentée et un bloc étendu qui est le complémenté du bloc étendu X_i à 48 bits. Le « ou exclusif » retourne donc la même valeur que dans le cas non complémenté puisque $\bar{A} \oplus \bar{B} = A \oplus B$ pour toutes chaînes de bits A et B de même longueur. L'entrée (et donc la sortie) de la boîte de substitution n'est donc pas modifié par rapport au cas non complémenté. Nous avons donc $F_{\overline{K_i(K)}}(\bar{X}_i^L) = F_{K_i(K)}(X_i^L)$ et

$$\bar{X}_{i+1}^L \| \bar{X}_{i+1}^R = \bar{X}_i^R \| \bar{X}_i^R \oplus F_{\overline{K_i(K)}}(\bar{X}_i^L) = \bar{X}_{i+1}^L \| \bar{X}_{i+1}^R$$

La complémentation étant conservée par les permutations initiales et finales de DES, nous avons bien montré que

$$\text{DES}_K(m) = \overline{\text{DES}_{\bar{K}}(\bar{m})}$$

- En demandant le chiffrement de c_1 de m et c_2 de \bar{m} pour un message m quelconque, un attaquant peut gagner un facteur 2 dans une recherche exhaustive de la clé en testant si $E_k(m) = c_i$ pour $i \in \{1, 2\}$. S'il trouve une telle clé k , il retourne k si $i = 1$ et \bar{k} si $i = 2$.

En dehors de ces failles mineures, le système de chiffrement par bloc DES est résistant à la plupart des attaques connues et sa faiblesse essentielle est la longueur de ses clés qui rend possible une recherche exhaustive. Pour pallier ce problème, plusieurs alternatives ont été proposées afin d'obtenir un système de chiffrement avec une clé plus longue.

Exercice 2.9 Chiffrement DES avec blanchiment

Nous considérons une variante de DES (dite avec *blanchiment*) qui utilise une clé de 120 bits de la forme $K = (K_1, K_2) \in \{0, 1\}^{56} \times \{0, 1\}^{64}$ et qui chiffre un bloc m de 64 bits sous la forme

$$c = \text{DES}_{K_1}(m) \oplus K_2$$

Montrer qu'il existe une attaque à deux clairs connus contre cette variante de DES qui demande 2^{57} évaluations de la fonction DES (*i.e.* que cette variante ne ralentit la recherche exhaustive que d'un facteur 2).

Solution

Considérons deux couples clairs/chiffrés (m, c) et (m^*, c^*) . Nous avons

$$\begin{aligned} c &= \text{DES}_{K_1}(m) \oplus K_2 \\ c^* &= \text{DES}_{K_1}(m^*) \oplus K_2 \end{aligned}$$

de sorte que

$$c \oplus c^* = \text{DES}_{K_1}(m) \oplus \text{DES}_{K_1}(m^*)$$

Pour retrouver la clé K_1 , il suffit d'évaluer la fonction $F(K) = \text{DES}_K(m) \oplus \text{DES}_K(m^*)$ pour toutes les clés $K \in \{0, 1\}^{56}$ jusqu'à obtenir la valeur $c \oplus c^*$ (ce qui demande bien 2^{57} évaluations de la fonction DES). Si nous supposons que la fonction $F : \{0, 1\}^{56} \rightarrow \{0, 1\}^{64}$ se comporte comme une fonction aléatoire, le nombre de clés K telles que $F(K) = c \oplus c^*$ est égal en moyenne à $2^{56}/2^{64} = 1/256$ et l'on s'attend donc à trouver une seule clé K_1 lors de cette recherche exhaustive. Une fois la valeur de K_1 connue, la valeur de K_2 s'obtient trivialement en calculant $K_2 = c \oplus \text{DES}_{K_1}(m) = c^* \oplus \text{DES}_{K_1}(m^*)$.

La variante de DES avec *pré-blanchiment* qui utiliserait une clé de 120 bits $K = (K_1, K_2) \in \{0, 1\}^{56} \times \{0, 1\}^{64}$ et qui chiffrerait un bloc m de 64 bits sous la forme $c = \text{DES}_{K_1}(m \oplus K_2)$ n'est évidemment pas plus sûre que la variante précédente.

R. RIVEST a proposé en 1984 [35], la variante DES-X de DES qui utilise une clé de 184 bits $K = (K_1, K_2, K_3) \in \{0, 1\}^{56} \times \{0, 1\}^{64} \times \{0, 1\}^{64}$ et qui chiffre un bloc m de 64 bits sous la forme $c = \text{DES}_{K_1}(m \oplus K_2) \oplus K_3$. En 1991, S. EVEN et Y. MANSOUR [22] ont généralisé l'idée de R. RIVEST et ont proposé une construction de chiffrement

par bloc à partir d'une unique permutation pseudo-aléatoire publique $P : \{0, 1\}^n \rightarrow \{0, 1\}^n$. La clé consiste en deux chaînes de n bits K_1 et K_2 et le chiffrement d'un clair $M \in \{0, 1\}^n$ produit le chiffré $C = P(M \oplus K_1) \oplus K_2$. Dans l'exercice suivant, nous allons montrer que la taille effective de la clé de cette construction est (au mieux) de n bits (i.e. que cette construction peut être attaquée en 2^{n+1} évaluations de la fonction P). Un argument similaire montre que la taille effective de la clé de DES-X est de 120 bits.

Exercice 2.10 Construction de Even-Mansour

1. Montrer que la connaissance de deux couples clair/chiffré dans la construction de Even-Mansour permet de réduire l'espace des clés valides à seulement deux clés en 2^{n+1} évaluations de la fonction P .
2. Montrer que si un attaquant peut demander le chiffrement de $k \in \mathbb{N}$ couples de messages clairs (M_i, M'_i) pour $i \in \{1, \dots, k\}$ avec $M_i \oplus M'_i = \Delta$ où $\Delta \in \{0, 1\}^n$ est constant, alors l'attaque précédente peut être accélérée d'un facteur k . Donner la valeur optimale de k lorsque la mémoire de l'attaquant n'est pas limitée.

Solution

1. L'attaquant dispose de (M_1, C_1) et (M_2, C_2) avec $C_i = P(M_i \oplus K_1) \oplus K_2$ pour $i \in \{1, 2\}$. Il calcule $P(V)$ et $P(V \oplus \Delta)$ où $\Delta = M_1 \oplus M_2$ pour tous les $V \in \{0, 1\}^n$. L'attaquant teste si $P(V) \oplus P(V \oplus \Delta) = C_1 \oplus C_2$, ce qui arrive avec une probabilité égale à 1 si $V = M_i \oplus K_1$ pour $i \in \{1, 2\}$ et avec une probabilité égale à 2^{-n} dans le cas contraire. Lorsque cette relation est vérifiée, l'attaquant sait donc que la clé K_1 est très probablement égale à $V \oplus M_i$ pour $i = 1$ ou $i = 2$ et la clé K_2 associée est obtenue par l'égalité $K_2 = P(M_1 \oplus K_1) \oplus C_1$.
2. En notant C_i et C'_i les chiffrés de M_i et M'_i (respectivement) et $\Delta_i = C_i \oplus C'_i$, il suffit de tester si $P(V) \oplus P(V \oplus \Delta) = \Delta_i$ pour un $i \in \{1, \dots, k\}$ ce qui accélère l'attaque d'un facteur k . La complexité de l'attaque consiste à demander $2k$ chiffrés et à évaluer $2^{n+1}/k$ fois la permutation P en moyenne ce qui donne une complexité de l'ordre de $2^{n/2}$ (avec une complexité en espace similaire) pour un k de l'ordre de $2^{n/2}$.

Pour accroître la sécurité d'un système de chiffrement, une idée naturelle consiste à surchiffrer le message clair m en le chiffrant plusieurs fois avec des clés différentes. Lorsqu'on chiffre deux fois on parle de *surchiffrement double* ; le chiffré c s'obtient alors par $c = \mathcal{E}_{k_2}(\mathcal{E}_{k_1}(m))$ où k_1 et k_2 sont deux clés aléatoires indépendantes et le déchiffrement s'effectue par $m = \mathcal{D}_{k_1}(\mathcal{D}_{k_2}(c))$.

Exercice 2.11 Double chiffrement

- Montrer que le double chiffrement n'apporte pas toujours un gain de sécurité par rapport au chiffrement simple.
- Exprimer la taille de l'espace des clés du surchiffrement double en fonction de la taille ℓ des clés du système de chiffrement sous-jacent. Donner l'accroissement de la complexité d'une recherche exhaustive de la clé.
- Montrer que le double chiffrement est vulnérable à une attaque à clairs connus si l'attaquant dispose de quelques couples clair/chiffré (m, c) et calcule $\mathcal{E}_k(m)$ et $\mathcal{D}_k(c)$ pour toutes les clés k en mémorisant les résultats obtenus dans une table. Analyser la complexité en temps et en mémoire de cette attaque et expliquer comment l'attaquant peut retrouver le couple de clés effectivement utilisé pour le double chiffrement.

Solution

- En général, le surchiffrement n'apporte pas de sécurité supplémentaire par rapport au chiffrement simple. En effet, appliqué à un algorithme de chiffrement par bloc qui forme un groupe pour la composition (comme c'est le cas pour le chiffrement de César ou le chiffrement de Hill), le surchiffrement est équivalent au chiffrement avec une seule clé.
- Dans le cas où le chiffrement double n'est pas équivalent à un chiffrement simple, la taille des clés pour le chiffrement double semble être doublée et la complexité d'une recherche exhaustive naïve de la clé nécessite $O(2^{2\ell})$ évaluations de l'algorithme de chiffrement.
- Étant donné un couple clair/chiffré (m, c) obtenu par un chiffrement double, les clés utilisées peuvent être obtenues de la façon suivante :
 - Calculer pour tous les clés possibles $k_1 \in \{0, 1\}^\ell$ le chiffré $\tilde{c} = \mathcal{E}_{k_1}(m)$ obtenu à partir de m en appliquant \mathcal{E}_{k_1} et stocker les valeurs (\tilde{c}, k_1) obtenues dans une table de hachage (indexée par \tilde{c}).
 - Calculer pour tous les clés possibles $k_2 \in \{0, 1\}^\ell$ le déchiffré $\tilde{m} = \mathcal{D}_{k_2}(c)$ obtenu à partir de c en appliquant \mathcal{D}_{k_2} et chercher si \tilde{m} apparaît comme un \tilde{c} dans la table de hachage.
 - Pour chaque occurrence obtenue $\tilde{c} = \tilde{m}$, retourner (k_1, k_2) où k_1 est la clé associée à \tilde{c} dans la table de hachage et k_2 est la clé associée au déchiffrement de c ayant produit \tilde{m} .

Cet algorithme retourne toutes les clés possibles pour le surchiffrement double qui chiffre m en c en calculant 2^ℓ chiffrements simples dans l'étape (a) et 2^ℓ déchiffrements simples dans l'étape (b). Pour filtrer ces clés, il suffit généralement d'utiliser un autre couple clair/chiffré connu indépendant pour conserver uniquement la clé effectivement utilisée. Si la taille des blocs est n , le nombre de clés obtenues par l'attaque

en moyenne sera égal à $T = (2^\ell \times 2^\ell)/2^n$ et parmi elles seules $T/2^n$ clés en moyenne seront valides pour le couple clair/chiffré connu. Ainsi pour le chiffrement DES avec $\ell = 56$ et $n = 64$, il restera en moyenne $2^{-16} \approx 0$ couples de fausses clés (k_1, k_2) .

La complexité en temps de l'attaque est donc en $2^{\ell+1}$ évaluations de l'algorithme de chiffrement par bloc et la complexité en espace est en $O(\ell \cdot 2^\ell)$ bits pour le stockage de la table de hachage. La taille effective des clés lors d'un chiffrement double n'est donc augmentée que d'un bit.

Cette attaque justifie l'utilisation du **Triple-DES** qui consiste à réaliser un *surchiffrement triple* avec le chiffrement DES. La taille de la clé est alors de $3 \times 56 = 168$ bits mais une adaptation de l'attaque par compromis temps-mémoire de l'exercice précédent montre qu'il est possible de retrouver la clé en environ 2^{112} chiffrements DES sous une attaque à clairs connus. Pour cette raison, P.C. VAN OORSCHOT et M. WIENER ont proposé d'utiliser le **Triple-DES** avec seulement deux clés indépendantes $(K, K^*) \in \{0, 1\}^{56} \times \{0, 1\}^{56}$ sous la forme suivante :

$$\text{Triple-DES}_{K, K^*}(X) = \text{DES}_K(\text{DES}_{K^*}^{-1}(\text{DES}_K(X)))$$

Le but de l'exercice suivant est de montrer qu'il existe une attaque contre cette variante en utilisant encore une fois une collision interne.

Exercice 2.12 Chiffrement Triple-DES avec deux clés indépendantes

Nous considérons le chiffrement d'un bloc P de 64 bits avec le **Triple-DES** utilisant seulement deux clés indépendantes $(K, K^*) \in \{0, 1\}^{56} \times \{0, 1\}^{56}$ sous la forme

$$C = \text{DES}_K(\text{DES}_{K^*}^{-1}(\text{DES}_K(P)))$$

1. Considérons la liste \mathcal{L} des valeurs (K_i, P_i) avec $K_i \in \{0, 1\}^{56}$ et $P_i = \text{DES}_{K_i}^{-1}(0^{64})$. Nous supposons connus pour tous les P_i de cette liste le chiffré de P_i :

$$C_i = \text{Triple-DES}_{K, K^*}(P_i)$$

Montrer que pour la clé $K_{i_0} = K$, il existe au moins une entrée $(K_j, \text{DES}_{K_{i_0}}^{-1}(C_{i_0}))$ dans la liste \mathcal{L} et la clé K^* est une des clés K_j ainsi obtenues.

2. En déduire une attaque à 2^{56} chiffrés choisis qui retrouve le couple de clés (K, K^*) en effectuant environ 2^{57} chiffrements DES.

Solution

- Pour l'indice i_0 tel que $K_{i_0} = K$, nous avons

$$\begin{aligned} C_{i_0} &= \text{DES}_K(\text{DES}_{K^*}^{-1}(\text{DES}_K(P_{i_0}))) \\ &= \text{DES}_K(\text{DES}_{K^*}^{-1}(\text{DES}_K(\text{DES}_K^{-1}(0^{64})))) \\ &= \text{DES}_K(\text{DES}_{K^*}^{-1}(0^{64})) \end{aligned}$$

En notant $B_i = \text{DES}_{K_i}^{-1}(C_i)$ pour tout $K_i \in \{0, 1\}^{56}$, nous avons $B_{i_0} = \text{DES}_{K^*}^{-1}(0^{64})$ et K^* est donc bien l'une des clés K_j telles que $(K_j, B_{i_0}) \in \mathcal{L}$.

- Dans un premier temps, l'attaque consiste à construire la liste (triée) \mathcal{L} en 2^{56} évaluations du chiffrement DES. L'attaquant demande ensuite le chiffrement Triple-DES avec deux clés indépendantes de tous les chiffrés P_i et stocke les 2^{56} valeurs

$$C_i = \text{DES}_K(\text{DES}_{K^*}^{-1}(\text{DES}_K(P_i)))$$

Il calcule ensuite $B_i = \text{DES}_{K_i}^{-1}(C_i)$ pour tout $K_i \in \{0, 1\}^{56}$ en 2^{56} nouvelles évaluations du chiffrement DES et recherche les clés K_j telles que $(K_j, B_i) \in \mathcal{L}$. Par le paradoxe des anniversaires, le nombre de clés attendues est d'environ :

$$\frac{2^{56} \times 2^{56}}{2^{64}} = 2^{48}$$

Pour filtrer les fausses alertes, il suffit d'utiliser un autre couple clair/chiffré connu. Le nombre de fausses alertes vérifiant cette deuxième équation est en moyenne égal à $2^{48}/2^{64} = 2^{-16}$. En effectuant (en moyenne) 2^{48} chiffrements Triple-DES, nous retrouvons finalement la vraie clé (K, K^*) . L'attaque nécessite 2^{56} chiffrés choisis, $2^{57} + 3 \cdot 2^{48} \approx 2^{57}$ chiffrements DES et une mémoire de 2^{56} fois $56 + 64 + 64 = 184$ bits (soit environ 2^{64} bits).

Une autre proposition pour renforcer la sécurité de DES face aux recherches exhaustives de la clé consiste à utiliser plusieurs modes opératoires en cascade, comme par exemple, le mode CBC-CBC-CBC et toutes les variantes possibles combinant les modes CBC, ECB et CFB. Malheureusement, E. BIHAM a montré que tous ces modes peuvent être attaqués plus efficacement que le Triple-DES. L'exercice suivant montre un exemple d'attaque sur le mode CBC-CBC-ECB (qui utilise deux vecteurs d'initialisation indépendants).

Exercice 2.13 Mode opératoire CBC-CBC-ECB

Nous considérons le mode opératoire triple CBC-CBC-ECB décrit dans la figure 2.9. Nous allons mener une attaque à chiffrés choisis pour retrouver les clés K_1 , K_2 et K_3 utilisées.

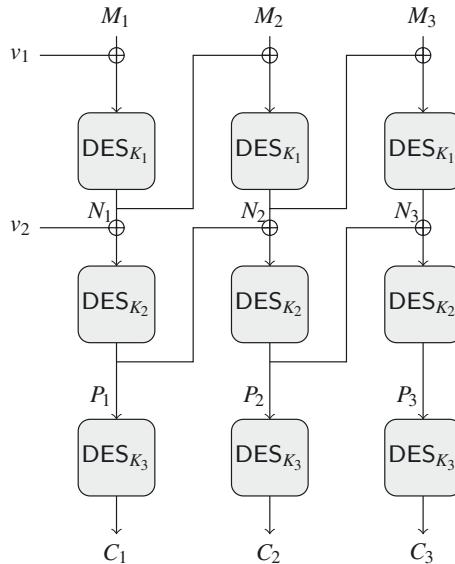


Figure 2.9 – Description du mode opératoire CBC-CBC-ECB

1. Considérons deux chiffrés $C = (v_1, v_2, C_1, C_2, C_3)$ et $C' = (v'_1, v'_2, C'_1, C'_2, C'_3)$ dont les deux derniers blocs sont égaux ; notons $M = (M_1, M_2, M_3)$ et $M' = (M'_1, M'_2, M'_3)$ les clairs correspondants. En notant (N_1, N_2, N_3) et (N'_1, N'_2, N'_3) les sorties du premier chiffrement par bloc et (P_1, P_2, P_3) et (P'_1, P'_2, P'_3) les entrées du troisième chiffrement par bloc, montrer que

$$N_2 \oplus M_3 = N'_2 \oplus M'_3$$

et

$$N_2 \oplus P_1 = N'_2 \oplus P'_1$$

2. En déduire une attaque à chiffrés choisis qui permet de retrouver la clé K_3 par une recherche exhaustive de K_3 (indépendamment des autres clés).
3. Montrer comment retrouver K_1 et K_2 une fois K_3 découverte.

Solution

1. Puisque $C_2 = C'_2$, nous avons

$$P_2 = \text{DES}_{K_3}^{-1}(C_2) = \text{DES}_{K_3}^{-1}(C'_2) = P'_2$$

et les entrées du deuxième chiffrement sont donc égales, d'où

$$N_2 \oplus P_1 = N'_2 \oplus P'_1$$

De même, puisque $C_3 = C'_3$, nous avons

$$P_3 = \text{DES}_{K_3}^{-1}(C_3) = \text{DES}_{K_3}^{-1}(C'_3) = P'_3$$

En combinant les deux égalités, nous en déduisons que, pour les deux chiffrés, les sorties du premier chiffrement sont égales et nous avons $N_3 = N'_3$ et donc

$$N_2 \oplus M_3 = N'_2 \oplus M'_3$$

2. En combinant les deux égalités de la question précédente, nous obtenons

$$N_2 \oplus N'_2 = M_3 \oplus M'_3 = P_1 \oplus P'_1$$

et comme $P_1 = \text{DES}_{K_3}^{-1}(C_1)$ et $P'_1 = \text{DES}_{K_3}^{-1}(C'_1)$, nous obtenons un test d'arrêt pour une recherche exhaustive de K_3 (qui demande 2^{56} chiffrements DES et seulement deux chiffrés choisis).

3. Lorsque la valeur de K_3 est connue, il est facile de déterminer les valeurs de K_1 et K_2 en adaptant l'attaque sur le double chiffrement de l'exercice 2.11 en (environ) 2^{57} chiffrements DES et une mémoire de 2^{64} bits environ.

2.4 CHIFFREMENT AES

Comme les schémas de Feistel, les *réseaux de substitutions-permutations* sont des systèmes de chiffrement itératif. À chaque tour, l'algorithme applique au bloc d'entrée une substitution non linéaire suivie d'une fonction (généralement linéaire) appelée permutation.

En 1997, la sécurité du DES n'était plus garantie face à une recherche exhaustive de la clé désormais envisageable et le chiffrement Triple-DES était jugé trop lent. Le *National Institute of Standards and Technology (NIST)* – une agence du département du commerce des États-Unis – a donc lancé un processus de standardisation appelé *Advanced Encryption Standard process* pour concevoir un nouvel algorithme de chiffrement par bloc. Le système qui a été retenu à la fin du processus en 2000 est l'algorithme Rijndael qui a été conçu par J. DAEMEN et V. RIJMEN.

La version de Rijndael standardisée par le NIST, appelé *chiffrement AES*, est un chiffrement par bloc de 128 bits de type réseau de substitutions-permutations. Les clés peuvent être de 128, 192 ou 256 bits mais dans ce livre nous ne considérerons que la variante avec des clés de 128 bits. Le chiffrement découpe chaque bloc de 128 bits en 16 octets et chaque bloc est représenté sous forme d'une matrice carrée 4×4 où les octets sont numérotés de 1 à 16 gauche à droite et de bas en haut.

Chapitre 2 • Chiffrement par bloc

Ce bloc est initialement le bloc à chiffré auquel on additionne bit à bit une clé de tour initiale K_0 . Le chiffrement AES applique ensuite de façon répétée quatre procédures appelées SubBytes, ShiftRows, MixColumns et AddRoundKey sur la matrice carrée 4×4 appelée *matrice état* (cf. Figure 2.10).

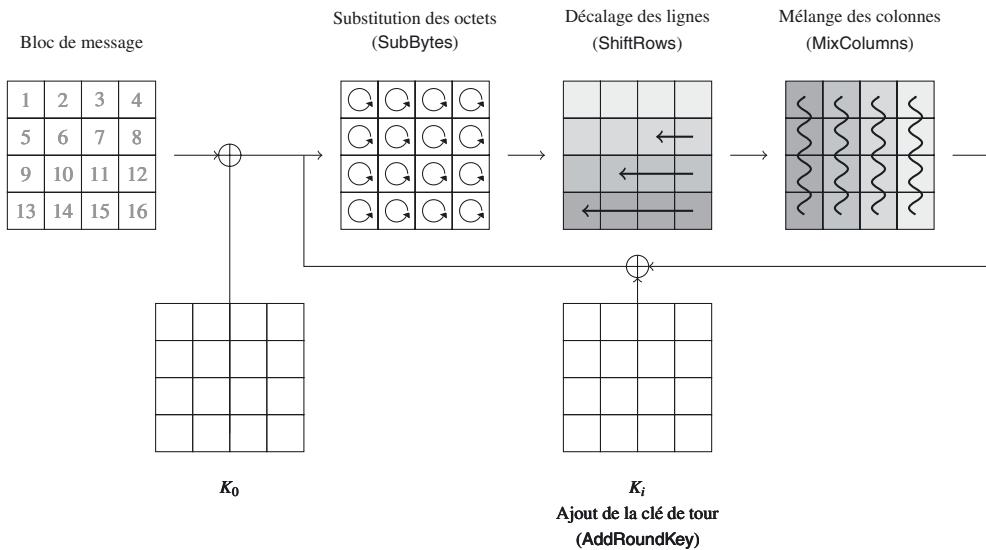


Figure 2.10 - Description haut niveau de AES

- La procédure « substitution des octets » (SubBytes) applique sur chacun des seize octets de l'état interne une S-boîte, assure une bonne confusion et a pour but de faire disparaître les structures tant linéaires qu'algébriques du chiffrement.
- La procédure « décalage de lignes » (ShiftRows) effectue des rotations vers la gauche sur les lignes de la matrice état. Le décalage pour chaque ligne est différent : la première ligne n'est pas modifiée, la seconde est décalée d'une case vers la gauche, la troisième ligne de deux cases et la dernière de trois cases.
- La procédure « mélange des colonnes » (MixColumns) combine les 4 octets de chaque colonne de la matrice état en appliquant une transformation linéaire. Le but de ces deux dernières opérations est d'assurer une bonne diffusion de l'information.
- La procédure « ajout de la clé de tour » (AddRoundKey) combine l'état interne avec une clé différente pour chaque tour.

Le nombre de tours dépend de la clé et dans le cas d'une clé de 128 bits, il est égal à 10. La procédure MixColumns est omise lors du dernier tour. La clé de tour K_i est

un bloc de 128 bits différent à chaque tour. Cette suite de clés est engendrée par un algorithme de diversification de clé à partir de la clé initiale.

Pour effectuer ces différentes opérations, chaque octet est traité comme un élément du corps fini $\mathbb{F}_{256} = \mathbb{F}_{2^8}$ en identifiant la suite des bits (du poids fort au poids faible) avec la suite des coefficients (classés par ordre de puissance décroissante) d'un polynôme de degré 7, qui représente un élément de \mathbb{F}_{2^8} . La somme de deux éléments est la somme bit à bit des vecteurs et leur produit est le produit polynomial modulo le polynôme irréductible de degré 8 :

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

La S-boîte de l'AES est calculée en déterminant l'inverse de l'entrée dans le corps $\mathbb{F}_{256} = \mathbb{F}_{2^8}$ construit comme $\mathbb{F}_{256} = \mathbb{F}_2[x]/m(x)$ (comme 0 n'a pas d'inverse dans le corps, on lui associe la valeur 0) puis en lui appliquant la transformation affine suivante :

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

où (x_7, \dots, x_0) est l'octet représentant l'inverse multiplicatif de l'entrée (vu comme un vecteur de \mathbb{F}_2^8).

Exercice 2.14 (avec programmation). S-Boîte de l'AES

1. Calculer l'image de 11011001 par la S-boîte de l'AES.
2. En généralisant cette méthode, construire (par informatique) la table complète de la S-boîte de l'AES.
3. Calculer l'inverse de la transformation affine de la S-boîte de l'AES et donner la table complète de l'inverse de la S-boîte.

Solution

1. Dans \mathbb{F}_{256} , le vecteur 11011001 correspond au polynôme $\alpha = x^7 + x^6 + x^4 + x^3 + 1 \in \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$. En appliquant l'algorithme d'Euclide étendu pour calculer le PGCD de α et $x^8 + x^4 + x^3 + x + 1$, nous obtenons

$$(x^7 + x^3 + x + 1)\alpha + (x^6 + x^5 + x^3 + x^2 + x)(x^8 + x^4 + x^3 + x + 1) = 1$$

et l'inverse de α dans \mathbb{F}_{256} est donc $x^7 + x^3 + x + 1$.

En interprétant α^{-1} comme un vecteur de \mathbb{F}_2^8 , nous obtenons

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) = (1, 0, 0, 0, 1, 0, 1, 1)$$

et en appliquant la transformation affine :

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

et l'image de α par la S-boîte de l'AES est 00110101.

2. Un calcul exhaustif par informatique donne le tableau 2.2, où la valeur de la S-boîte en un octet est obtenue à l'intersection de la ligne formée par ses quatre premiers bits et de la colonne formée par ses quatre derniers bits (interprétés en hexadécimal). Ainsi on retrouve bien que la valeur de la S-boîte en 11011001 est 35 = 00110101 qui se situe à l'intersection de la ligne D = 1101 et de la colonne 9 = 1001.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tableau 2.2 – Table de la S-boîte de AES

3. De l'algèbre linéaire élémentaire dans \mathbb{F}_2^8 donne la transformation affine inverse suivante :

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Un calcul exhaustif par informatique donne le tableau 2.3, où la valeur de l'inverse de la S-boîte en un octet est encore obtenue à l'intersection de la ligne formée par ses quatre premiers bits et de la colonne formée par ses quatre derniers bits (interprétés en hexadécimal). Ainsi on retrouve que la valeur de l'inverse de la S-boîte en 00110101 est D9 = 11011001 qui se situe à l'intersection de la ligne 3 = 0011 et de la colonne 9 = 0101.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Tableau 2.3 – Table de l'inverse de la S-boîte de AES

Pour l'opération MixColumns, chaque vecteur colonne de la matrice état est vu comme un élément de l'anneau $A = \mathbb{F}_{2^8}[X]/(X^4 + 1)$. Nous utiliserons l'élément $a(X)$ de A défini par :

$$\begin{aligned} a(X) &= \{03\}X^3 + \{01\}X^2 + \{01\}X + \{02\} \\ &= (x + 1)X^3 + X^2 + X + x \end{aligned}$$

Étant donnée une colonne de la matrice état vue comme un élément $b(X)$ de l'anneau A , le résultat de l'opération MixColumns sur b est égal au vecteur $a(X) \cdot b(X)$.

Exercice 2.15 (avec programmation). Opération MixColumns

1. Considérons deux polynômes de $A[X]$:

$$\begin{aligned} P(X) &= p_3X^3 + p_2X^2 + p_1X + p_0 \\ Q(X) &= q_3X^3 + q_2X^2 + q_1X + q_0 \end{aligned}$$

Montrer que modulo $X^4 + 1$, nous avons

$$P(X) \cdot Q(X) \equiv \sum_{k=0}^3 \sum_{i+j \equiv k \pmod{3}} p_i q_j X^k$$

2. Montrer que (la classe) de $a(X)$ est inversible dans A et calculer son inverse.
3. Donner l'expression matricielle du produit par $a(X)$ et par $a(X)^{-1}$.

Solution

1. Nous avons

$$P(X) \cdot Q(X) = \sum_{k=0}^6 \sum_{i+j=k} p_i q_{i-j} X^i$$

Le résultat découle immédiatement en remarquant que $X^4 = -1 = 1$ dans l'anneau A .

2. Nous recherchons un élément de A

$$\alpha_3 X^3 + \alpha_2 X^2 + \alpha_1 X + \alpha_0$$

s'il existe, tel que

$$a(X)(\alpha_3 X^3 + \alpha_2 X^2 + \alpha_1 X + \alpha_0) = 1$$

D'après la question précédente, nous obtenons le système linéaire suivant

$$\begin{cases} \alpha_0\{02\} + \alpha_1\{03\} + \alpha_2\{01\} + \alpha_3\{01\} = 1 \\ \alpha_0\{01\} + \alpha_1\{02\} + \alpha_2\{03\} + \alpha_3\{01\} = 0 \\ \alpha_0\{01\} + \alpha_1\{01\} + \alpha_2\{02\} + \alpha_3\{03\} = 0 \\ \alpha_0\{03\} + \alpha_1\{01\} + \alpha_2\{01\} + \alpha_3\{02\} = 0 \end{cases}$$

Dans \mathbb{F}_{256} , nous obtenons le système

$$\begin{cases} \alpha_0 x + \alpha_1(x+1) + \alpha_2 + \alpha_3 = 1 \\ \alpha_0 + \alpha_1 x + \alpha_2(x+1) + \alpha_3 = 0 \\ \alpha_0 + \alpha_1 + \alpha_2 x + \alpha_3(x+1) = 0 \\ \alpha_0(x+1) + \alpha_1 + \alpha_2 + \alpha_3 x = 0 \end{cases}$$

qui donne

$$\begin{cases} \alpha_0 = x^3 + x^2 + x \\ \alpha_1 = x^3 + 1 \\ \alpha_2 = x^3 + x^2 + 1 \\ \alpha_3 = x^3 + x + 1 \end{cases}$$

Nous obtenons donc que $a(X)$ est inversible dans A et que son inverse est

$$a(X)^{-1} = \{0B\}X^3 + \{0D\}X^2 + \{09\}X + \{0E\}$$

3. D'après la première question, nous obtenons que l'expression matricielle du produit par $a(X)$ d'un élément $s(X) = s_3X^3 + s_2X^2 + s_1X + s_0$ et $r(X) = a(X) \cdot s(X) = r_3X^3 + r_2X^2 + r_1X + r_0$ avec

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

et l'expression matricielle du produit par $a(X)^{-1}$ est

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 09 \\ 09 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 09 & 14 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix}$$

Nous posons

$$M = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \text{ et } M^{-1} = \begin{bmatrix} 14 & 11 & 13 & 09 \\ 09 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 09 & 14 \end{bmatrix}$$

Dans l'exercice suivant, nous utiliserons sans le démontrer², le fait que tous les mineurs de la matrice M (*i.e.* les déterminants de ses sous-matrices) sont non nuls.

Exercice 2.16 Propriétés de l'opération MixColumns

Considérons deux vecteurs colonnes v_0 et v_1 de \mathbb{F}_{256} ⁴.

- Montrer que si v_0 et v_1 diffèrent en seulement un octet, alors $\text{MixColumns}(v_0)$ et $\text{MixColumns}(v_1)$ diffèrent en leurs quatre octets.
- Montrer que si v_0 et v_1 diffèrent en leurs quatre octets, alors $\text{MixColumns}(v_0)$ et $\text{MixColumns}(v_1)$ diffèrent en au moins un octet.
- Montrer que si v_0 et v_1 diffèrent en exactement deux octets (*resp.* exactement trois octets), alors $\text{MixColumns}(v_0)$ et $\text{MixColumns}(v_1)$ diffèrent en au moins trois octets (*resp.* au moins deux octets).

Solution

- Par linéarité de l'opération MixColumns, il suffit de montrer que si l'on applique MixColumns à un vecteur v de \mathbb{F}_{256} formé d'un octet non nul $\alpha \in \mathbb{F}_{256}^*$ en position
- Cette propriété peut être vérifiée aisément par informatique.

$i \in \{1, 2, 3, 4\}$ et de trois octets nuls, alors $\text{MixColumns}(v)$ est constitué uniquement d'octets non nuls. Le vecteur $\text{MixColumns}(v)$ est simplement la i -ième colonne de la matrice M multipliée par α et dans tous les cas, il est bien formé uniquement d'octets non nuls.

2. Il suffit de remarquer que la matrice M est inversible et l'image d'un vecteur non nul est nécessairement un vecteur non nul.
3. Considérons un vecteur v de \mathbb{F}_{256}^4 non nul en seulement deux positions notées j_0 et j_1 de $\{1, 2, 3, 4\}$. Supposons par l'absurde que $\text{MixColumns}(v)$ est nul en deux positions $i_0, i_1 \in \{1, 2, 3, 4\}$. En considérant la sous-matrice de M formé des colonnes j_0 et j_1 et des lignes i_0 et i_1 nous obtenons le système linéaire homogène suivant

$$\begin{bmatrix} M_{i_0, j_0} & M_{i_0, j_1} \\ M_{i_1, j_0} & M_{i_1, j_1} \end{bmatrix} \begin{bmatrix} v_{j_0} \\ v_{j_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

D'après la propriété sur les mineurs de M , la matrice du système est inversible ce qui contredit le fait que v_{j_0} et v_{j_1} sont non nuls. Par linéarité, nous en déduisons que si v_0 et v_1 diffèrent en exactement deux octets, alors $\text{MixColumns}(v_0)$ et $\text{MixColumns}(v_1)$ diffèrent en au moins trois octets. La propriété lorsque v_0 et v_1 diffèrent en exactement trois octets se montre de façon analogue en utilisant le fait que les sous-matrices 2×3 de M sont de rang 2.

La diversification de clé de l'AES produit un vecteur (w_0, \dots, w_{43}) où w_i est un mot de quatre octets pour $i \in \{0, \dots, 43\}$ et la clé K_i , utilisée où tour i , est formée des vecteurs colonnes $(w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3})$ (pour $i \in \{1, \dots, 10\}$). Elle s'effectue de la façon suivante :

- La clé K_0 est simplement la clé K , autrement dit, les vecteurs w_0, \dots, w_3 sont les vecteurs colonnes de la clé K ;
- La clé K_i formée des vecteurs colonnes $(w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3})$ est obtenue à partir de la clé K_{i-1} , des constantes

$$\text{Rcon}_i = (\{02\}^{254+i}, \{00\}, \{00\}, \{00\})$$

en appliquant la fonction **SubBytes** et des rotations **RotWord** avec

$$\text{SubWord}(a_0, a_1, a_2, a_3) = (S(a_0), S(a_1), S(a_2), S(a_3))$$

$$\text{RotWord}(a_0, a_1, a_2, a_3) = (a_1, a_2, a_3, a_0)$$

Nous avons alors

$$w_{4i} = \text{SubWord}(\text{RotWord}(w_{4i-1})) \oplus \text{Rcon}_i \oplus w_{4i-1}$$

$$w_{4i+1} = w_{4i} \oplus w_{4i-3}$$

$$w_{4i+2} = w_{4i+1} \oplus w_{4i-2}$$

$$w_{4i+3} = w_{4i+2} \oplus w_{4i-1}$$

Exercice 2.17 (avec programmation). Diversification de clé de l'AES

1. Calculer les valeurs Rcon_i pour $i \in \{1, \dots, 10\}$.
2. Calculer la clé K_1 utilisée dans le premier tour du chiffrement AES 128-bits si la clé maître utilisée est

$$K = 00\ 11\ 22\ 33\ 44\ 55\ 66\ 77\ 88\ 99\ AA\ BB\ CC\ DD\ EE\ FF$$
3. Calculer la clé étendue AES formée de 176 octets à partir de la clé maître K de la question précédente.

Solution

1. Nous avons $\text{Rcon}_1[0] = \{02\}^{255} = x^{255} = 1 = \{01\}$. Pour $i \in \{2, \dots, 8\}$, nous obtenons immédiatement pour le premier octet des constantes Rcon :

$$\begin{aligned}\text{Rcon}_2[0] &= \{02\}^{256} = x^{256} = x = \{02\} \\ \text{Rcon}_3[0] &= \{02\}^{257} = x^2 = \{04\} \\ \text{Rcon}_4[0] &= \{02\}^{258} = x^3 = \{06\} \\ \text{Rcon}_5[0] &= \{02\}^{259} = x^4 = \{08\} \\ \text{Rcon}_6[0] &= \{02\}^{260} = x^5 = \{10\} \\ \text{Rcon}_7[0] &= \{02\}^{261} = x^6 = \{20\} \\ \text{Rcon}_8[0] &= \{02\}^{262} = x^7 = \{40\}\end{aligned}$$

Pour $i = 9$, nous avons

$$\text{Rcon}_9[0] = \{02\}^8 = x^8 = x^4 + x^3 + x + 1 = \{1B\}$$

Enfin, pour $i = 10$, nous obtenons

$$\text{Rcon}_{10}[0] = \{02\}^9 = x^9 = x(x^4 + x^3 + x + 1) = \{36\}$$

Les octets $\text{Rcon}_i[j]$ pour $i \in \{1, \dots, 10\}$ et $j \in \{1, 2, 3\}$ sont égaux à $\{00\}$ par définitions.

2. Nous avons

$$K_0 = 00\ 11\ 22\ 33\ 44\ 55\ 66\ 77\ 88\ 99\ AA\ BB\ CC\ DD\ EE\ FF$$

En posant $w_0 = 00\ 44\ 88\ CC$ et $w_3 = 33\ 77\ BB\ FF$, nous avons

$$T_1 = \text{RotWord}(w_3) = 77\ BB\ FF\ 33$$

puis

$$T_2 = \text{SubWord}(T_1) = F5\ EA\ 16\ C3$$

Chapitre 2 • Chiffrement par bloc

En ajoutant Rcon₁, nous obtenons

$$T_2 \oplus \text{Rcon}_1 = \text{F4 EA 16 C3}$$

et enfin

$$\begin{aligned} w_4 &= \text{F4 EA 16 C3} \oplus w_0 \\ &= \text{F4 EA 16 C3} \oplus \text{00 44 88 CC} \\ &= \text{F4 AE 9E 0F} \end{aligned}$$

Nous avons ensuite

$$w_5 = w_1 \oplus w_4 = \text{E5 FB 07 D2}$$

$$w_6 = w_2 \oplus w_5 = \text{C7 9D AD 3C}$$

$$w_7 = w_3 \oplus w_6 = \text{F4 EA 16 C3}$$

et la clé K₁ est égale à

$$K_1 = \text{F4 E5 C7 F4 AE FB 9D EA 9E 07 AD 16 0F D2 3C C3}$$

3. Un calcul exhaustif par informatique donne les clés de tour suivantes :

00	11	22	33
44	55	66	77
88	99	AA	BB
CC	DD	EE	FF

Clé K₀

F4	E5	C7	F4
AE	FB	9D	EA
9E	07	AD	16
0F	D2	3C	C3

Clé K₁

71	94	53	A7
E9	12	8F	65
B0	B7	1A	0C
B0	62	5E	9D

Clé K₂

38	AC	FF	58
17	05	8A	EF
EE	59	43	4F
EC	8E	D0	4D

Clé K₃

EF	43	BC	E4
93	96	1C	F3
0D	54	17	58
86	08	D8	95

Clé K₄

F2	B1	0D	E9
F9	6F	73	80
27	73	64	3C
EF	E7	3F	AA

Clé K₅

1F	AE	A3	4A
12	7D	0E	8E
8B	F8	9C	A0
F1	16	29	83

Clé K₆

46	E8	4B	01
F2	8F	81	0F
67	9F	03	A3
27	31	18	9B

Clé K₇

B0	58	13	12
F8	77	F6	F9
73	EC	EF	4C
5B	6A	72	E9

Clé K₈

32	6A	79	6B
D1	A6	50	A9
6D	81	6E	22
92	F8	8A	63

Clé K₉

D7	BD	C4	AF
42	E4	B4	1D
96	17	79	5B
ED	15	9F	FC

Clé K₁₀

En particulier, nous avons K₀ = K et nous retrouvons la valeur de K₁ de la question précédente.

FONCTIONS DE HACHAGE – TECHNIQUES AVANCÉES DE CRYPTANALYSE

Les fonctions de hachage cryptographiques sont des fonctions qui associent à un message de longueur arbitraire une valeur de longueur fixe appelée *empreinte* du message. Différentes données numériques donneront presque toujours des empreintes différentes et l'empreinte ne contient pas suffisamment d'information pour permettre la reconstitution du texte original. Le but d'une empreinte n'est donc pas de transporter de l'information mais seulement de représenter une donnée particulière. Les fonctions de hachage sont utilisées dans tous les domaines de la cryptographie à commencer par les techniques d'authentification (codes d'authentification de messages, signatures numériques, protection de mots de passe) mais également pour renforcer la sécurité d'autres primitives cryptographiques (comme le chiffrement).

Dans la première partie de ce chapitre, nous étudierons les propriétés de sécurité des fonctions de hachage cryptographique (résistance à la (seconde) pré-image et résistance aux collisions). Nous étudierons les techniques de construction de fonction de hachage cryptographique à partir de système de chiffrement par bloc. Nous étudierons également des attaques génériques sur les fonctions de hachage itérées reposant sur la construction de Merkle-Damgård.

Nous avons vu qu'un moyen pour attaquer un système de chiffrement par bloc itératif est de construire un distingueur à partir d'une relation portant sur certains tours, puis, d'utiliser ce distingueur, pour faire des tests d'hypothèse sur tout ou partie des clés des autres tours. Dans la deuxième partie du chapitre, nous étudierons trois grandes familles d'attaque qui ont été proposées au cours des trente dernières années contre les systèmes de chiffrement par bloc (et qui peuvent également parfois s'appliquer pour attaquer des fonctions de hachage) :

- la *cryptanalyse différentielle* (de E. BIHAM et A. SHAMIR) qui consiste en l'analyse des modifications produites par des différences entre les textes clairs sur les différences des textes chiffrés associés ;
- la *cryptanalyse linéaire* (de M. MATSUI) qui consiste à utiliser une approximation linéaire de l'algorithme de chiffrement et à analyser les corrélations entre les entrées et les sorties obtenues par cette approximation ;

- les *attaques par saturation* (de D. WAGNER et L. KNUDSEN) qui étudient le comportement d'un système lorsque l'on chiffre une famille de messages de telle sorte que certaines de ses composantes prennent toutes les valeurs possibles.

3.1 GÉNÉRALITÉS SUR LES FONCTIONS DE HACHAGE

En fonction de l'application, les propriétés de sécurité que l'on attend d'une fonction de hachage \mathcal{H} sont :

- la résistance à la pré-image* : étant donnée une empreinte h , il doit être calculatoirement difficile de trouver un message m tel que $\mathcal{H}(m) = h$;
- la résistance à la seconde pré-image* : étant donné un message m , il doit être calculatoirement difficile de trouver un message $m' \neq m$ tel que $\mathcal{H}(m') = \mathcal{H}(m)$;
- la résistance aux collisions* : il doit être calculatoirement difficile de trouver deux messages m et m' différents tels que $\mathcal{H}(m') = \mathcal{H}(m)$.

Les définitions formelles de ces notions de sécurité sont en dehors du cadre de ce livre mais il est important de noter que la résistance aux collisions n'a de sens mathématique que lorsqu'on considère une famille de fonctions de hachage et pas une fonction de hachage unique. Notons que l'existence de pré-images, de secondes pré-images et de collisions est inévitable, en raison des tailles des ensembles de départ et d'arrivée d'une fonction de hachage.

Exercice 3.1 Résistance à la pré-image et aux collisions

Montrer qu'une fonction de hachage résistante aux collisions n'est pas nécessairement résistante à la pré-image.

Solution

Supposons qu'il existe une fonction $g : \{0, 1\}^* \rightarrow \{0, 1\}^n$ résistante aux collisions pour un entier $n \in \mathbb{N}$. Nous allons construire à partir de g une autre fonction résistante aux collisions notée h (sous l'hypothèse que g l'est) mais pour laquelle la recherche de pré-image est polynomiale en n pour une proportion significative de son image.

Il suffit de définir h de la façon suivante :

$$h : \{0, 1\}^* \longrightarrow \{0, 1\}^{n+1}$$

$$x \longmapsto \begin{cases} 0\|x & \text{si } |x| = n \\ 1\|g(x) & \text{sinon} \end{cases}$$

La fonction h est bien résistante aux collisions. En effet, soient x et x' deux chaînes de bits distinctes telles que $h(x) = h(x')$. Nécessairement x et x' ne sont pas de longueur n . En effet, si c'était le cas, nous aurions $h(x) = h(x')$, soit $0\|x = 0\|x'$ et donc $x = x'$. Il est également impossible que x ou bien x' soit de longueur n car dans ce cas les empreintes

commenceraient par des bits différents (et ne seraient donc pas égales). Nous avons donc $h(x) = 1\|g(x)$ et $h(x') = 1\|g(x')$ et le couple (x, x') est une collision pour la fonction g .

Cependant, la fonction h n'est pas résistante à la recherche de pré-image. En effet, un élément aléatoire $y = (y_1, \dots, y_{n+1})$ de $\{0, 1\}^{n+1}$, l'image de h , commence par un $y_1 = 0$ avec une probabilité $1/2$. Pour un tel élément, un antécédent est obtenu de façon immédiate comme les n derniers bits de y , $x = (y_2 \dots y_{n+1})$.

Une fonction de hachage résistante aux collisions n'est pas nécessairement résistante à la pré-image.

Les méthodes de construction de fonctions de hachage cryptographiques sont empiriques. La plus utilisée (due à R. MERKLE [47] et I. DAMGÅRD [17]) consiste à itérer une *fonction de compression* $f : \{0, 1\}^{n+\ell} \rightarrow \{0, 1\}^n$ en découplant le message en blocs et en appliquant cette fonction successivement à chaque bloc et au résultat de la compression du bloc précédent (*cf.* Figure (3.1)).

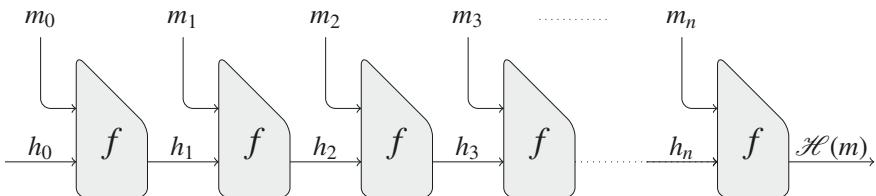


Figure 3.1 - Construction de Merkle-Damgaard

Considérons une *fonction de compression* $f : \{0, 1\}^{n+\ell} \rightarrow \{0, 1\}^n$ avec $\ell \geq 2$ et $h_0 \in \{0, 1\}^n$ un vecteur d'initialisation (utilisée pour le premier bloc). Nous considérons un processus de remplissage \mathcal{R} défini sur $\{0, 1\}^*$ et vérifiant $|\mathcal{R}(m)| \equiv 0 \pmod{\ell}$ pour tout message $m \in \{0, 1\}^*$ (*i.e.* qui transforme le message à hacher en un message dont la longueur est un multiple de ℓ). La valeur de la fonction de hachage itérée \mathcal{H} construite à partir de f et R est définie par $\mathcal{H}(m) = f(h_k, m_k)$ où

- la valeur $\mathcal{R}(m)$ est divisée en $(k + 1)$ blocs de ℓ bits $\mathcal{R}(m) = (m_0, \dots, m_k) \in (\{0, 1\}^\ell)^{k+1}$;
- $h_i = f(h_{i-1}, m_{i-1})$ pour tout $i \in \{1, \dots, k\}$.

R. MERKLE et I. DAMGÅRD ont montré que si la fonction de compression est résistante aux collisions et si le processus de bourrage est bien construit, alors la fonction de hachage itérée \mathcal{H} obtenue à partir de f est résistante aux collisions.

Exercice 3.2 Construction de Merkle-Damgaard

1. Supposons que les messages dont la longueur n'est pas un multiple de la longueur du bloc ℓ sont complétés par une chaîne de zéros jusqu'à ce que la longueur soit un multiple de ℓ (*i.e.* en posant $i = |m| \bmod \ell$, $\mathcal{R}(m) = m||0^{\ell-i}$).

Montrer que la fonction itérée obtenue à partir de f et R n'est pas résistante aux collisions.

2. Supposons désormais que le processus de bourrage est défini de la façon suivante :

$$\mathcal{R}(m) = m \| 10^{\ell-i-1} \text{ avec } i = |m| \bmod \ell$$

Montrer que si l'on dispose d'un bloc de message z tel que $f(h_0, z) = h_0$ alors il est possible de trouver des collisions pour la fonction itérée obtenue à partir de f et R .

3. Supposons enfin qu'un dernier bloc contenant la longueur binaire du message est concaténé au procédé de bourrage de la question précédente (*i.e.* en notant τ_m un encodage binaire de la longueur $|m|$ de m , nous avons $\mathcal{R}(m) = (m \| 10^{\ell-i-1} \| \tau_m)$ avec $i = |m| \bmod \ell$).

Montrer que la fonction itérée obtenue à partir de f et R est résistante aux collisions si f est résistante aux collisions.

Solution

1. Il suffit de remarquer par exemple que les deux messages binaires $m = 1$ et $m' = 10$ et de longueur respectives 1 et 2 sont tous les deux complétés de la même façon par le procédé de bourrage suggéré, à savoir le message $\tilde{m} = 10\dots0 = 10^{\ell-1}$. Leurs images par la fonction de hachage itérée sont donc égales et la fonction n'est pas résistante aux collisions.
2. Il suffit de remarquer que pour un tel bloc z , nous avons $\mathcal{R}(z\|m) = z\|\mathcal{R}(m)$ pour tout message $m \in \{0, 1\}^*$ et puisque $f(h_0, z) = h_0$, la valeur h_1 dans le calcul de $\mathcal{H}(z\|m)$ est égale à h_0 et nous avons $\mathcal{H}(z\|m) = \mathcal{H}(m)$ pour tout message $m \in \{0, 1\}^*$.
3. Considérons deux messages distincts m et m' tels que $\mathcal{H}(m) = \mathcal{H}(m')$. Notons k et k' le nombre de blocs de ℓ bits de $\mathcal{R}(m)$ et $\mathcal{R}(m')$ et h_i et h'_j (avec $i \in \{0, \dots, k\}$ et $j \in \{0, \dots, k'\}$) les valeurs de chaînage successives lors du calcul des hachés (avec $h_0 = h'_0$).

Si m et m' ne sont pas de même longueur binaire, nous avons $\tau_m \neq \tau_{m'}$ et $f(h_k, \tau_m) = \mathcal{H}(m) = \mathcal{H}(m') = f(h'_{k'}, \tau_{m'})$ et nous avons une collision explicite pour la fonction de compression f .

Si m et m' sont de la même longueur, nous avons $k = k'$ et $\tau_m = \tau_{m'}$. Notons $\alpha \in \{0, \dots, k-1\}$ le plus petit entier tel que $m_\alpha \neq m'_\alpha$ où $\mathcal{R}(m) = (m_0, \dots, m_k)$ et $\mathcal{R}(m') = (m'_0, \dots, m'_k)$. Si $f(h_\alpha, m_\alpha) = f(h'_\alpha, m'_\alpha)$ nous avons une collision explicite pour la fonction de compression f . Sinon, posons β le plus petit entier dans $\{\alpha, \dots, k\}$ tel que $h_{\beta+1} = h'_{\beta+1}$. Cet entier existe puisque $h_{k+1} = \mathcal{H}(m) = \mathcal{H}(m') = h'_{k+1}$. Nous avons alors $h_\beta \neq h'_\beta$ et $h_{\beta+1} = f(h_\beta, m_\beta) = f(h'_\beta, m'_\beta) = h'_{\beta+1}$ et nous obtenons donc une collision explicite pour la fonction de compression f .

La fonction itérée \mathcal{H} obtenue à partir de f et R est donc bien résistante aux collisions dès que f est résistante aux collisions.

3.1. Généralités sur les fonctions de hachage

Une fonction de hachage \mathcal{H} qui produit des empreintes de n bits pour des messages de taille arbitraire est considérée comme « sûre » s'il n'existe pas d'attaque permettant de trouver une (seconde) pré-image en un nombre d'appels à la fonction de hachage inférieur à 2^n , ni de trouver une collision en un nombre d'appels inférieur à $2^{n/2}$. Il existe des algorithmes génériques (*i.e.* ne nécessitant aucune connaissance du fonctionnement de la fonction attaquée) pour résoudre ces problèmes avec ce nombre d'appels à la fonction de hachage (*cf.* (3.1) et (3.2)).

Algorithme 3.1 Recherche générique de pré-image

ENTRÉE: $y \in \{0, 1\}^n$, $m \in \mathbb{N}$ avec $m > n$

SORTIE: $x \in \{0, 1\}^m$ tel que $y = \mathcal{H}(x)$

tant que VRAI faire

$x \xleftarrow{u.a.} \{0, 1\}^m$

si $\mathcal{H}(x) = y$ **alors**

retourner x

fin si

fin tant que

Algorithme 3.2 Recherche générique de collision

ENTRÉE: $m \in \mathbb{N}$ avec $m > n$

SORTIE: $x, x' \in \{0, 1\}^m$ tel que $\mathcal{H}(x) = \mathcal{H}(x')$ et $x \neq x'$

$\Upsilon \leftarrow \emptyset$

tant que VRAI faire

$x_i \xleftarrow{u.a.} \{0, 1\}^m$

$y_i \leftarrow \mathcal{H}(x_i)$

$j \leftarrow \text{RECHERCHE}(y_i, \Upsilon)$

si $j \neq -1$ et $x_i \neq x_j$ **alors**

retourner (x_i, x_j)

fin si

$\Upsilon \leftarrow \Upsilon \cup (x_i, y_i)$

▷ table triée selon la coordonnée y

fin tant que

Un exemple de fonction de hachage qui a longtemps été utilisée est la fonction MD5 (de l'anglais, *Message Digest 5*) inventée en 1991 par R. RIVEST. Elle repose sur la construction de Merkle-Damgaard où le message est divisé en blocs de taille fixe $\ell = 512$ et produit des empreintes de taille $n = 128$ bits. Le vecteur d'initialisation h_0 est la concaténation de 4 mots de 32 bits

(67452301, EFCDAB89, 98BADCFE, 10325476).

La fonction de compression se décomposent en quatre tours (eux-mêmes divisés en 16 opérations basées sur une fonction non-linéaire choisie parmi quatre qui varie selon la ronde, une addition et une rotation vers la gauche). La fonction MD5 est décrite en détail sur l'algorithme (3.3) (où toutes les variables sont de 32 bits sauf les m_i qui sont de 512 bits, où + désigne l'addition modulo 2^{32} , et $\lll i$ indique une rotation sur la gauche de i positions). La valeur de 128 bits rentrée est concaténée avec le mot de poids faible en tête (*i.e. little-endian* ou petit-boutiste).

Algorithme 3.3 Fonction de hachage MD5

ENTRÉE: $m \in \{0, 1\}^*$, $|m| < 2^{64} - 1$

SORTIE: $h \in \{0, 1\}^{128}$, $h = \text{MD5}(m)$

```

 $r[0..15] \leftarrow \{7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22\}$ 
 $r[16..31] \leftarrow \{5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20\}$ 
 $r[32..47] \leftarrow \{4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23\}$ 
 $r[48..63] \leftarrow \{6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21\}$ 
pour  $i$  de 0 à 63 faire
     $k[i] \leftarrow \lfloor (|\sin(i+1)| \cdot 2^{32}) \rfloor$ 
fin pour
 $h^0 \leftarrow 67452301$ ;  $h^1 \leftarrow \text{EFCDAB89}$ ;  $h^2 \leftarrow 98BADCCE$ ;  $h^3 \leftarrow 10325476$ 
 $i = |m| \bmod \ell$ 
 $(m_0, \dots, m_k) \leftarrow \mathcal{R}(m) = m || 10^{\ell-i-65} || \tau_m$  ▷ avec  $|m_i| = 512$ 
pour  $j$  de 1 à  $k$  faire
     $(w_0, \dots, w_{15}) \leftarrow m_k$  ▷ avec  $|w_0| = 32, \dots, |w_{15}| = 32$ 
     $a \leftarrow h^0$ ;  $b \leftarrow h^1$ ;  $c \leftarrow h^2$ ;  $d \leftarrow h^3$ 
    pour  $i$  de 0 à 63 faire
        si  $0 \leq i \leq 15$  alors
             $f \leftarrow (b \wedge c) \vee ((\neg b) \wedge d)$ ;  $g \leftarrow i$ 
        sinon si  $16 \leq i \leq 31$  alors
             $f \leftarrow (d \wedge b) \vee ((\neg d) \wedge c)$ ;  $g \leftarrow (5i + 1) \bmod 16$ 
        sinon si  $32 \leq i \leq 47$  alors
             $f \leftarrow b \oplus c \oplus d$ ;  $g \leftarrow (3i + 5) \bmod 16$ 
        sinon si  $48 \leq i \leq 63$  alors
             $f \leftarrow c \oplus (b \vee (\neg d))$ ;  $g \leftarrow (7i) \bmod 16$ 
        fin si
         $(a, b, c, d) \leftarrow (d, ((a + f + k[i] + w[g]) \lll r[i]) + b, b, c)$ 
    fin pour
     $h^0 \leftarrow h^0 + a$ ;  $h^1 \leftarrow h^1 + b$ ;  $h^2 \leftarrow h^2 + c$ ;  $h^3 \leftarrow h^3 + d$ 
fin pour
retourner  $(h^0 || h^1 || h^2 || h^3)$ 

```

Exercice 3.3 (avec programmation). Collisions sur la fonction MD5 tronquée

En utilisant l'algorithme (3.2), trouver deux chaînes de caractères de la forme

"Collision MD5 ??????????"

(où les caractères ? peuvent être des caractères affichables arbitraires) dont les empreintes MD5 commencent par les mêmes 32 premiers bits (*i.e.* une collision sur la fonction MD5 tronquée à 32 bits).

Solution

Une application immédiate de l'algorithme (3.2) en remplaçant les caractères ? par des chiffres aléatoires entre 0 et 9 révèle rapidement une collision sur les 32 premiers bits des empreintes MD5 des messages :

"Collision MD5 62905482264" et "Collision MD5 59667753395"

qui ont pour empreintes respectives (en hexadécimal)

84972a5f93dce1271ecfb0cddcbfa8ea
84972a5f850a31b1ae828a1c919debbf

dont les quatre premiers octets sont égaux à 84972a5f.

En 2004 et 2005, X. WANG et ses collaborateurs ont publié plusieurs attaques en collision dévastatrices contre plusieurs fonctions de hachage proche de la fonction MD5. Ils ont notamment obtenus des collisions effectives pour les fonctions MD5, MD4 (une fonction de hachage antérieure à MD5 et pour laquelle des failles majeures de conception étaient déjà connues) ou SHA-0. Il s'agit d'une fonction de hachage conçue sur le modèle de MD4 et publiée par le NIST en 1993 et pour laquelle des failles étaient également déjà connues. Parmi les travaux de X. WANG et ses collaborateurs, les auteurs présentent une attaque de complexité théorique 2^{63} pour la recherche de collisions de la fonction SHA-1 (de l'anglais *Secure Hash Algorithm*) qui produit des empreintes de 160 bits. Ces attaques sont trop complexes pour pouvoir être présentées dans ce livre mais elles reposent sur des idées proches de celles que nous verrons dans la deuxième partie de ce chapitre.

Suite à la publication de ces résultats, la majorité des fonctions de hachage normalisées avaient été mises en défaut (seule la famille de fonctions de hachage SHA-2 semble résister aux attaques). En 2007, le NIST a lancé une compétition internationale – sur le modèle de celle ayant conduit à la sélection de l'algorithme de chiffrement AES – pour choisir une fonction de hachage de nouvelle génération appelée SHA-3. Le vainqueur de la compétition a été révélé en 2012 ; il s'agit de la fonction Keccak qui a été conçue par G. BERTONI, J. DAEMEN, M. PEETERS et G. VAN ASSCHE.

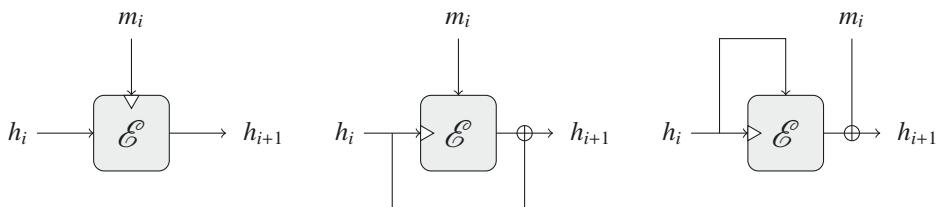
3.2 CHIFFREMENT PAR BLOC ET FONCTION DE COMPRESSION

Les fonctions de compression peuvent être construites en utilisant des techniques *ad hoc* ou à partir de système de chiffrement par bloc. Les fonctions de compression construites à partir d'un système de chiffrement par bloc permettent d'obtenir une fonction de hachage à faible « coût » lorsqu'un système de chiffrement par bloc est déjà implanté dans un système.

Exercice 3.4 Chiffrement par bloc et fonction de compression

Soit $\mathcal{E} : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ un système de chiffrement par blocs qui utilise des clés de n bits pour chiffrer des messages de n bits. Montrer que les trois fonctions de compression f_1 , f_2 et f_3 ne sont pas résistantes à la pré-image.

1. $f_1 : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, $f_1(h, m) = \mathcal{E}_m(h)$
2. $f_2 : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, $f_2(h, m) = \mathcal{E}_h(m) \oplus h$
3. $f_3 : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, $f_3(h, m) = \mathcal{E}_h(h) \oplus m$



Solution

1. Étant donnée une valeur $h^* \in \{0, 1\}^n$, un attaquant peut pour tout message m de son choix calculer $h = \mathcal{E}_m^{-1}(h^*)$ et par définition, $f_1(h, m) = h^*$.
2. Étant donnée une valeur $h^* \in \{0, 1\}^n$, un attaquant peut pour toute valeur h de son choix calculer $m = \mathcal{E}_h^{-1}(h^* \oplus h)$, de sorte que $f_2(h, m) = h^*$.
3. Étant donnée une valeur $h^* \in \{0, 1\}^n$, un attaquant peut pour toute valeur h de son choix calculer $m = \mathcal{E}_h^{-1}(h) \oplus h^*$, de sorte que $f_3(h, m) = h^*$.

Les fonctions f_1 , f_2 et f_3 ne sont pas résistantes à la pré-image et elles ne sont donc pas *a fortiori* résistantes aux collisions.

Une construction de fonction de compression construite à partir d'un système de chiffrement par bloc et résistante aux collisions a été proposée par Matyas, Meyer et Oseas. La fonction de compression $f : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ est définie par $f(h, m) = \mathcal{E}_h(m) \oplus m$ où \mathcal{E} est un système de chiffrement par blocs de n bits (cf. Figure (3.2)). Il est conjecturé qu'il n'existe aucune attaque plus efficace que les attaques génériques sur cette construction ou sur la construction duale connue sous le nom de construction de Davies-Meyer (*i.e.*, une attaque en collision nécessite environ $2^{n/2}$ évaluations de la fonction f et une attaque en (seconde) pré-image nécessite environ 2^n évaluations de la fonction f). Cette conjecture a été démontrée sous l'hypothèse que la primitive de chiffrement par bloc utilisée a un comportement idéal (cf. [9]). En pratique, les systèmes de chiffrement par bloc n'ont pas les mêmes propriétés que les fonctions aléatoires et l'exercice suivant montre que si le chiffrement par bloc utilisé a des propriétés spécifiques alors la fonction de compression peut ne pas être sûre.

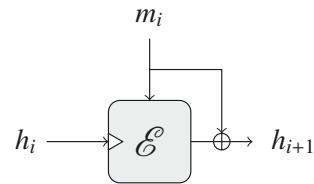


Figure 3.2 - Construction de Matyas-Meyer-Oseas

Exercice 3.5 Sécurité de la construction de Matyas-Meyer-Oseas avec le DES
Montrer que la fonction de compression f n'est pas résistante aux collisions lorsque $\mathcal{E} = \text{DES}$ dans la construction de Matyas-Meyer-Oseas.

Solution

D'après la propriété de complémentation de DES, nous avons pour tout $h \in \{0, 1\}^{56}$ et tout $m \in \{0, 1\}^{64}$

$$f(\bar{h}, \bar{m}) = \mathcal{E}_{\bar{h}}(\bar{m}) \oplus \bar{m} = \overline{\mathcal{E}_h(m)} \oplus \bar{m} = \mathcal{E}_h(m) \oplus m = f(h, m)$$

et f n'est donc pas résistante aux collisions.

En particulier, la preuve de Merkle-Damgård ne s'applique pas et il est possible qu'une fonction de hachage itérée construite à partir de f ne soit pas résistante aux collisions.

Dans l'exercice (3.4), nous avons vu trois exemples de fonction de compression construites à partir d'un système de chiffrement par bloc qui ne sont pas résistantes à la pré-image. Cette construction a été proposée en 1978 par M. O. RABIN. Nous allons montrer dans l'exercice suivant que cette faiblesse dans la fonction de compression f_1 induit une perte de la résistance à la pré-image de la fonction de hachage itérée associée.

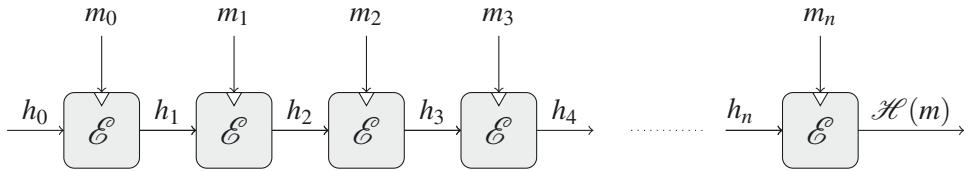


Figure 3.3 – Construction de Rabin.

L'analyse de cette attaque en pré-image reposera sur les bornes de Chernoff rappelées dans le théorème suivant :

Théorème 3.1 Bornes de Chernoff

Soient X_1, \dots, X_n n variables aléatoires indépendantes dans $\{0, 1\}$ avec $\Pr[X_i = 1] = p_i \in [0, 1]$ et $\Pr[X_i = 0] = 1 - p_i$ pour tout $i \in \{1, \dots, n\}$. En notant $X = \sum_{i=1}^n X_i$, nous avons

1. $\mu = E[X] = \sum_{i=1}^n p_i$
2. $\forall \delta \in [0, 1], \Pr[X < (1 - \delta)\mu] \leq \left(\frac{e^{-\delta}}{(1 - \delta)^{1-\delta}}\right)^\mu \leq e^{-\mu\delta^2/2}$
3. $\forall \delta > 0, \Pr[X > (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right)^\mu$
4. $\forall \delta > 2e - 1, \Pr[X > (1 + \delta)\mu] \leq 2^{-\epsilon\mu}$

Exercice 3.6 Attaque en pré-image pour la construction de M. O. RABIN

Considérons une fonction de hachage \mathcal{H} obtenue par la construction de Merkle-Damgård sans ajouter de bloc supplémentaire contenant la longueur du message et où la fonction de compression est $f(h, m) = \mathcal{E}_m(h)$ où \mathcal{E} est un système de chiffrement par bloc.

1. Montrer que si le vecteur d'initialisation h_0 de la construction de Merkle-Damgård peut être choisi librement, alors \mathcal{H} n'est pas résistante à la pré-image. Nous allons montrer que cette construction n'est pas résistante à la pré-image même si h_0 est fixé. Étant donnée une empreinte h^* , considérons la stratégie suivante :
 - (i) L'adversaire tire uniformément aléatoirement des blocs de messages m_i pour $i \in \{1, \dots, T_1\}$ et calcule récursivement $h_i = \mathcal{E}_{m_i}(h_{i-1})$ pour $i \in \{1, \dots, T_1\}$;
 - (ii) L'adversaire choisit un message m'_1 compatible avec la processus de bourrage utilisé et tire uniformément aléatoirement des blocs de messages m'_j pour $i \in \{2, \dots, T_2\}$. Il pose $h'_0 = h^*$ et calcule récursivement $h'_j = \mathcal{E}_{m'_j}^{-1}(h'_{j-1})$ pour $j \in \{1, \dots, T_2\}$.

2. Expliquer comment utiliser cette stratégie pour obtenir une pré-image de h . Analyser l'algorithme obtenu.

Indication. On pourra estimer $\#\{h_i | i \in \{1, \dots, T_1\}\}$ puis évaluer la probabilité que $\{h_i | i \in \{1, \dots, T_1\}\} \cap \{h'_j | j \in \{1, \dots, T_2\}\} \neq \emptyset$.

Solution

- D'après l'exercice (3.4), la fonction de compression f n'est pas résistante à la pré-image et pour toute empreinte h^* , il est facile de construire (h, m) tel que $f(h, m) = h^*$. Si l'on peut choisir $h_0 = h$, le message m est une pré-image de l'empreinte h^* .
- La stratégie proposée suggère une attaque de type « rencontre par le milieu » comme pour le double chiffrement (*cf.* Exercice (2.11)). En effet s'il existe deux indices $i \in \{1, \dots, T_1\}$ et $j \in \{1, \dots, T_2\}$ tels que $h_i = h'_j$, alors nous avons

$$h_i = f(h_{i-1}, m_i) = f(f(h_{i-2}, m_{i-1}), m_i) = \dots = f(f(\dots f(h_0, m_1), \dots, m_{i-1}), m_i)$$

et

$$\begin{aligned} h^* &= f(h'_1, m'_1) \\ &\vdots \\ &= f(f(\dots f(f(h'_j, m'_j), m'_{j-1}), \dots), m'_j), m'_1 \\ &= f(f(\dots f(f(h_i, m'_j), m'_{j-1}), \dots), m'_2), m'_1 \\ &= f(f(\dots f(f(f(\dots f(h_0, m_1), \dots, m_{i-1}), m_i), m'_j), m'_{j-1}), \dots), m'_2), m'_1 \end{aligned}$$

En particulier, nous obtenons que l'empreinte du message

$$(m_1, m_2, \dots, m_{i-1}, m_i, m'_j, m'_{j-1}, \dots, m'_2, m'_1)$$

est égal à h^* . Il reste à déterminer les paramètres T_1 et T_2 pour obtenir une bonne probabilité que

$$\{h_i | i \in \{1, \dots, T_1\}\} \cap \{h'_j | j \in \{1, \dots, T_2\}\} \neq \emptyset.$$

Notons $N = \#\{h_i | i \in \{1, \dots, T_1\}\}$. En supposant que la fonction de compression se comporte comme une fonction aléatoire, la probabilité qu'une valeur fixée de $\{0, 1\}^n$ ne soit jamais obtenue comme h_i pour $i \in \{1, \dots, T_1\}$ est égale à $(1 - 2^{-n})^{T_1}$ et l'entier N est donc égal en moyenne à

$$\mu = 2^n \cdot \left(1 - \left(1 - \frac{1}{2^n}\right)^{T_1}\right).$$

Intuitivement, si T_1 n'est pas trop grand, la probabilité d'obtenir deux valeurs h_i égales est petite et μ doit donc être proche de T_1 . En appliquant l'inégalité de Taylor-Lagrange à l'ordre 2 sur l'intervalle $[0, 2^{-n}]$, nous obtenons effectivement

$$|\mu - T_1| \leq \frac{T_1(T_1 - 1)}{2} 2^{-n}$$

et $\mu \geq T_1/2$ si $T_1 \leq 2^n$. En appliquant la borne de Chernoff, la probabilité que N soit inférieur à $\mu/2$ est majorée par $e^{-\mu/8}$.

En supposant toujours que la fonction de compression se comporte comme une fonction aléatoire, la probabilité qu'un élément h'_j pour $j \in \{1, \dots, T_2\}$ soit égal à une valeur h_i pour $i \in \{1, \dots, T_1\}$ est égal à $N/2^n$. La probabilité qu'il existe au moins une valeur h'_j pour $j \in \{1, \dots, T_2\}$ égale à une valeur h_i pour $i \in \{1, \dots, T_1\}$ est donc égale à $1 - (1 - N/2^n)^{T_2} \geq 1 - \exp(-NT_2/2^n)$.

Le nombre d'évaluations de la fonction de compression est égal à $T_1 + T_2$ et nous obtenons une probabilité de succès supérieure à $1/2$ dès que nous avons $NT_2 > 2^n \ln(2)$. En équilibrant le coût des deux étapes de la stratégie avec $T_1 = T_2 = 2^{n/2+2}$, nous obtenons $N \geq \mu/2 \geq T_1/4 = 2^{n/2}$ avec une probabilité supérieure à $1 - e^{-T_1/16}$ et, dans ce cas, nous avons $N \cdot T_2 \geq 2^{n+2} > 2^n \ln 2$.

Si $T_1 = T_2 = 2^{n/2+2}$, l'attaque réussit donc avec une probabilité supérieure à $(1 - e^{-T_1/16})/2 \approx 1/2$ et fournit une pré-image de h^* en $T_1 + T_2 = 2^{n/2+3}$ évaluations de la fonction de compression (à comparer aux 2^n nécessaires pour une fonction de hachage ayant une sécurité optimale).

3.3 ATTAQUES GÉNÉRIQUES SUR LES FONCTIONS DE HACHAGE ITÉRÉES

Nous présentons dans cette section, trois attaques tirant partie de la construction de Merkle-Damgård.

Une t -multi-collision pour une fonction de hachage \mathcal{H} est un ensemble de messages $\{m_1, \dots, m_t\}$ deux à deux distincts tels que $\mathcal{H}(m_1) = \dots = \mathcal{H}(m_t)$. Une généralisation du paradoxe des anniversaires montre que le coût de la recherche d'une t -multi-collision pour une fonction de hachage \mathcal{H} aléatoire qui produit des empreintes de n bits est de l'ordre de $2^{(t-1)n/t}$ évaluations de la fonction de hachage. L'exercice suivant montre que pour une fonction de hachage itérée suivant la méthode de Merkle-Damgård, le nombre de requêtes nécessaires est beaucoup moins grand.

Exercice 3.7 Multi-collisions pour les fonctions de hachage itérées

Nous considérons une fonction de hachage $\mathcal{H} : \{0, 1\}^r \rightarrow \{0, 1\}^n$ construite à partir d'une fonction de compression $f : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ par la méthode de Merkle-Damgård (avec $\ell > 2n$).

Soit $\mathcal{H}_c : \{0, 1\}^{c \cdot \ell} \rightarrow \{0, 1\}^n$ une fonction construite à partir de f par la méthode de Merkle-Damgård mais sans ajouter de bourrage et utilisée uniquement pour les messages de longueur fixe égale à un multiple de ℓ .

3.3. Attaques génériques sur les fonctions de hachage itérées

1. En cherchant deux collisions bien choisis pour la fonction de compression, montrer comment obtenir une 4-multi-collision pour \mathcal{H}_2 .
2. Expliquer comment transformer cette 4-multi-collision pour \mathcal{H}_2 en une 4-multi-collision pour \mathcal{H} .
3. Généraliser en montrant qu'on peut obtenir une 2^t -multi-collision pour \mathcal{H} pour le coût de t collisions sur f .

Solution

1. Dans un premier temps, il suffit de rechercher une collision de la forme

$$f(h_0, m_1^0) = f(h_0, m_1^1) = h_1$$

avec $m_1^0 \neq m_1^1$. Si la fonction de compression f se comporte comme une fonction aléatoire, le nombre d'évaluations de f nécessaires pour obtenir une telle collision est de l'ordre de $2^{n/2}$ (par le paradoxe des anniversaires).

Nous cherchons ensuite une collision de la forme

$$f(h_1, m_2^0) = f(h_1, m_2^1) = h_2$$

avec $m_2^0 \neq m_2^1$. Le coût pour obtenir une telle collision est encore de l'ordre de $2^{n/2}$ si f se comporte comme une fonction aléatoire.

Les quatre couples (m_i^i, m_j^j) avec $(i, j) \in \{0, 1\}^2$ donnent immédiatement une 4-multi-collision pour \mathcal{H}_2 puisque

$$\mathcal{H}_2(m_1^0, m_2^0) = \mathcal{H}_2(m_1^0, m_2^1) = \mathcal{H}_2(m_1^1, m_2^0) = \mathcal{H}_2(m_1^1, m_2^1) = h_2.$$

Le coût total pour obtenir cette 4-multi-collision est donc de l'ordre de $2^{n/2+1}$ si f se comporte comme une fonction aléatoire (au lieu de $2^{2n/3}$).

2. Puisque les messages formant la 4-multi-collision précédente sont de même longueur, en ajoutant un bloc complet contenant le bourrage (par exemple le bloc $1||0^{n-1}$) puis la longueur du message, nous obtenons une 4-multi-collision pour \mathcal{H} .
3. La méthode est une simple généralisation de la première question (cf. Figure 3.4). Il suffit de rechercher une collision de la forme $f(h_0, m_1^0) = f(h_0, m_1^1) = h_1$ avec $m_1^0 \neq m_1^1$, puis des collisions de la forme $f(h_{i-1}, m_i^0) = f(h_{i-1}, m_i^1) = h_{i+1}$ avec $m_i^0 \neq m_i^1$ pour $i \in \{2, \dots, t\}$. Les 2^t couples $(m_1^{i_1}, m_2^{i_2}, \dots, m_t^{i_t})$ avec $(i_1, \dots, i_t) \in \{0, 1\}^t$ donnent une 2^t -multi-collision pour \mathcal{H}_t (et donc pour \mathcal{H} comme dans la question 2).

Le coût total pour obtenir cette 2^t -multi-collision est donc de l'ordre de $t \cdot 2^{n/2}$ si f se comporte comme une fonction aléatoire (au lieu de $2^{(2^t-1)n/2^t}$).

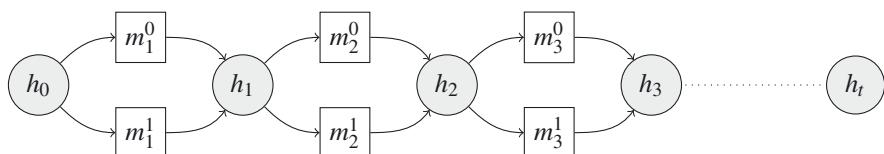


Figure 3.4 - Multi-collisions sur une fonction de hachage itérée.

La concaténation de fonctions de hachage indépendantes a été proposée pour renforcer la sécurité face aux attaques en pré-image, en seconde pré-image et en collisions. Des versions (maintenant obsolètes) du protocole TLS/SSL de sécurisation des échanges sur Internet utilisaient par exemple la concaténation des fonctions MD5 et SHA-1. L'exercice suivant montre que la concaténation d'une fonction de hachage quelconque et d'une fonction de hachage itérée suivant la méthode de Merkle-Damgård est aussi résistante aux collisions que la fonction la plus sûre (mais pas plus).

Exercice 3.8 Attaque en collision contre fonctions de hachage concaténées

Soient \mathcal{H}_1 et \mathcal{H}_2 deux fonctions de hachage de même domaine qui produisent des empreintes de n bits et considérons la fonction de hachage \mathcal{H} définie pour tout message m par $\mathcal{H}(m) = \mathcal{H}_1(m)\|\mathcal{H}_2(m)$ (en particulier \mathcal{H} produit des empreintes de $2n$ bits).

1. Montrer que \mathcal{H} est résistante aux collisions dès que l'une des fonctions \mathcal{H}_1 ou \mathcal{H}_2 est résistante aux collisions.
2. En supposant que \mathcal{H}_1 est une fonction de hachage itérée vulnérable à l'attaque des multi-collisions de l'exercice précédent, proposer un algorithme pour construire une collision pour la fonction \mathcal{H} en environ $2^{n/2}(n/2)$ évaluations de la fonction de hachage \mathcal{H}_1 et $2^{n/2}$ évaluations de la \mathcal{H}_2 .

Solution

1. Supposons que nous disposons de deux messages m_1 et m_2 avec $m_1 \neq m_2$ tels que $\mathcal{H}(m_1) = \mathcal{H}(m_2)$. Nous avons

$$\mathcal{H}(m_1) = \mathcal{H}_1(m_1)\|\mathcal{H}_2(m_1) = \mathcal{H}_1(m_2)\|\mathcal{H}_2(m_2) = \mathcal{H}(m_2)$$

Une collision pour \mathcal{H} fournit donc une collision pour \mathcal{H}_1 et une collision pour \mathcal{H}_2 . Si l'une de ces deux fonctions est résistante aux collisions, la fonction concaténée \mathcal{H} est nécessairement résistante aux collisions.

2. Si \mathcal{H}_1 est une fonction de hachage itérée vulnérable à l'attaque des multi-collisions, il est possible de construire une $2^{n/2}$ -multi-collision en $2^{n/2}(n/2)$ évaluations de la fonction de compression associée. Il suffit ensuite d'évaluer la fonction \mathcal{H}_2 en les $2^{n/2}$ messages obtenus. Par le paradoxe des anniversaires, la probabilité d'obtenir une collision est constante. En répétant cette attaque un nombre constant de fois, nous obtenons une collision pour \mathcal{H}_2 qui est aussi une collision pour \mathcal{H}_1 et donc une collision pour \mathcal{H} .

Note

Les deux faiblesses des fonctions de hachage itérées des exercices précédents ont été présentées par A. Joux lors de la conférence *Crypto 2004* [32].

Dans l'exercice (3.6), nous avons vu que si la fonction de compression d'une fonction de hachage itérée suivant la méthode de Merkle-Damgård sans ajouter de bloc supplémentaire contenant la longueur du message n'est pas résistante à la pré-image alors la fonction de hachage n'est pas résistante à la pré-image. Dans l'exercice suivant, nous allons montrer qu'une fonction de hachage itérée vulnérable à l'attaque des multi-collisions n'est pas résistante à la seconde pré-image pour des longs messages.

Problème 3.9 Attaque de Kelsey-Schneier

Soit \mathcal{H} une fonction de hachage itérée suivant la méthode de Merkle-Damgård à partir d'une fonction de compression f et qui produit des empreintes de n bits .

1. Supposons que le processus de bourrage utilisé est indépendant du message (*i.e.*, en particulier, qu'il n'ajoute pas de bloc supplémentaire contenant la longueur du message). Montrer qu'une seconde pré-image d'un message formé de T blocs peut être obtenu en environ $2^n/T$ évaluations de la fonction f .
2. Montrer comment adapter l'attaque en multi-collisions de Joux pour obtenir une 4-multi-collision où les quatre messages sont formés de 2, 3, 4 et 5 blocs.

Indication. On pourra commencer par construire une collision entre un message formé d'un bloc et un message formé de deux blocs.

3. Généraliser en montrant comment construire pour tout entier t une 2^t -multi-collision où les 2^t messages de même empreinte sont formés d'un nombre de blocs distincts dans l'intervalle $\{t, t + 1, \dots, t + 2^t - 1\}$.
4. Utiliser cette 2^t -multi-collision pour adapter l'attaque de la première question lorsque le processus de bourrage utilisé ajoute un bloc final contenant la longueur du message.

Solution

1. Considérons un message formé de T blocs $m = m_1\|m_2\|\dots\|m_T$ où le dernier bloc contient le bourrage. Le calcul de $\mathcal{H}(m)$ définit T valeurs intermédiaires $h_i = f(h_{i-1}, m_i)$ pour $i \in \{1, \dots, T\}$ (où h_0 est la valeur initiale de la construction de la fonction de hachage itérée).

Un adversaire qui tente de construire une seconde pré-image de $\mathcal{H}(m) = h_T$ peut tirer uniformément aléatoirement des blocs de messages m'_i pour $i \in \{1, \dots, S\}$ et

calcule récursivement $h'_j = f(h'_{j-1}, m'_j)$ pour $j \in \{1, \dots, S\}$ avec $h'_0 = h_0$. S'il existe deux indices $i \in \{1, \dots, T\}$ et $j \in \{1, \dots, S\}$, tels que $h_i = h'_j$ alors le message

$$m'_1 \| m'_2 \| \dots \| m'_j \| m'_{j+1} \| \dots \| m_T$$

a la même empreinte que le message m par la fonction \mathcal{H} (où m_T contient les informations du processus de bourrage qui est indépendant du message). En notant $N = \#\{h_i | i \in \{1, \dots, T\}\}$ et en supposant que la fonction de compression f se comporte comme une fonction aléatoire, la probabilité pour chaque $j \in \{1, \dots, S\}$ que $h'_j \in \{h_i | i \in \{1, \dots, T\}\}$ est égale à $N/2^n$. La probabilité que

$$\{h_i | i \in \{1, \dots, T\}\} \cap \{h'_j | j \in \{1, \dots, S\}\} \neq \emptyset.$$

est égale à $1 - (1 - N/2^n)^S \geq 1 - \exp(-NS/2^n)$. En reprenant l'analyse de l'exercice (3.6), avec $S = 2^{n+4}/T$, une seconde pré-image de $\mathcal{H}(m)$ est obtenue avec une probabilité supérieure à 1/2.

2. Dans un premier temps, suivant l'indication, nous recherchons une collision de la forme

$$f(h_0, m_1) = f(f(h_0, m_1^a), m_1^b) = h_1$$

Si la fonction de compression f se comporte comme une fonction aléatoire, le nombre d'évaluations de f nécessaires pour obtenir une telle collision est de l'ordre de $2^{n/2}$ (par le paradoxe des anniversaires).

Nous cherchons ensuite une collision entre un message formé d'un bloc et un message formé de trois blocs de la forme

$$f(h_1, m_2) = f(f(f(h_1, m_2^a), m_2^b), m_2^c) = h_2$$

avec $m_2^0 \neq m_2^1$. Le coût pour obtenir une telle collision est encore de l'ordre de $2^{n/2}$ si f se comporte comme une fonction aléatoire.

Nous obtenons quatre couples de message (m_1, m_2) , (m_1^a, m_1^b, m_2) , $(m_1, m_2^a, m_2^b, m_2^c)$ et $(m_1^a, m_1^b, m_2^a, m_2^b, m_2^c)$ qui ont la même empreinte par \mathcal{H} et qui sont bien formés de 2, 3, 4 et 5 blocs respectivement (*cf.* Figure (3.5)).

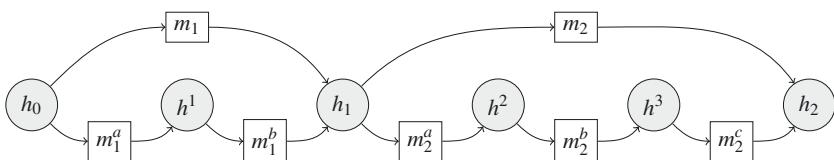


Figure 3.5 - 4-multi-collisions de tailles différentes

3. Il suffit de généraliser la méthode précédente en calculant
 - deux messages m_1^0 et m_1^1 (où m_1^0 est formé d'un bloc et m_1^1 de deux blocs) tels que $f(h_0, m_1^0) = f(h_0, m_1^1) = h_1$;
 - deux messages m_2^0 et m_2^1 (où m_2^0 est formé d'un bloc et m_2^1 de trois blocs) tels que $f(h_1, m_2^0) = f(h_1, m_2^1) = h_2$;

3.3. Attaques génériques sur les fonctions de hachage itérées

- deux messages m_3^0 et m_3^1 (où m_3^0 est formé d'un bloc et m_3^1 de cinq blocs) tels que $f(h_2, m_3^0) = f(h_2, m_3^1) = h_3$;
- et par récurrence, deux messages m_k^0 et m_k^1 (où m_k^0 est formé d'un bloc et m_k^1 de $2^{k-1} + 1$ blocs) tels que $f(h_k, m_{k+1}^0) = f(h_k, m_{k+1}^1) = h_{k+1}$ pour $k \in \{3, \dots, t\}$.

Comme dans l'attaque de Joux, nous obtenons bien une 2^t collision et tous les messages sont formés d'un nombre de blocs distincts dans l'intervalle d'entiers $\{t, t+1, \dots, t+2^t-1\}$ (et tous ces nombres de bloc sont effectivement obtenus). Pour chaque $k \in \{1, \dots, t\}$, l'attaque demande environ $t + 2^{n/2}$ évaluations de la fonction de compression et le coût de l'attaque est essentiellement de l'ordre de $t2^{n/2}$.

4. Considérons comme dans la première question, un message formé de T blocs $m = m_1 \| m_2 \| \dots \| m_T$ où le dernier bloc encode la longueur du message m . Le calcul de $\mathcal{H}(m)$ définit T valeurs intermédiaires $h_i = f(h_{i-1}, m_i)$ pour $i \in \{1, \dots, T\}$.

Supposons construite une 2^t -multi-collision comme dans la question précédente avec 2^t messages formés d'un nombre de blocs distincts dans l'intervalle d'entiers $\{t, t+1, \dots, t+2^t-1\}$ qui ont la même empreinte h^* . Un adversaire qui tente de construire une seconde pré-image de $\mathcal{H}(m) = h_T$ peut tirer uniformément aléatoirement des blocs de messages m'_i pour $i \in \{1, \dots, S\}$ et calcule récursivement $h'_j = f(h'_{j-1}, m'_j)$ pour $j \in \{1, \dots, S\}$ avec $h'_0 = h^*$ (cf. Figure (3.6)).

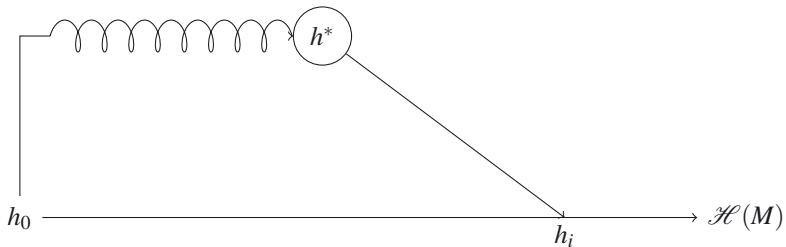


Figure 3.6 - Attaque de Kelsey et Schneier

S'il existe deux indices $i \in \{1, \dots, T\}$ et $j \in \{1, \dots, S\}$, tels que $h_i = h'_j$ pour tout message m^* de la multi-collision, le message

$$m \| m'_1 \| m'_2 \| \dots \| m'_j \| m_{i+1} \| \dots \| m_T$$

a la même empreinte que le message m par la fonction \mathcal{H} s'il a la même longueur que le message m . Si l'entier $i-j$ est dans l'intervalle d'entiers $\{t, t+1, \dots, t+2^t-1\}$, alors en choisissant le message m^* de la multi-collision de longueur $i-j$ nous obtenons une seconde pré-image de $\mathcal{H}(m)$.

Le choix $t = \log(T)$ fournit une probabilité de succès constante avec un algorithme qui évalue la fonction de compression environ $2^{n/2} \log T + 2^n/T$ fois.

Note

Cette attaque a été présentée par J. KELSEY et B. SCHNEIER lors de la conférence *Eurocrypt 2005* [34].

3.4 CRYPTANALYSE DIFFÉRENTIELLE

La cryptanalyse différentielle est une méthode de cryptanalyse qui s'applique aux systèmes de chiffrement par bloc (mais également aux algorithmes de chiffrement par flot et aux fonctions de hachage). Elle s'effectue généralement dans un contexte d'attaque à textes clairs choisis en se basant sur des couples de textes clairs qui ont une différence constante (généralement pour la fonction « ou exclusif » et parfois pour l'addition modulaire). L'attaquant calcule ensuite les différences entre les textes chiffrés et recherche des motifs spécifiques pouvant indiquer un biais. Le choix des différences appliquées en entrée est crucial pour la réussite de l'attaque. Une analyse précise de l'algorithme de chiffrement est donc nécessaire pour déterminer les différences qui ont la plus grande chance d'apparaître et obtenir une propriété statistiquement observable permettant de tester des clés.

La découverte de la cryptanalyse différentielle est attribuée à E. BIHAM et A. SHAMIR à la fin des années 1980. Les concepteurs du système de chiffrement DES avaient cependant découvert le principe de la cryptanalyse différentielle dès les années 1970 et le DES a été conçu pour y résister. Nous allons étudier les propriétés différentielles de la première S-boîte du système de chiffrement DES. Cette boîte, notée S_1 , est décrite dans la table (3.1). Elle prend en entrée une valeur de 6 bits et retourne en sortie une valeur de 4 bits.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
1	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8
2	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
3	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

Tableau 3.1 – Boîte S_1 du chiffrement DES

La valeur de S_1 sur une entrée de 6 bits $\alpha = \alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5 \in \{0, 1\}^6$ est obtenue (en hexadécimal) à l'intersection de la ligne indexée par le bloc formé des deux bits extrêmaux $\alpha_\ell = \alpha_0\alpha_5$ et de la ligne indexée par le bloc formé des quatre bits internes $\alpha_c = \alpha_1\alpha_2\alpha_3\alpha_4$.

Exemple : la valeur de $S_1(011011)$ est 0101 puisqu'on trouve la valeur 5 = 0101 à l'intersection de la ligne 1 = 01 et de la colonne D = 1101.

Exercice 3.10 (avec programmation). Table des différences du DES

Construire la table des différences de la boîte S_1 du chiffrement DES, c'est-à-dire la table des cardinaux des ensembles

$$\Delta_{\alpha,\beta} = \{i \in \{0, 1\}^6 \mid S_1(i \oplus \alpha) \oplus S_1(i) = \beta\}$$

pour $\alpha \in \{0, 1\}^6$ et $\beta \in \{0, 1\}^4$ (avec $\alpha \neq 0^6$). Donner le cardinal maximal obtenu.

Solution

Une recherche informatique montre par exemple que, pour $\alpha = 001010$ et $\beta = 0101$, nous avons $S_1(i \oplus \alpha) \oplus S_1(i) = \beta$ pour les valeurs

$$i \in \Delta_{\alpha,\beta} = \{000100, 001110, 100100, 101110, 110000, 110001, 111010, 111011\}$$

et $\#\Delta_{\alpha,\beta} = 8$. Par une recherche exhaustive, nous obtenons la table des différences (3.2) où la valeur à l'intersection de la ligne indexée par $\alpha \in \{0, 1\}^6$ et la colonne indexée par $\beta \in \{0, 1\}^4$ est le cardinal de $\Delta_{\alpha,\beta}$ (*i.e.* le nombre de différences obtenues $i \in \{0, 1\}^6$ telles que $S_1(i \oplus \alpha) \oplus S_1(i) = \beta$). La valeur maximale de $\#\Delta_{\alpha,\beta}$ est 14 obtenue par exemple pour $\alpha = 000011$ et $\beta = 0000$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000001	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
000010	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
000011	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
000100	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
000101	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
000110	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
000111	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
001000	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
001001	10	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
001010	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
001011	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
001100	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
001101	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
001110	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
001111	2	0	2	4	4	6	4	2	4	8	2	2	2	6	8	8

Tableau 3.2 - Table des différences pour la boîte S_1 du DES (extrait)

Le chiffrement DES a été conçu pour résister à la cryptanalyse différentielle, mais d'autres algorithmes créés à la même époque se sont révélés particulièrement vulnérables à ce type d'attaque. Le système FEAL (pour *Fast Data Encipherment Algorithm*) est un algorithme de chiffrement par bloc qui utilise des clés de 64 bits pour chiffrer des blocs de 64 bits. Il a été proposé par A. SHIMIZU et S. MIYAGUCHI en 1987 comme une alternative au système DES.

Nous allons étudier une attaque par cryptanalyse différentielle contre sa version originale (appelée FEAL-4) qui est un schéma de Feistel à 4 tours avec pré-

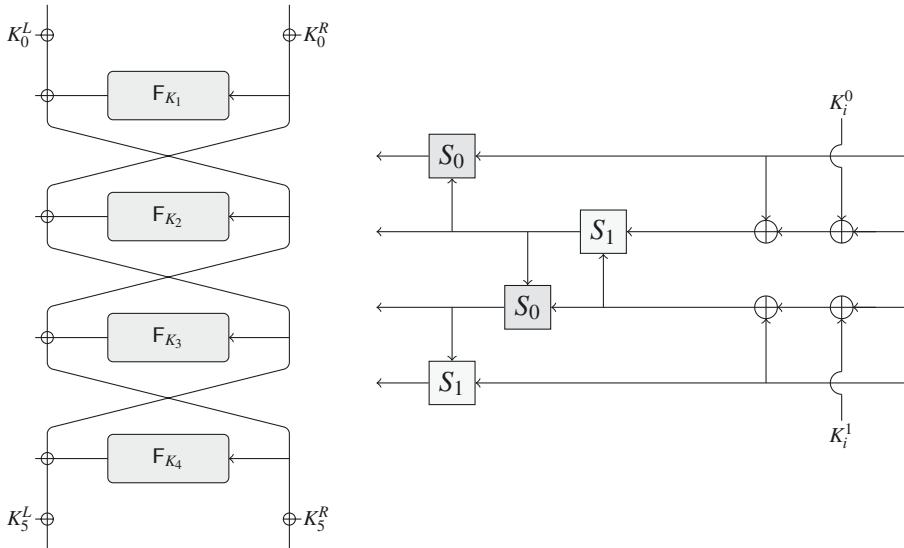


Figure 3.7 – Description du système de chiffrement par bloc FEAL-4

blanchiment et post-blanchiment (*cf.* Exercice (2.10)). Le système FEAL-4 est décrit dans la figure (3.7).

Après une procédure de diversification de clé, la clé K produit deux sous-clés de 64 bits K_0 et K_5 et quatre sous-clés de 16 bits K_1, K_2, K_3 et K_4 . La clé K_0 est utilisée pour le pré-blanchiment (*i.e.* elle est ajoutée au texte clair¹ avant d’appliquer le schéma de Feistel) et la clé K_5 est utilisée pour le post-blanchiment (*i.e.* elle est ajoutée à la sortie du schéma de Feistel pour produire le chiffré).

La fonction de tour F utilise deux S-boîtes S_0 et S_1 définies par

$$S_i(x, y) = (x + y + i \bmod 256) \lll 2 \text{ pour } i \in \{0, 1\}$$

En utilisant une clé de tour de 16 bits K_i décomposée en deux octets $K_i = (K_i^{(0)}, K_i^{(1)})$ pour $i \in \{1, 2, 3, 4\}$, la fonction F prenant en entrée 32 bits décomposés en quatre octets $X^i = (x_0^i, x_1^i, x_2^i, x_3^i)$ calcule

$$u_i = S_1(x_0^i \oplus x_1^i \oplus K_i^{(0)}, x_2^i \oplus x_3^i \oplus K_i^{(1)}) \text{ et } v_i = S_0(x_2^i \oplus x_3^i \oplus K_i^{(1)}, u_i)$$

puis retourne

$$F_{K_i}(X) = (S_0(x_0^i, u_i), u_i, v_i, S_1(x_3^i, v_i))$$

1. Comme pour le chiffrement DES, les textes clair et chiffrés subissent des permutations publiques supplémentaires au début et à la fin du chiffrement. Cependant, elles n’ont pas d’impact sur la sécurité du schéma et nous les ignorons dans la suite.

Problème 3.11 Cryptanalyse différentielle de FEAL-4

- Montrer que pour tout couple d'octets $(x, y) \in \{0, \dots, 255\}^2$, nous avons

$$S_0(x \oplus 80, y) = S_0(x, y) \oplus 20.$$

- Soient $X_0 \in \{0, 1\}^{64}$ et $Y_0 = X_0 \oplus (80\ 80\ 00\ 00, 00\ 00\ 00\ 00)$. En notant X_2 et Y_2 leurs images après application de deux tours du chiffrement FEAL avec une clé arbitraire, montrer que

$$X_2 \oplus Y_2 = (02\ 00\ 00\ 00, 80\ 80\ 00\ 00).$$

- En déduire une relation liant les chiffrés de deux messages dont la différence est égale à $(80\ 80\ 00\ 00, 00\ 00\ 00\ 00)$ qui ne dépend que de la valeur $K_5^R \oplus (0, K_4^0, K_4^1, 0)$.
- Proposer une attaque contre le chiffrement FEAL-4 utilisant seulement deux clairs choisis permettant de retrouver la valeur $K_5^R \oplus (0, K_4^0, K_4^1, 0)$ en 2^{33} évaluations de la fonction F .
- En supposant connu $K_5^R \oplus (0, K_4^0, K_4^1, 0)$ et en utilisant une propriété différentielle sur un tour, proposer une attaque à deux clairs choisis permettant de retrouver la valeur de $K_5^L \oplus (0, K_3^0, K_3^1, 0)$ en 2^{33} évaluations de la fonction F .
- Proposer une attaque à huit clairs choisis permettant de trouver une clé équivalente de FEAL-4 en 2^{35} évaluations de la fonction F .

Solution

- Soient $x, y \in \{0, \dots, 255\}$. Notons $x = x_7x_6x_5x_4x_3x_2x_1x_0$, la décomposition binaire de x et y . Nous avons

$$x \oplus 80 = x_7x_6x_5x_4x_3x_2x_1x_0 \oplus 10000000 = \overline{x}_7x_6x_5x_4x_3x_2x_1x_0$$

Les sommes modulo 256 de x et $x \oplus 80$ avec y ne diffèrent donc que pour le bit de poids fort. Indépendamment de la retenue venant du bit précédent, le bit de poids fort de la somme est toujours modifié. La retenue correspondante peut être éventuellement modifiée mais comme elle correspond à l'ajout de l'entier 256, elle n'influence pas le calcul de S_0 .

La différence est donc uniquement sur le bit de poids fort, soit 80 et en faisant la rotation de deux bits vers la gauche, nous obtenons bien une différence égale à 20.

- Considérons deux textes clairs X_0 et Y_0 tels que

$$X_0 \oplus Y_0 = (80\ 80\ 00\ 00, 00\ 00\ 00\ 00)$$

Après l’ajout de la clé de pré-blanchiment, nous avons

$$(X_0 \oplus K_0) \oplus (Y_0 \oplus K_0) = X_0 \oplus Y_0 = (80\ 80\ 00\ 00, 00\ 00\ 00\ 00)$$

Les deux parties droites de $X_0 \oplus K_0$ et de $Y_0 \oplus K_0$ sont donc égales. Indépendamment de la clé du premier tour K_1 , les valeurs après application de la fonction F_{K_1} sont égales et ne modifient pas la différence dans les parties gauches de $X_0 \oplus K_0$ et de $Y_0 \oplus K_0$. Nous avons donc à la sortie du premier tour, après l’échange des parties droite et gauche,

$$X_1 \oplus Y_1 = (00\ 00\ 00\ 00, 80\ 80\ 00\ 00)$$

Nous devons maintenant calculer $F_{K_2}(X_1^R) \oplus F_{K_2}(Y_1^R)$ sachant que $X_1^R \oplus Y_1^R = 80\ 80\ 00\ 00$. Posons $X_1^R = (x_0, x_1, x_2, x_3)$ et $Y_1^R = (y_0, y_1, y_2, y_3)$. En regardant le résultat octet par octet, posons

$$u = S_1(x_0 \oplus x_1 \oplus K_i^{(0)}, x_2 \oplus x_3 \oplus K_i^{(1)}) \text{ et } v = S_0(x_2 \oplus x_3 \oplus K_i^{(1)}, u)$$

et

$$r = S_1(y_0 \oplus y_1 \oplus K_i^{(0)}, y_2 \oplus y_3 \oplus K_i^{(1)}) \text{ et } s = S_0(y_2 \oplus y_3 \oplus K_i^{(1)}, r)$$

Nous avons $x_0 \oplus y_0 = 80$ et $x_1 \oplus y_1 = 80$, donc

$$x_0 \oplus y_0 \oplus x_1 \oplus y_1 = 80 \oplus 80 = 00 \text{ et donc } x_0 \oplus x_1 = y_0 \oplus y_1.$$

De même, puisque $x_2 \oplus y_2 = x_3 \oplus y_3 = 00$, nous avons $x_2 \oplus x_3 = y_2 \oplus y_3$. Nous en déduisons que $u = r$ et $v = s$. Par ailleurs, puisque $x_3 = y_3$, nous avons $S_1(x_3, v) = S_1(y_3, s)$. Enfin, nous avons

$$S_0(y_0, r) = S_0(y_0, u) = S_0(x_0 \oplus 80, u) = S(x_0, u) \oplus 20$$

d’après la question précédente. Finalement, nous obtenons

$$\begin{aligned} F_{K_2}(X_1^R) \oplus F_{K_2}(Y_1^R) &= (S_0(x_0, u), u, v, S_1(x_3, v)) \oplus (S_0(y_0, r), r, s, S_1(y_3, s)) \\ &= 02\ 00\ 00\ 00 \end{aligned}$$

3. D’après la question précédente, nous avons à la sortie du deuxième tour, après l’échange des parties droite et gauche,

$$X_2 \oplus Y_2 = (80\ 80\ 00\ 00, 02\ 00\ 00\ 00)$$

et donc à la sortie du troisième tour, après l’échange des parties droite et gauche,

$$X_3^L \oplus Y_3^L = 02\ 00\ 00\ 00$$

Notons $X_5 = X_4 \oplus K_5$ et $Y_5 = Y_4 \oplus K_5$ les chiffrés correspondants aux clairs X_0 et Y_0 (avec X_4 et Y_4 les valeurs à la sortie du quatrième tour du schéma de Feistel). Nous avons

$$X_4^R \oplus K_5^R = X_5^R \text{ et } Y_4^R \oplus K_5^R = Y_5^R$$

et

$$\begin{aligned}
 F_{K_4}(X_4^R) \oplus F_{K_4}(Y_4^R) &= (X_4^L \oplus X_3^L) \oplus (Y_4^L \oplus Y_3^L) \\
 &= (X_3^L \oplus Y_3^L) \oplus (X_5^L \oplus K_5^L) \oplus (Y_5^L \oplus K_5^L) \\
 &= (X_3^L \oplus Y_3^L) \oplus (X_5^L \oplus Y_5^L) \\
 &= (\text{02 00 00 00}) \oplus (X_5^L \oplus Y_5^L)
 \end{aligned}$$

En notant G la fonction F associée à la clé nulle, nous avons

$$F_{K_4}(X_4^R) = G(X_4^R \oplus (0, K_4^0, K_4^1, 0)) = G(X_5^R \oplus K_5^R \oplus (0, K_4^0, K_4^1, 0))$$

Enfin, nous obtenons la relation

$$\begin{aligned}
 &G(X_5^R \oplus K_5^R \oplus (0, K_4^0, K_4^1, 0)) \oplus G(Y_5^R \oplus K_5^R \oplus (0, K_4^0, K_4^1, 0)) \\
 &= (\text{02 00 00 00}) \oplus (X_5^L \oplus Y_5^L)
 \end{aligned}$$

qui ne dépend que des chiffrés et de la valeur $K_5^R \oplus (0, K_4^0, K_4^1, 0)$

4. Un attaquant peut demander le chiffrement de deux messages X_0 et Y_0 dont la différence est égale à $(\text{80 80 00 00}, \text{00 00 00 00})$. Il teste ensuite les 2^{32} valeurs possibles pour $K_5^R \oplus (0, K_4^0, K_4^1, 0)$ et vérifie si la relation de la question précédente se produit ou non (cette étape réclame l'évaluation de la fonction F , 2^{33} fois). Le nombre de clés obtenues par cette attaque sera petit (en moyenne égal à 1).
5. En supposant connue la valeur $K_5^R \oplus (0, K_4^0, K_4^1, 0)$ et un chiffré X_5 , nous pouvons calculer la valeur de sortie de la fonction F au quatrième tour. En calculant le « ou exclusif » de cette valeur, avec X_5^L et la clé $K_5^L \oplus (0, K_3^0, K_3^1, 0)$, nous obtenons la valeur en entrée de la fonction G au troisième tour.

L'approche est donc la même que dans la question précédente mais il faut utiliser une autre propriété différentielle pour obtenir la clé utilisée dans le troisième tour (car la propriété précédente est vérifiée indépendamment de cette clé). Nous pouvons utiliser une variante de la différentielle utilisée dans les premières questions. Considérons par exemple deux textes clairs U_0 et V_0 tels que

$$U_0 \oplus V_0 = (\text{00 00 00 00}, \text{80 80 00 00})$$

D'après la question 1, nous avons (avec les mêmes notations)

$$U_1 \oplus V_1 = (\text{80 80 00 00}, \text{20 00 00 00}) \text{ et } U_2^L \oplus V_2^L = \text{20 00 00 00}$$

Notons $U_5 = U_4 \oplus K_5$ et $V_5 = V_4 \oplus K_5$ les chiffrés correspondants aux clairs U_0 et V_0 (avec U_4 et V_4 les valeurs de sortie du quatrième tour du schéma de Feistel). Nous avons

$$U_4^R \oplus K_5^R = U_5^R \text{ et } V_4^R \oplus K_5^R = V_5^R$$

et

$$\begin{aligned}
 F_{K_3}(U_3^L) \oplus F_{K_3}(V_3^L) &= (U_3^R \oplus U_2^L) \oplus (V_3^R \oplus V_2^L) \\
 &= (U_2^L \oplus V_2^L) \oplus (U_3^R \oplus V_3^R) \\
 &= (\texttt{02 00 00 00}) \oplus (U_3^R \oplus V_3^R) \\
 &= (\texttt{02 00 00 00}) \oplus (U_4^R \oplus V_4^R) \\
 &= (\texttt{02 00 00 00}) \oplus (U_5^R \oplus K_5^R) \oplus (V_5^R \oplus K_5^R) \\
 &= (\texttt{02 00 00 00}) \oplus (U_5^R \oplus V_5^R)
 \end{aligned}$$

En notant $\widetilde{U}_4 = F_{K_4}(U_4^R) = G(U_5^R \oplus K_5^R \oplus (0, K_4^0, K_4^1, 0))$, nous avons

$$\begin{aligned}
 F_{K_3}(U_3^L) &= F_{K_3}(U_4^L \oplus \widetilde{U}_4) = F_{K_3}(U_5^L \oplus K_5^L \oplus \widetilde{U}_4) \\
 &= G(U_5^L \oplus K_5^L \oplus (0, K_3^0, K_3^1, 0) \oplus \widetilde{U}_4)
 \end{aligned}$$

Enfin, nous obtenons la relation

$$\begin{aligned}
 &G(U_5^L \oplus K_5^L \oplus (0, K_3^0, K_3^1, 0) \oplus \widetilde{U}_4) \oplus G(V_5^L \oplus K_5^L \oplus (0, K_3^0, K_3^1, 0) \oplus \widetilde{V}_4) \\
 &= (\texttt{02 00 00 00}) \oplus (U_5^R \oplus V_5^R)
 \end{aligned}$$

avec $\widetilde{V}_4 = G(V_5^R \oplus K_5^R \oplus (0, K_4^0, K_4^1, 0))$. Cette relation ne dépend que des chiffrés, de la valeur $K_5^R \oplus (0, K_4^0, K_4^1, 0)$ (supposée connue) et de la valeur $K_5^L \oplus (0, K_3^0, K_3^1, 0)$ recherchée.

Un attaquant peut donc demander le chiffrement de deux messages U_0 et V_0 dont la différence est égale à $(\texttt{00 00 00 00}, \texttt{80 80 00 00})$. Il teste ensuite les 2^{32} valeurs possibles pour $K_5^L \oplus (0, K_3^0, K_3^1, 0)$ et vérifie si la relation est satisfaite. Comme dans la question précédente, cette étape réclame l'évaluation de la fonction de tour 2^{33} fois.

6. La méthode est encore similaire. En utilisant les distingueurs (à clairs connus) génériques sur les réseaux de Feistel à un et deux tours (*cf. Exercice (2.4)*), l'attaquant peut obtenir successivement les valeurs $K_5^R \oplus (0, K_2^0, K_2^1, 0)$ et $K_5^L \oplus (0, K_1^0, K_1^1, 0)$ en évaluant la fonction de tour 2^{33} fois. L'attaque demande deux clairs choisis pour déterminer chaque valeur. Enfin il est ensuite immédiat de déduire la valeur K_1 comme dans l'exercice (2.9). Le coût total de l'attaque est donc $4 \cdot 2^{33} = 2^{35}$ évaluations de la fonction F (soit 2^{33} chiffrements FEAL complets) et huit clairs choisis.

Note

L'attaque de l'exercice précédent peut être étendue en augmentant le nombre de tours du chiffrement. E. BIHAM et A. SHAMIR [8] ont en effet montré que FEAL est vulnérable à une attaque par cryptanalyse différentielle dans toutes ses versions allant jusqu'à 31 itérations de la fonction de tour.

3.5 CRYPTANALYSE DIFFÉRENTIELLE IMPOSSIBLE

La *cryptanalyse différentielle impossible* est une variante de la cryptanalyse différentielle qui a été proposée en 1999 par E. BIHAM, A. BIRYUKOV et A. SHAMIR [7]. Par opposition à la cryptanalyse différentielle qui observe le comportement de modifications de messages clairs dans la structure de chiffrement, la cryptanalyse différentielle impossible recherche des propagations qui ne se produisent jamais. L'attaque élimine ensuite les clés qui produisent un comportement considéré comme impossible. Nous allons voir deux exemples de telles cryptanalyses, contre DEAL un système de chiffrement par bloc dérivé de DES proposé par L. KNUDSEN en 1997 [37] et l'AES réduit à 5 tours.

Le système de chiffrement par bloc DEAL est basé sur un réseau de Feistel où la fonction de tour emploie la permutation DES. Il traite un bloc de 128 bits et fait appel à des clés de 128, 192 ou 256 bits. Le nombre de tours dépend de la taille de la clé : 6 pour 128 ou 192 bits, 8 pour 256 bits. Dans cet exercice, nous considérons uniquement la variante de DEAL à 6 tours avec 128 bits de clé.

Exercice 3.12 Attaque par différentielle impossible contre DEAL

1. Donner une attaque de complexité équivalente à $3 \cdot 2^{168}$ chiffrements DEAL contre le chiffrement par bloc DEAL à 6 tours indépendamment de l'algorithme de diversification de clé.
2. Dans cette question, nous considérons une version à 5 tours de DEAL. Montrer que pour deux couples clair/chiffré (X_0, X_5) et (Y_0, Y_5) où les clairs ont la même partie de droite et une différence $\Delta \neq 0$ dans la partie gauche (*i.e.* $(Y_0^L||Y_0^R) = (X_0^L \oplus \Delta)||X_0^R)$, il est impossible que les chiffrés aient la même partie de droite et la différence Δ dans la partie gauche (*i.e.* $(Y_5^L||Y_5^R) = (X_5^L \oplus \Delta)||X_5^R)$).
3. En déduire une attaque contre la version à 6 tours de DEAL qui permet de retrouver (indépendamment de l'algorithme de diversification de clé) la clé du dernier tour sous une attaque à 2^{70} clairs choisis en évaluant 2^{121} fois le chiffrement DES.

Solution

1. Notons tout d'abord qu'une recherche exhaustive des clés (indépendamment de l'algorithme de diversification de clé) demanderait de tester les $2^{56 \cdot 6} = 2^{336}$ clés possibles.

L'attaque est similaire à celle vue dans l'exercice (2.11) contre le double chiffrement : DEAL est vulnérable à une attaque de type *rencontre au milieu*. Étant donné

un couple clair/chiffré (m, c) pour **DEAL**, les clés utilisées à chaque tour du schéma de Feistel peuvent être obtenues de la façon suivante :

- Calculer pour toutes les clés possibles $K_1, K_2, K_3 \in \{0, 1\}^{56}$ le chiffré partiel \tilde{c} obtenu à partir de m après trois tours d'un schéma de Feistel où la fonction de tour pour le tour i est le **DES** paramétré par la clé K_i (pour $i \in \{1, 2, 3\}$) et stocker les valeurs $(\tilde{c}, K_1, K_2, K_3)$ obtenues dans une table de hachage (indexée par \tilde{c}).
- Calculer pour toutes les clés possibles $K_4, K_5, K_6 \in \{0, 1\}^{56}$ le clair partiel \tilde{m} obtenu à partir de c après inversion de trois tours d'un schéma de Feistel où la fonction de tour pour le tour i est le **DES** paramétré par la clé K_{i+3} (pour $i \in \{1, 2, 3\}$) et chercher si \tilde{m} apparaît comme un \tilde{c} dans la table de hachage.
- Pour chaque occurrence obtenue $\tilde{c} = \tilde{m}$, retourner $(K_1, K_2, K_3, K_4, K_5, K_6)$ où (K_1, K_2, K_3) est la clé associée à \tilde{c} dans la table de hachage et (K_4, K_5, K_6) est la clé associée au déchiffrement partiel de c ayant produit \tilde{m} .

Cet algorithme retourne toutes les clés possibles pour **DEAL** qui chiffre m en c en calculant 2^{168} chiffrement partiel dans l'étape (a) et 2^{168} déchiffrement partiel dans l'étape (b). Le coût d'un (dé)chiffrement étant égal à la moitié d'un chiffrement **DEAL** complet, le coût de cette attaque est équivalent à 2^{168} chiffrement **DEAL**.

Le nombre de clés obtenues par cette attaque sera égal en moyenne à $2^{56 \cdot 6}/2^{128} = 2^{208}$. Il est donc nécessaire d'utiliser plus d'informations pour filtrer ces clés. La méthode la plus simple est d'adapter cette méthode en utilisant deux couples clair/chiffré dans la table de hachage. La complexité de l'attaque est multipliée par 3 mais le nombre de clés incorrectes finalement obtenues sera de l'ordre de $2^{56 \cdot 6}/2^{128 \cdot 3} = 2^{-48} \simeq 0$.

- Nous avons $(Y_0^L || Y_0^R) = ((X_0^L \oplus \Delta) || X_0^R)$. Donc $\text{DES}_{K_1}(X_0^R) = \text{DES}_{K_1}(Y_0^R)$ et

$$Y_0^L \oplus \text{DES}_{K_1}(Y_0^R) = (X_0^L \oplus \Delta) \oplus \text{DES}_{K_1}(X_0^R)$$

$$\text{et } (Y_1^L || Y_1^R) = X_1^L || (X_1^R \oplus \Delta).$$

Puisque $\Delta \neq 0$ et **DES** est une permutation, nous avons

$$\text{DES}_{K_2}(Y_1^R) = \text{DES}_{K_2}(X_1^R \oplus \Delta) = \text{DES}_{K_2}(X_1^R) \oplus \Delta_1$$

avec $\Delta_1 \neq 0$. Nous en déduisons $(Y_2^L || Y_2^R) = (X_2^L \oplus \Delta) || (X_2^R \oplus \Delta_1)$. De même, puisque $\Delta_1 \neq 0$ nous avons $\text{DES}_{K_3}(Y_2^R) = \text{DES}_{K_3}(X_2^R \oplus \Delta_1) = \text{DES}_{K_3}(X_2^R) \oplus \Delta_2$ avec $\Delta_2 \neq 0$. Nous en déduisons $Y_3^R = X_3^R \oplus \Delta \oplus \Delta_2$.

Supposons par l'absurde que $(Y_5^L || Y_5^R) = ((X_5^L \oplus \Delta) || X_5^R)$, nous obtenons de même $(Y_4^L || Y_4^R) = (X_4^L \oplus \Delta) || X_4^R$ puis $(Y_3^L || Y_3^R) = X_3^L || (X_3^R \oplus \Delta)$.

Nous avons finalement $Y_3^R = X_3^R \oplus \Delta = X_3^R \oplus \Delta \oplus \Delta_2$, ce qui contredit $\Delta_2 \neq 0$. Nous obtenons donc bien le résultat demandé (qui est également représenté sur la figure (3.8)).

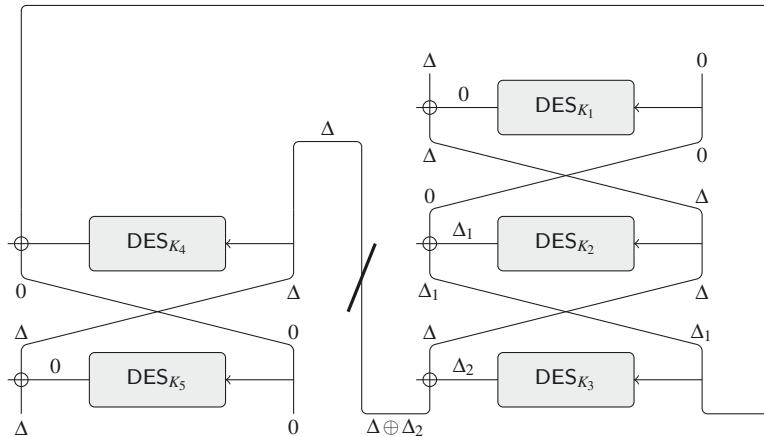


Figure 3.8 - Attaque par différentielle impossible contre DEAL réduit à 5 tours

3. Il suffit d'utiliser la remarque précédente pour filtrer les clés possibles pour le dernier tour de DEAL. L'attaquant demande le chiffrement de T messages $X_0^{(i)}$ pour $i \in \{0, \dots, T\}$ qui ont tous la même partie droite. Les chiffrés obtenus sont notés $X_6^{(i)}$ pour $i \in \{0, \dots, T\}$. Pour les 2^{56} clés possibles de dernier tour, l'attaquant déchiffre partiellement les chiffrés $X_6^{(i)}$ pour obtenir les valeurs $X_5^{(i)}$ pour $i \in \{0, \dots, T\}$. Il teste ensuite pour tous les couples $(X_5^{(i)}, X_5^{(j)})$ avec $i \neq j$ si la différence de la partie gauche des déchiffrés partiels $X_5^{(i)} \oplus X_5^{(j)}$ est égale à la différence des parties gauches initiales $X_0^{(i)} \oplus X_0^{(j)}$ et si les parties droites de $X_5^{(i)}$ et $X_5^{(j)}$ sont égales. Si les deux conditions sont satisfaites, alors la clé testée peut être éliminée de la liste des clés valides.

Il reste à déterminer la valeur de T pour laquelle l'attaque fonctionne correctement. La différence des parties gauches des déchiffrés partiels est évidemment égale à la différence des parties droites des chiffrés $X_5^{(i)R} \oplus X_5^{(j)R} = X_6^{(i)R} \oplus X_6^{(j)R}$ donc le premier test peut être effectué avant même d'effectuer le déchiffrement partiel. Le nombre de couples $(X_0^{(i)}, X_0^{(j)})$ tels que

$$X_0^{(i)L} \oplus X_0^{(j)L} = X_6^{(i)R} \oplus X_6^{(j)R}$$

sera en moyenne égal à $n = T(T-1)/2^{64}$. La probabilité qu'une clé incorrecte testée donne la relation $X_5^{(i)R} = X_5^{(j)R}$ est alors égale à $n/2^{64}$.

Choisissons $T = 2^{64}$, nous obtenons $n \approx 2^{63}$ et la méthode éliminera environ 50% des clés. Pour éliminer les clés incorrectes restantes, il faut répéter l'attaque k fois avec une nouvelle famille de clairs choisis à chaque fois (avec $k \geq 1$). Le nombre de clés incorrectes restantes sera alors de l'ordre de $2^{56}/2^k$, le nombre de clairs choisis égal à $k \cdot 2^{64}$ et le nombre de chiffrement DES égal à

$$(2^{56} + 2^{55} + \dots + 2^{56-k+1}) \cdot 2^{64}$$

En choisissant $k = 56$, le nombre de clés incorrectes non éliminées sera en moyenne égal à 1, l'attaque aura demandé $56 \cdot 2^{64} \simeq 2^{70}$ clairs choisis et environ $2^{57} \cdot 2^{64} \simeq 2^{121}$ chiffrements DES.

Dans l'exercice suivant, nous considérons une cryptanalyse par différentielle impossible (à clairs choisis) contre deux variantes de l'AES réduit à 4 ou 5 tours. Pour ces variantes de l'AES, nous considérons qu'il n'y a pas d'opération MixColumns dans le dernier tour (comme c'est le cas pour le chiffrement AES complet).

Problème 3.13 Attaque par différentielle impossible contre l'AES

- Montrer que si un attaquant obtient le chiffré AES réduit à 4 tours de deux clairs qui ne diffèrent que d'un octet, les chiffrés obtenus ne peuvent pas prendre la même valeur aux quatre octets indiqués par les positions (1, 8, 11, 14), (2, 5, 12, 15), (3, 6, 9, 16), ou (4, 7, 10, 13) (où les octets sont numérotés de gauche à droite et de bas en haut comme sur la figure (2.10)).
- Proposer une attaque contre l'AES réduit à 5 tours en utilisant cette propriété pour éliminer des clés possibles utilisées lors du premier tour.

L'attaque cherchera à construire, pour chaque position de la matrice état, des couples de blocs qui après application des opérations SubBytes, ShiftRows et MixColumns donnent deux blocs qui ne diffèrent que dans cette position. Il sera utile de stocker ces couples en utilisant une table de hachage indexée par leur différence et contenant l'un des blocs en entrée.

Solution

- Considérons le chiffrement AES de deux clairs qui ne diffèrent que d'un octet (*cf.* Figure (3.9)), pour une illustration où l'on suppose que les clairs diffèrent en le premier octet et que les chiffrés sont égaux aux quatre octets des positions (1, 8, 11, 14)). Nous avons les propriétés suivantes :
 - L'opération AddRoundKey initiale (ARK) (*i.e.* le « ou exclusif » avec la clé du tour d'initialisation) produit deux blocs qui ne diffèrent qu'en cet octet.
 - L'application successive des opérations SubBytes (SB) et ShiftRows (SR) produit également deux blocs qui ne diffèrent qu'en un octet (éventuellement déplacé).
 - L'application de l'opération Mixcolumns (MC) à ces deux blocs donne deux nouveaux blocs pour lesquelles trois colonnes ont quatre valeurs identiques alors que les quatre octets d'une quatrième colonne sont tous différents. (*cf.* Exercice (2.16)).

3.5. Cryptanalyse différentielle impossible

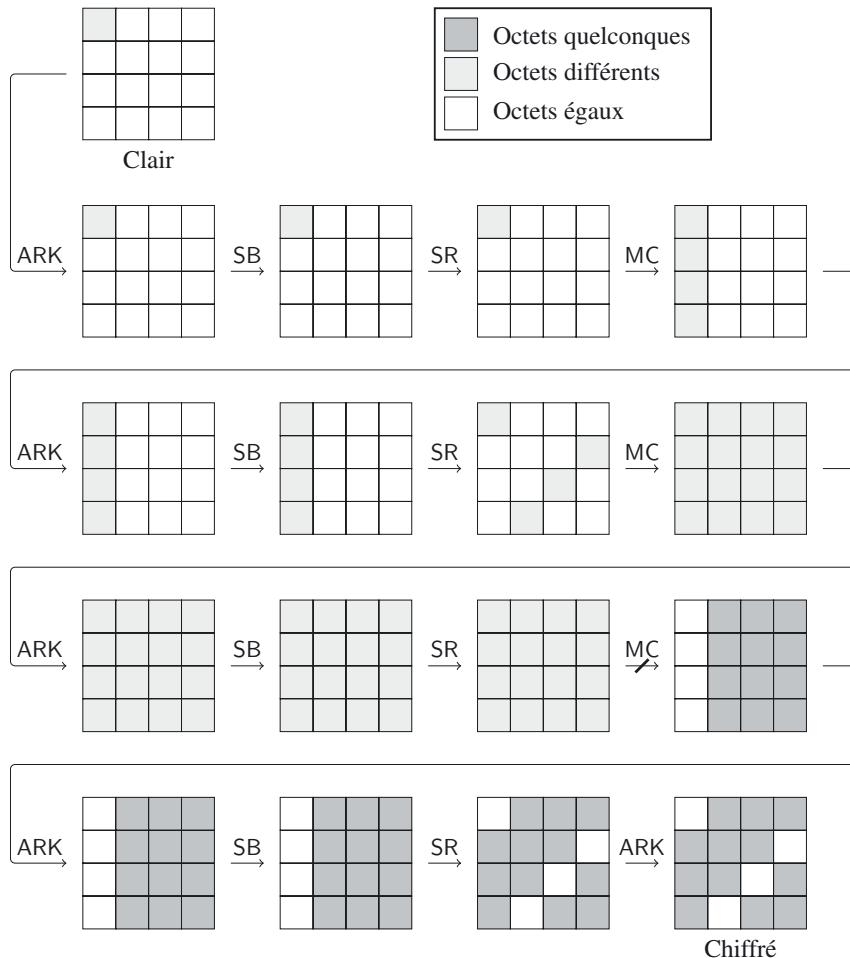


Figure 3.9 - Attaque par différentielle impossible contre AES réduit à 4 tours

- Les opérations AddRoundKey et SubBytes du second tour produisent un couple de blocs du même type.
- L'opération ShiftRows suivante produit deux blocs où chaque colonne et chaque ligne contient trois positions où les deux octets sont égaux et une position où ils diffèrent.
- L'application de l'opération Mixcolumns du deuxième tour donne deux blocs où les seize octets prennent des valeurs différentes.
- Les opérations AddRoundKey, SubBytes et ShiftRows produisent deux blocs du même type.

Les deux valeurs en entrée de l'opération **Mixcolumns** du troisième tour diffèrent donc en tous leurs octets.

Supposons que les chiffrés AES produits à partir de ces deux clairs possèdent des octets identiques en l'une des positions suivantes : (1, 8, 11, 14), (2, 5, 12, 15), (3, 6, 9, 16) ou (4, 7, 10, 13).

- L'inversion de l'opération **AddRoundKey** du quatrième tour fournit deux blocs du même type (*i.e.* des octets identiques en l'une des positions suivantes : (1, 8, 11, 14), (2, 5, 12, 15), (3, 6, 9, 16) ou (4, 7, 10, 13)).
- L'inversion de l'opération **ShiftRows** produit deux blocs qui ont une colonne (ou plusieurs) possédant les mêmes valeurs pour les quatre octets (rappelons qu'il n'y a pas de mélange de colonnes lors du dernier tour de l'AES).
- L'inversion des opérations **SubBytes** et **AddRoundKey** du troisième tour donnent encore deux blocs du même type.

Dans ce cas, les deux valeurs en sortie de l'opération **Mixcolumns** du troisième tour ont une colonne possédant les mêmes valeurs pour les quatre octets. L'inversion de l'opération **Mixcolumns** donnerait donc une configuration où les deux blocs possèdent une colonne égale, ce qui contredit la conclusion précédente.

2. L'attaque contre l'AES réduit à cinq tours utilise la différentielle impossible précédente pour éliminer des mauvaises clés lors du premier tour.

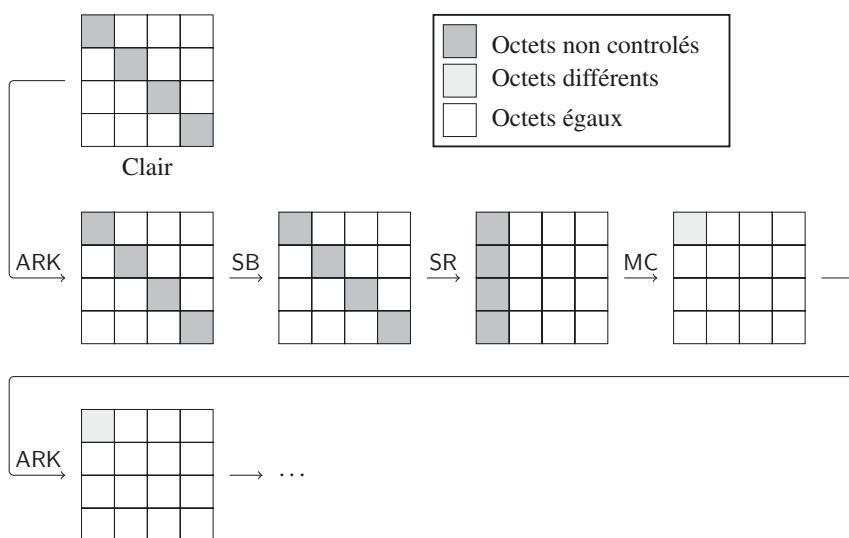


Figure 3.10 – Attaque contre AES réduit à 5 tours

Suivant l'indication, dans une première étape de pré-calcul, l'attaquant choisit une colonne et une position de cette colonne (sur l'illustration de la figure (3.10), il s'agit de l'octet en position 1) et il considère tous les couples de blocs qui diffèrent en cette position, prennent les mêmes valeurs aux trois autres positions de la colonne et une valeur constante pour les douze autres octets de la matrice état. Nous obtenons ainsi $2^8 \cdot (2^8 - 1) \cdot 2^{24} \simeq 2^{40}$ couples différents.

3.5. Cryptanalyse différentielle impossible

Pour ces couples de bloc, il inverse les opérations MixColumns, ShiftRows et SubBytes et construit une table de hachage

- indexée par le « ou exclusif » des deux entrées de 32 bits correspondant à la colonne choisie (il s'agit de la diagonale principale sur la figure (3.10)) du premier tour
- contenant l'une des deux entrées comme valeur (l'autre pouvant bien sûr être reconstruite à partir de la clé de hachage).

L'attaquant dispose à la fin de cette première étape d'une table contenant des valeurs $(r, (x_1, \dots, x_t))$ telles que pour tout $i \in \{1, \dots, t\}$, x_i et $r \oplus x_i$ donnent après l'application des opérations SubBytes, ShiftRows et MixColumns du premier tour deux blocs qui diffèrent en la position considérée et prennent les mêmes valeurs aux trois autres positions de la colonne. Il y a 2^{32} valeurs possibles pour r et environ 2^{40} valeurs prises par les x_i , donc en moyenne 2^8 valeurs de x sont associées à chaque clé de hachage r .

L'attaquant considère ensuite les 2^{32} clairs choisis qui prennent toutes les valeurs sur les 4 octets correspondant à l'inversion du chiffrement du premier tour de la colonne considérée (*i.e.* la diagonale principale sur la figure (3.10)) et une valeur constante dans les autres. Il y a environ $(2^{32})^2/2 = 2^{63}$ couples de clairs et donc environ $2^{63}/2^{30} = 2^{33}$ couples de clairs pour lesquels les chiffrés sont égaux dans au moins une des configurations de la question précédente.

Pour ces couples, nous calculons le « ou exclusif » des octets contenus dans les quatre positions et nous utilisons la table de hachage pour trouver les éléments qui correspondent à cette valeur. Si nous notons m_1 et m_2 les clairs, nous avons dans la table de hachage $(m_1 \oplus m_2, x_1, \dots, x_t)$ et par la propriété de différentielle impossible, nous savons que x_i ne peut pas être l'image de m_1 après l'ajout de la clé du premier tour. En calculant le « ou exclusif » de m_1 et de x_i pour $i \in \{1, \dots, t\}$, ce procédé permet d'éliminer t fausses clés avec $t = 2^8$ en moyenne. En considérant les collisions parmi les fausses clés éliminées, le nombre de fausses clés restantes après avoir traité tous les couples de chiffrés est :

$$2^{32}(1 - 2^{-32})^{2^{33}2^8} \simeq 2^{32}(\exp(-1))^{2^9} = 2^{32} \exp(-512) \simeq 0$$

Cette attaque à 2^{32} clairs choisis permet de retrouver 32 bits de clé en déchiffrant une colonne d'un tour d'AES pour 2^{40} couples de chiffrés (ce qui correspond à $2^{41}/(4 \cdot 5) = 2^{37}$ chiffrements complets) et en gérant une table de hachage de taille 2^{50} bits. En faisant varier la colonne sélectionnée pour l'attaque, il est possible d'obtenir les 128 bits de clé du premier tour.

Note

Il existe d'autres variantes de la cryptanalyse différentielle, comme la *cryptanalyse différentielle d'ordre supérieur* et la *cryptanalyse différentielle tronquée* introduites par L. KNUDSEN en 1994 [38] ou l'attaque boomerang présentée par D. WAGNER en 1999 [68].

3.6 CRYPTANALYSE LINÉAIRE

La *cryptanalyse linéaire* a été formalisée par M. MATSUI en 1993 [44]. Elle consiste à faire une approximation linéaire de certains tours de l'algorithme de chiffrement en le simplifiant puis, d'utiliser cette approximation pour faire des tests d'hypothèse sur tout ou partie des clés du dernier tour. La cryptanalyse linéaire permet de réaliser des attaques à clairs connus (là où la cryptanalyse différentielle demande généralement une attaque à clairs choisis).

Étant donné une S -boîte de $\{0, 1\}^m \rightarrow \{0, 1\}^n$, la *table d'approximation linéaire* de S est l'ensemble des valeurs entières $(N(\alpha, \beta) - 2^{m-1})$ pour $\alpha \in \{0, 1\}^m$ et $\beta \in \{0, 1\}^n$, où

$$N(\alpha, \beta) = \{i \in \{0, 1\}^m \mid \alpha \cdot i \oplus \beta \cdot S(i) = 0^n\}$$

Pour qu'un système de chiffrement par bloc soit résistant à la cryptanalyse linéaire, il faut que la valeur absolue des éléments de la table d'approximation linéaire soit la plus petite possible (en valeur absolue).

Exercice 3.14 (avec programmation). Table d'approximation linéaire du DES

Construire la table d'approximation linéaire de la boîte S_1 du chiffrement DES (cf. Tableau (3.1), p. 82).

Solution

Une recherche informatique montre que, par exemple pour $\alpha = 111111$ et $\beta = 0001$, nous avons $\alpha \cdot i = \beta \cdot S_1(i)$ pour les 18 valeurs

$$i \in \{000000, 000100, 001001, 001101, 010000, 010001, 010010, 010100, 011001, 011010, 100100, 100111, 101111, 110001, 110100, 111000, 111001, 111010\}$$

(ce qui donne la valeur -14 dans la table d'approximation linéaire). Par une recherche exhaustive, nous obtenons la table d'approximation linéaire de S_1 (cf. Tableau (3.3)) où la valeur à l'intersection de la ligne indexée par $\alpha \in \{0, 1\}^6$ et la colonne indexée par $\beta \in \{0, 1\}^4$ est égale à $\#N(\alpha, \beta) - 2^5 = \#N(\alpha, \beta) - 32$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
000010	0	-2	-2	-4	-2	0	-4	6	2	0	0	6	4	-2	-6	4
000011	0	-2	-2	-4	-2	0	-4	6	2	8	0	-2	4	6	-6	-4
000100	0	2	-2	-4	-2	0	-4	-6	-2	4	8	2	0	-2	-6	12
000101	0	-2	-2	0	-2	-4	-4	-2	2	-4	-4	2	4	-10	-2	-4
000110	0	0	0	4	0	4	0	0	0	-4	4	4	0	0	-4	-8
000111	0	-4	0	8	0	0	0	4	4	-4	-8	-4	4	0	0	0
001000	0	4	-2	6	-6	0	-4	-4	-4	2	-2	2	-2	0	0	0
001001	0	0	6	-6	-2	-6	4	-4	0	-4	-2	6	2	-6	0	-4
001010	0	-2	0	2	0	6	8	2	-2	0	-2	4	-2	0	-2	4
001011	0	2	-8	-2	-4	-10	4	2	-6	8	2	4	-2	-4	-2	0
001100	0	-2	0	6	0	2	0	2	2	0	6	-4	2	-4	6	0
001101	0	6	0	6	4	-2	-4	-2	2	0	6	4	-2	8	-6	-4
001110	0	0	-2	-2	2	2	0	0	4	4	6	-2	2	2	-4	4
001111	0	0	-2	6	-2	-2	4	-4	-4	-4	-2	-2	-2	-2	0	0
010000	0	2	2	0	-2	0	4	-6	0	6	2	-4	6	-4	-4	-18
010001	0	2	-2	-4	2	-4	-4	10	-4	2	2	-4	-2	-4	0	-6
010010	0	4	0	0	-4	4	0	4	-6	2	2	6	2	6	6	-10
010011	0	4	-4	-4	0	0	-8	-12	-2	-2	-6	6	2	6	2	2
010100	0	4	0	4	-8	-4	4	0	2	6	-2	2	6	2	-2	2
010101	0	0	4	-4	-4	4	4	-4	10	2	2	2	-6	2	6	-2
010110	0	6	2	0	2	-4	0	2	4	2	2	0	-2	0	0	2
010111	0	2	6	-8	6	4	0	-2	-12	-2	-2	0	-6	0	0	-2
011000	0	2	8	2	0	6	4	2	4	-2	4	6	0	-2	-4	2
011001	0	-2	4	-6	0	-6	0	2	4	-6	8	6	0	2	0	-6
011010	0	0	-6	2	-2	-2	4	4	-2	-2	0	0	-4	4	2	2
011011	0	4	6	2	-10	2	-8	4	-2	-6	4	0	4	0	-2	2
011100	0	-4	2	2	2	-6	0	-4	-2	-2	4	0	0	4	2	2
011101	0	4	-2	-2	2	-6	-4	0	2	2	-4	0	-12	0	-6	-6
011110	0	2	0	-2	4	-2	0	-2	0	6	-4	-2	0	-2	0	2
011111	0	2	-4	2	-4	-2	4	2	4	-6	4	-2	-4	2	0	2

Tableau 3.3 - Table d'approximation linéaire de S_1 (extrait)**Note**

Le système de chiffrement DES n'a pas été conçu pour prévenir ce type d'attaques. M. MATSUI [44] a obtenu une équation linéaire qui lie les entrées et les sorties de quinze tours du DES qui est vérifiée avec une probabilité égale à $1/2 + 2^{-22}$. Cette équation permet d'attaquer le DES en 2^{43} chiffrements à l'aide de 2^{43} couples de clairs/chiffrés. Les algorithmes plus récents comme l'AES ont été conçus pour être résistants à la cryptanalyse linéaire.

Étant données deux fonctions booléennes $f, g : \{0, 1\}^m \rightarrow \{0, 1\}$, nous notons

$$d(f, g) = \#\{\alpha \in \{0, 1\}^m, f(\alpha) \neq g(\alpha)\}$$

Il s'agit d'une distance définie sur l'ensemble des fonctions booléennes, appelée *distance de Hamming*. Il est facile de vérifier que d vérifie l'inégalité triangulaire. Une fonction affine g est une *meilleure approximation affine* de f si $d(f, g)$ est minimale parmi toutes les distances de Hamming de la fonction f à une fonction affine. L'exercice suivant montre que l'addition modulaire admet une meilleure approximation linéaire très simple.

Exercice 3.15 Approximation linéaire de l'addition

Soit $n \geq 1$ un entier. Considérons la fonction booléenne

$$f_n : \begin{cases} \{0, 1\}^{2n+2} & \longrightarrow \{0, 1\} \\ (x_0, \dots, x_n, y_0, \dots, y_n) \longmapsto z_n \end{cases}$$

où $z_{n+1}z_n \dots z_0$ est la représentation binaire de la somme dans \mathbb{N} des entiers x et y représentés par $x_n \dots x_0$ et $y_n \dots y_0$ (respectivement).

- Montrer que la fonction booléenne définie par

$$g_n : \begin{cases} \{0, 1\}^{2n+2} & \longrightarrow \{0, 1\} \\ (x_0, \dots, x_n, y_0, \dots, y_n) \longmapsto x_n \oplus y_n \oplus x_{n-1} \end{cases}$$

vérifie $d(f_n, g_n) = 2^{2n}$.

- En déduire que g_n est une meilleure approximation affine de f_n .

Solution

- Considérons $\Omega = \{0, 1\}^{2n+2}$ muni de la probabilité uniforme et la variable aléatoire c_k pour $k \in \{0, \dots, n\}$ la retenue obtenue au k -ième bit lors du calcul de la somme dans \mathbb{N} des entiers x et y représentés par $x_n \dots x_0$ et $y_n \dots y_0$ (respectivement). Nous avons

$$\begin{aligned} & \Pr_{\Omega}[f_n(x, y) \neq g_n(x, y)] \\ &= \Pr_{\Omega}[x_n \oplus y_n \oplus c_n \neq x_n \oplus y_n \oplus x_{n-1}] \\ &= \Pr_{\Omega}[x_{n-1} = 1 \wedge c_n = 0] + \Pr_{\Omega}[x_{n-1} = 0 \wedge c_n = 1] \\ &= \Pr_{\Omega}[x_{n-1} = 1 \wedge y_{n-1} = 0 \wedge c_{n-1} = 0] + \Pr_{\Omega}[x_{n-1} = 0 \wedge y_{n-1} = 1 \wedge c_{n-1} = 1] \\ &= \Pr_{\Omega}[x_{n-1} = 1] \cdot \Pr_{\Omega}[y_{n-1} = 0] \cdot \Pr_{\Omega}[c_{n-1} = 0] \\ &\quad + \Pr_{\Omega}[x_{n-1} = 0] \cdot \Pr_{\Omega}[y_{n-1} = 1] \cdot \Pr_{\Omega}[c_{n-1} = 1] \\ &= \frac{1}{4} \Pr_{\Omega}[c_{n-1} = 0] + \frac{1}{4} \Pr_{\Omega}[c_{n-1} = 1] = \frac{1}{4} \end{aligned}$$

et nous obtenons $d(f_n, g_n) = \#\Omega/4 = 2^{2n+2}/4 = 2^{2n}$.

- Soit h_n une fonction affine de $\{0, 1\}^{2n+2} \longrightarrow \{0, 1\}$ différente de g_n . Nous avons $d(h_n, g_n) = 2^{2n+2}/2 = 2^{2n+1}$. Par l'inégalité triangulaire, nous obtenons

$$d(f_n, h_n) \geq |d(f_n, g_n) - d(g_n, h_n)| = 2^{2n+1} - 2^{2n} = 2^{2n} = d(f_n, g_n),$$

donc g_n est bien une meilleure approximation linéaire de f_n .

Note

L'exercice précédent montre que, pour $n = 7$, la valeur $(x \oplus y \oplus (x \ll 1))$ est une bonne approximation linéaire de $x + y \bmod 256$ et que $(x \oplus y \oplus (x \ll 1) \oplus 1)$ est une bonne approximation linéaire de $x + y + 1 \bmod 256$. A. TARDY-CORFDIR et H. GILBERT ont utilisé cette approximation linéaire pour réaliser une attaque à clairs connus par cryptanalyse linéaire lors de la conférence *Crypto'91* [63] sur le système de chiffrement FEAL.

L'analyse d'une attaque par cryptanalyse linéaire repose sur le *lemme d'empilement* (*piling-up lemma*, en anglais). Considérons une suite de variables aléatoires X_1, \dots, X_n qui prennent uniquement les valeurs 0 et 1 et posons $\epsilon_i = \Pr[X_i = 0] - 1/2$. La valeur ϵ_i est le *biais* de X_i .

Lemme d'empilement. En notant $\epsilon_{1,\dots,n}$ le biais de la variable aléatoire $X_1 \oplus \dots \oplus X_n$, nous avons

$$\epsilon_{1,\dots,n} = 2^{n-1} \prod_{i=1}^n \epsilon_i$$

Ce lemme permet d'estimer la qualité d'une approximation linéaire d'un système de chiffrement itératif en considérant les qualités des approximations linéaires tour par tour.

Le système de chiffrement par bloc **SAFER** (pour *Secure And Fast Encryption Routine*, en anglais) a été proposé par J. MASSEY en 1993 [43] (cf. Figure (3.11)). Il s'agit d'un réseau de substitutions-permutations qui utilise une clé de 64 bits pour chiffrer des messages de 64 bits (interprétés comme 8 octets dans $\{0, \dots, 255\}$). La confusion est assurée par l'addition modulo 256 (ou le « ou exclusif ») avec une clé de tour, une permutation P et sa réciproque Q . La diffusion est assurée par la fonction linéaire

$$L(a, b) = (L_1(a, b), L_2(a, b)) = (2a + b, a + b) \bmod 256$$

La clé est représentée par 8 entiers k_1, \dots, k_8 sur 8 bits et un procédé de diversification de clé fabrique des sous-clés k_1^i, \dots, k_8^i pour $i \in \{1, \dots, 13\}$. Pour l'exercice, il suffit de savoir que chaque clé k_j^i dépend uniquement de la valeur k_j (pour $i \in \{1, \dots, 13\}$).

Les deux exercices suivants analysent une attaque par cryptanalyse linéaire de **SAFER** due à S. VAUDENAY [65] utilisant une approximation linéaire du chiffrement en considérant le biais

$$\epsilon = \left(\Pr_x[x \equiv P(x) \bmod 2] - \frac{1}{2} \right) = \left(\Pr_x[x \equiv Q(x) \bmod 2] - \frac{1}{2} \right)$$

qui mesure la dépendance entre $x \bmod 2$ et $P(x) \bmod 2$.

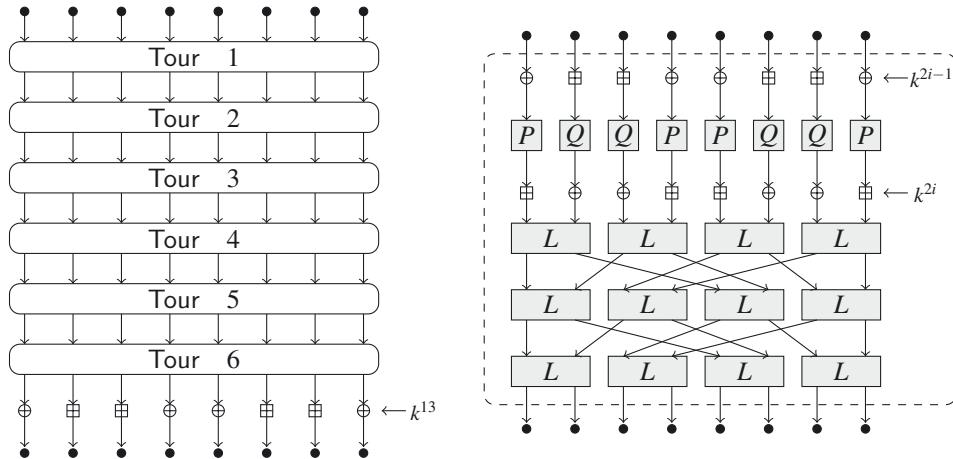


Figure 3.11 – Description du système de chiffrement par bloc SAFER

Problème 3.16 Cryptanalyse linéaire de SAFER

- Notons $(y_1, \dots, y_8) \in \{0, \dots, 255\}^8$ une entrée des boîtes de diffusion (i.e. les trois applications de la fonction L représentées sur la figure (3.11)) et $(z_1, \dots, z_8) \in \{0, \dots, 255\}^8$ la sortie correspondante. Montrer que

$$z_3 + z_4 \equiv y_3 + y_4 \pmod{2}$$

Soient $(\alpha_1, \dots, \alpha_8)$ un texte clair et $(\beta_1, \dots, \beta_8)$ le chiffré obtenu en appliquant SAFER avec une clé k . Notons $(\theta_3, \theta_4) = (Q(\alpha_3 + k_3^1 \pmod{256}), P(\alpha_4 \oplus k_4^1))$, $\varphi(k)$ la parité de la somme des clés k_3^i et k_4^i pour $i \in \{2, \dots, 13\}$ et t_3^i (*resp.* t_4^i) le changement de parité entre l'entrée et la sortie de P (*resp.* Q) pour le troisième octet (*resp.* le quatrième octet) au tour i pour $i \in \{1, \dots, 6\}$.

- Exprimer $\beta_3 + \beta_4 \pmod{2}$ en fonction de $\theta_3, \theta_4, \kappa$ et de t_3^i et t_4^i pour $i \in \{1, \dots, 6\}$.
- Posons $b(\alpha, k) = (\theta_3 + \theta_4 + \beta_3 + \beta_4) \pmod{2}$. Montrer que

$$\Pr[b(\alpha, k) = \varphi(k_3, k_4)] = \frac{1}{2} (1 \pm (2\epsilon)^{10})$$

- Considérons maintenant une clé $k' \neq k$ et notons comme précédemment $(\theta'_3, \theta'_4) = (\alpha_3 + k'_3 \pmod{256}, \alpha_4 \oplus k'_4)$, $\varphi(k')$, t_3^i , t_4^i , avec les notations naturelles. Montrer que

$$\Pr[b(\alpha, k') = \varphi(k'_3, k'_4)] = \frac{1}{2} (1 \pm (2\epsilon)^{10+i})$$

avec $i = 1$ si $k'_3 = k_3$ ou $k'_4 = k_4$ et $i = 2$ sinon.

5. Expliquer comment utiliser cette propriété pour réaliser une attaque à clairs connus contre SAFER.

Solution

1. Nous avons

$$(z_3, z_4) = L(L_1(L_2(y_1, y_2), L_2(y_3, y_4)), L_1(L_2(y_5, y_6), L_2(y_7, y_8)))$$

En remarquant que $L_1(x, y) = 2x + y \bmod 256 = y \bmod 2$ et $L_1(x, y) + L_2(x, y) = (2x + y) + (x + y) \bmod 256 = 3x + y \bmod 256 = x \bmod 2$, nous obtenons donc

$$\begin{aligned} z_3 + z_4 &\equiv (L_1(L_2(y_1, y_2), L_2(y_3, y_4))) \bmod 2 \\ &\equiv L_2(y_3, y_4) \bmod 2 \\ &\equiv (y_3 + y_4 \bmod 256) \bmod 2 = y_3 + y_4 \bmod 2 \end{aligned}$$

2. Il suffit d'appliquer la propriété de la question précédente pour chacun des tours de SAFER. Ainsi pour le premier tour, l'entrée des boîtes de diffusion pour les troisième et quatrième octets est égale à

$$(Q(\theta_3) \oplus k_3^2, P(\theta_4) \oplus k_4^2)$$

La somme de la sortie de ces boîtes de diffusion est donc congrue à

$$Q(\theta_3) \oplus k_3^2, P(\theta_4) \oplus k_4^2 \bmod 2$$

soit

$$\theta_3 + k_3^2 + \theta_4 + k_4^2 + t_3^1 + t_4^1$$

En appliquant cet argument à chaque tour de SAFER, nous obtenons que la parité de $\beta_3 \oplus \beta_4$ est égale à la parité de $\theta_3 + \theta_4$ plus la parité de la somme des clés k_3^i et k_4^i pour $i \in \{2, \dots, 13\}$ plus la parité du nombre de changement de parité pour chaque application des permutations P et Q . Nous obtenons donc

$$\beta_3 + \beta_4 = \theta_3 + \theta_4 + \varphi(k) + \sum_{i=2}^6 (t_3^i + t_4^i) \bmod 2$$

3. D'après la question précédente, nous avons $b(\alpha, k) = \varphi(k_3, k_4)$ si et seulement si

$$\sum_{i=2}^6 (t_3^i + t_4^i) \bmod 2 = 0$$

et le résultat découle immédiatement du lemme d'empilement (les fonctions P et Q sont appliquées 5 fois chacune).

4. L'analyse est identique à celle des deux questions précédentes, mais il faut tenir compte en plus du changement de parité éventuel supplémentaire entre

- $\theta_3 = Q(\alpha_3 + k_3^1 \bmod 256)$ et $\theta'_3 = Q(\alpha_3 + k'_3^1 \bmod 256)$ si $k_3 \neq k'_3$
- $\theta_4 = P(\alpha_4 \oplus k_4^1)$ et $\theta'_4 = P(\alpha_4 \oplus k'_4^1)$ si $k_4 \neq k'_4$

Un tel changement de parité se produit avec probabilité $1/2 + \epsilon_P$. Nous obtenons ainsi

$$\Pr[b(\alpha, k') = \varphi(k'_3, k'_4)] = \frac{1}{2} (1 \pm (2\epsilon)^{10+i})$$

avec $i = 1$ si $k'_3 = k_3$ ou $k'_4 = k_4$ et $i = 2$ sinon.

5. Étant donnée une collection de n couples clairs/chiffrés connus, un attaquant peut (pour tous les couples d'octets de clés possibles pour k_3 et k_4), calculer $c(k)$ le nombre de couples de clairs chiffrés vérifiant $b(\alpha, k) = \varphi(k_3, k_4)$. Si pour une clé k , $c(k)$ est très supérieur (ou très inférieur) aux autres valeurs $c(k')$ pour $k' \neq k$, alors cette clé est vraisemblablement celle qui a été utilisée pour le chiffrement. La variabilité des valeurs $c(k)$ dépend très fortement de la valeur de ϵ (plus ϵ est grand, plus l'attaque sera efficace).

Note

Il existe d'autres dépendances linéaires entre les entrées et les sorties des boîtes de confusions qui permettent de retrouver les autres octets de clés :

$$\begin{aligned} z_2 + z_6 &\equiv y_2 + y_6 \bmod 2 \\ z_5 + z_7 &\equiv y_5 + y_7 \bmod 2 \\ z_3 + z_7 &\equiv y_5 + y_6 \bmod 2 \\ z_5 + z_6 &\equiv y_2 + y_4 \bmod 2 \\ z_2 + z_4 &\equiv y_3 + y_7 \bmod 2 \end{aligned}$$

Comme nous l'avons vu, pour que l'attaque de l'exercice précédent fonctionne, il faut que le biais ϵ soit suffisamment grand. Nous allons étudier la valeur de ϵ pour une permutation P aléatoire et pour la permutation P effectivement utilisée dans SAFER.

Exercice 3.17 Biais de la parité d'une permutation

1. Soit P une permutation aléatoire de $\{0, \dots, n-1\}$ où n est un multiple de 4. Montrer que pour tout entier $k \in \{-n/4, \dots, n/4\}$,

$$\Pr_P \left[\epsilon_P = \frac{2k}{n} \right] = \binom{n/2}{n/4+k}^4 \frac{[(n/4+k)!]^2 [(n/4-k)!]^2}{n!}$$

2. La permutation P utilisée dans SAFER est définie par $P(a) = (45^a \bmod 257) \bmod 256$. Montrer que P est effectivement une permutation et que le biais ϵ_P est nul.

Solution

- Soit P une permutation aléatoire telle que $\epsilon_P = 2k/n$. L'entier $n/4 + k$ est alors le nombre de valeurs paires $x \in \{0, \dots, n-1\}$ telles que $P(x)$ est pair. Il suffit donc d'énumérer les permutations P pour une valeur $n/4 + k$ donnée.

Une telle permutation est déterminée par quatre ensembles

$$A_{b_1, b_2} = \{x \in \{0, \dots, n-1\}, x \equiv b_1 \pmod{2} \wedge P(x) \equiv b_2 \pmod{2}\}$$

pour $(b_1, b_2) \in \{0, 1\}^2$. Nous avons $\#A(0, 0) = n/4 + k$, $\#A(0, 1) = n/2 - (n/4 + k) = n/4 - k$. Puisque P est une permutation, nous avons $\#A(1, 1) = n/2 - \#A(0, 1) = n/4 + k$ et enfin $\#A(1, 0) = n/4 - k$. Pour construire une permutation P telle que $\#A(0, 0) = n/4 + k$, il faut donc choisir :

- les $(n/4 + k)$ éléments de $A(0, 0)$ parmi les $n/2$ valeurs paires possibles ;
- les $(n/4 + k)$ images de ces éléments par P parmi les $n/2$ valeurs paires possibles ;
- les $(n/4 + k)$ éléments de $A(1, 1)$ parmi les $n/2$ valeurs impaires possibles ;
- les $(n/4 + k)$ images de ces éléments par P parmi les $n/2$ valeurs impaires possibles.

Une fois ces valeurs sélectionnées, il faut déterminer les deux bijections qui envoient $A(0, 0)$ et $A(1, 1)$ sur leurs images et les deux bijections qui envoient $A(1, 0)$ et $A(0, 1)$ sur leurs images. Les deux premiers ensembles sont de cardinaux $n/4 + k$ et les deux derniers de cardinaux $n/4 - k$. Nous obtenons donc le nombre de permutations

$$\binom{n/2}{n/4+k}^4 [(n/4+k)!]^2 [(n/4-k)!]^2$$

- L'entier 257 est un nombre premier, donc l'application $a \mapsto 45^a \pmod{257}$ est une bijection de $\{0, \dots, 255\}$ dans $\{1, \dots, 256\}$ si et seulement si 45 est un générateur de \mathbb{Z}_{257}^* . L'ordre de 45 est un diviseur de $\varphi(257) = 256$ et en calculant $45^{128} \pmod{256}$ nous obtenons la valeur $256 \equiv -1 \pmod{257}$, donc 45 est bien un générateur de \mathbb{Z}_{257}^* et P est une permutation de $\{0, \dots, 255\}$.

Pour tout $x \in \{0, \dots, 255\}$, nous avons $45^{x+128} \equiv (-1) \cdot 45^x \pmod{257}$ et comme 257 est impair nous obtenons que les parités de $P(x)$ et de $P(x+128)$ sont opposées pour tout $x \in \{0, \dots, 255\}$ alors que les parités de x et $x+128 \pmod{256}$ sont les mêmes. Avec les notations de la question précédente, nous avons donc $\#A_{0,0} = \#A_{0,1}$ et $\#A_{1,0} = \#A_{1,1}$.

Comme l'ensemble $A_{0,0} \cup A_{0,1}$ consiste en les valeurs paires de $x \in \{0, \dots, 255\}$, $\#(A_{0,0} \cup A_{0,1}) = 128$ (et de même $\#(A_{1,0} \cup A_{1,1}) = 128$). Nous avons donc $\#A_{b_1, b_2} = 64$ pour tout $(b_1, b_2) \in \{0, 1\}^2$ et

$$\#\{x \in \{0, \dots, 255\}, x \equiv P(x) \pmod{2}\} = \#(A_{0,0} \cup A_{1,1}) = 128$$

Nous obtenons donc bien $\epsilon_P = 0$.

Note

Le choix de l'exponentiation comme permutation P rend donc le schéma SAFER résistant à la cryptanalyse linéaire mais un autre choix aurait permis d'attaquer le système plus rapidement que la recherche exhaustive. En particulier, pour $n = 256$, une permutation aléatoire vérifie $\epsilon \geq 2^{-4}$ avec probabilité supérieure à 0,061. Ces résultats ont été présentés par S. VAUDENAY à la conférence *Fast Software Encryption 1994* [65].

3.7 ATTAQUES PAR SATURATION

Une attaque par saturation (*saturation attack*, en anglais) ou *attaque par cryptanalyse intégrale (integral cryptanalysis*, en anglais) étudie le comportement d'un cryptosystème lorsque l'on chiffre une famille de messages choisie pour que certaines composantes du système prennent toutes les valeurs possibles.

L'attaquant utilise ensuite certaines propriétés induites par la fonction de tour pour créer un distinguateur. Les propriétés généralement utilisées sont, qu'après un certain nombre de tours, certains octets du chiffré obtenu peuvent être vus comme une bijection de certains octets du clair (ou comme une somme de bijections de certains octets du clair). La technique a tout d'abord été utilisée par L. KNUDSEN contre le chiffrement par bloc Square [16], un ancêtre de Rjindael, puis formalisée par D. WAGNER et L. KNUDSEN pour attaquer des chiffrements insensibles à la cryptanalyse différentielle.

Dans l'exercice suivant, nous considérons des attaques par saturation (à textes clairs choisis) contre les variantes de l'AES réduites à 4,5 ou 6 tours. Pour ces variantes de l'AES, nous considérons qu'il n'y a pas d'opération MixColumns dans le dernier tour (comme c'est le cas pour le chiffrement AES complet).

Problème 3.18 Attaque par saturation contre l'AES

1. Nous considérons une attaque à clairs choisis contre une variante de l'AES réduite à 4 tours. L'attaquant demande le chiffrement de 256 clairs où quinze octets prennent une valeur constante quelconque et le seizième octet prend les 256 valeurs possibles. Montrer que la sortie du troisième tour est *équilibrée* (i.e. la somme des 16 octets vaut 0).
2. En déduire un moyen de retrouver la clé du quatrième tour par une attaque à clairs choisis.
3. Étendre l'attaque précédente en ajoutant un tour final.
4. Étendre l'attaque précédente en ajoutant un tour initial. Donner la complexité de l'attaque globale sur une variante de l'AES réduite à 6 tours.

Solution

1. L'attaquant demande le chiffrement de 256 clairs où quinze octets prennent une valeur constante quelconque (pas nécessairement toujours la même) et le seizième octet prend les 256 valeurs possibles. La figure (3.12) représente l'évolution des 16 octets de l'état d'AES pour chaque opération (où l'on a choisi que le premier octet est celui qui prend les 256 valeurs possibles).

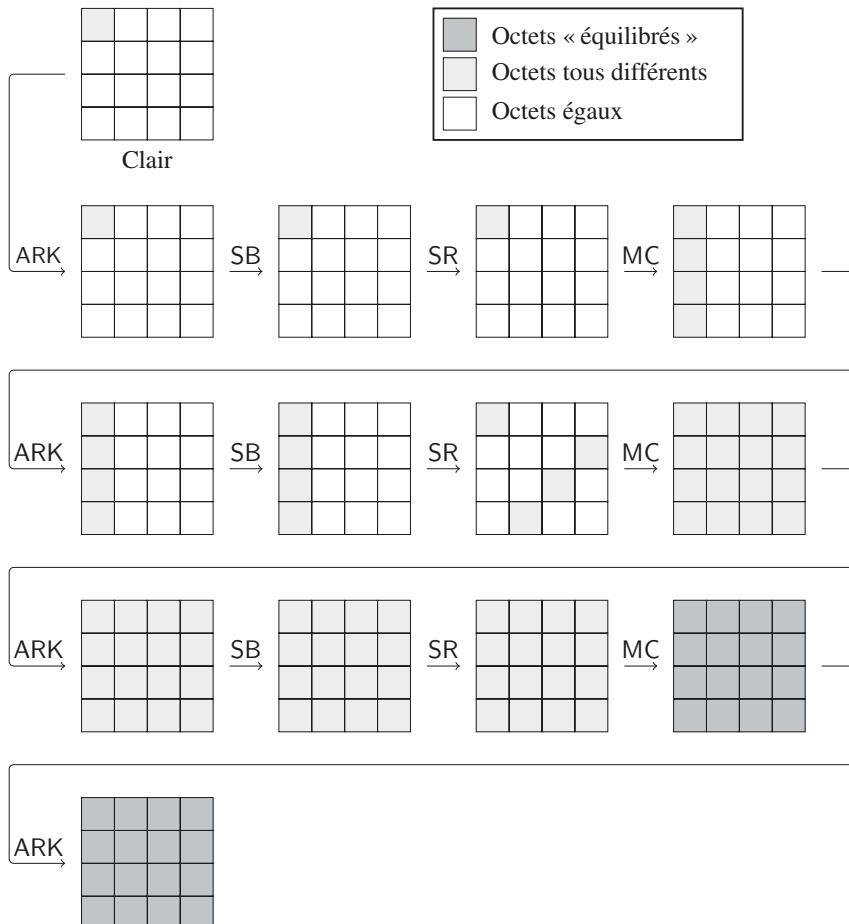


Figure 3.12 - Attaque par saturation sur l'AES

- Le « ou exclusif » avec la clé K_0 (AddRoundKey, ARK) produit 256 blocs où quinze octets sont identiques et le seizième prend toutes les valeurs possibles.
- L'application successive de l'opération SubBytes (SB) et du décalage de ligne ShiftRows (SR) produit également 256 blocs où 15 octets sont identiques et le seizième prend toutes les 256 valeurs possibles (en fonction de la position initiale de l'octet prenant toutes les valeurs, la position peut changer suite à l'application de ShiftRows).

- L’application du mélange de colonnes **MixColumns** (MC) à ces 256 blocs donne 256 nouveaux blocs pour lesquels trois colonnes ont des valeurs identiques alors que les quatre octets d’une quatrième colonne prennent toutes les valeurs possibles (mais pas nécessairement les mêmes). Cette propriété vient des propriétés de la fonction **MixColumns** vues dans l’exercice (2.16).
- Le « ou exclusif » avec la clé du premier tour et l’application de la procédure **SubBytes** fournissent ensuite 256 blocs du même type.
- Le décalage de lignes **ShiftRows** suivant produit 256 blocs où chaque colonne et chaque ligne comportent trois positions qui contiennent des octets constants et une position contient toutes les valeurs possibles.
- L’application de **MixColumns** donne 256 blocs où les seize octets prennent individuellement toutes les valeurs possibles.
- Le « ou exclusif » avec la clé du deuxième tour, l’application des boîtes *S* et le décalage de ligne qui suivent fournissent enfin 256 blocs du même type.

Les 256 blocs en entrée de l’opération **MixColumns** du troisième tour sont tels que leurs seize octets prennent toutes les valeurs possibles. La somme des 256 valeurs de chacun des seize octets vaut donc 0 modulo 256. L’opération **MixColumns** étant linéaire, cette propriété est vérifiée également pour la sortie de cette opération et la sortie du troisième tour est donc équilibrée.

2. L’opération **AddRoundKey** du troisième tour fournit également 256 blocs équilibrés. Ces blocs subissent l’action des deux opérations **SubBytes** et **ShiftRows**, puis de l’opération **AddRoundKey** du quatrième tour (rappelons qu’il n’y a pas d’opération **MixColumns** dans le dernier tour de l’AES).

Un attaquant disposant du chiffré AES réduit à quatre tours de 256 clairs où quinze octets ne sont pas modifiés et le seizième prend toutes les valeurs possibles peut retrouver les seize octets de clé l’un après l’autre. Pour cela, il lui suffit d’effectuer les opérations **AddRoundKey**⁻¹, pour chaque octet de clé testé, puis d’inverser les opérations **SubBytes**⁻¹ et **ShiftRows**⁻¹ et de tester si l’octet correspondant est équilibré. Le nombre de clés vérifiant ce test sera en moyenne égal à 1 (et si plusieurs clés sont obtenues, l’attaquant peut les filtrer en utilisant une autre famille de clairs choisis).

3. Pour étendre l’attaque lorsqu’un cinquième tour est ajouté à la fin, il suffit pour un attaquant de deviner une clé partielle de 4 octets (*i.e.* une colonne) de la clé du cinquième tour. Pour les 2^{32} sous-clés possibles, l’attaquant inverse les opérations **AddRoundKey**, **SubBytes** et de **ShiftRows** du cinquième tour (en supposant toujours qu’il n’y a pas d’opération **MixColumns** dans le dernier tour de l’AES). Si la clé partielle a été bien devinée, l’attaquant se retrouve dans la même position que dans la question précédente sauf qu’il y a une opération **MixColumns** dans le quatrième tour. Il n’a cependant pas à effectuer une recherche exhaustive de la clé de nouveau sur toute la colonne (ce qui donnerait au final une attaque en 2^{64}) car les opérations **AddRoundKey** et **MixColumns** commutent.

Il peut donc appliquer d'abord le MixColumns et deviner ensuite un seul octet de la clé comme dans la question précédente. Pour chaque colonne, l'attaquant demande de tester 2^{40} clés possibles (sur seulement deux tours d'AES) avec 256 clairs choisis.

- Pour étendre l'attaque lorsqu'un tour initial est ajouté, l'attaquant doit réaliser une attaque à clairs choisis assurant que pour tous les choix de la clé du premier tour, il dispose après le MixColumns du premier tour de 256 blocs où quinze octets sont identiques et le seizième prend toutes les valeurs possibles.

L'entrée de l'opération MixColumns du premier tour doit donc vérifier qu'une combinaison linéaire spécifique de ces octets prend les 256 valeurs possibles tandis que trois autres combinaisons linéaires sont constantes. Cette propriété impose des conditions sur les 4 octets à l'entrée de l'opération ShiftRows et sur les 4 octets à l'entrée de l'opération SubBytes du premier tour. Si les octets correspondants de la clé du premier tour sont devinés par l'attaquant, ces conditions peuvent être converties en des conditions sur quatre octets du texte clair.

Il suffit donc à l'attaquant de demander le chiffrement de 2^{32} clairs prenant toutes les valeurs possibles sur ces quatre octets et une valeur constante pour les 12 autres octets. Ensuite, en devinant les quatre octets de clés correspondants pour le premier tour, l'attaquant peut reproduire l'attaque vue dans les questions précédentes pour les 5 tours restants. L'attaque demande donc le chiffrement de 2^{32} clairs choisis, et a une complexité de $2^{32} \cdot 2^{40} = 2^{72}$ évaluations de trois tours de l'AES.

Ladder-DES est un système de chiffrement par bloc qui a été proposé par T. RITTER en 1994. Il s'agit d'un schéma de Feistel à 4 tours avec une taille de blocs de 128 bits où la fonction de tour est la permutation DES. Il n'y a pas d'algorithme de diversification de clé et une clé indépendante K_i de 56 bits est utilisée pour le i -ème tour de Ladder-DES (pour $i \in \{1, 2, 3, 4\}$).

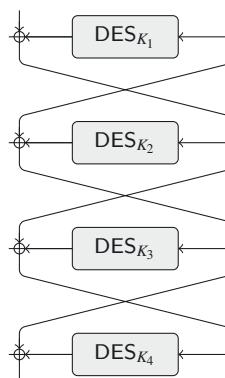


Figure 3.13 - Description du chiffrement par bloc *Ladder-DES*

Exercice 3.19 Attaque par distinguateur sur Ladder-DES

1. Considérons t messages de la forme $(L_{0,i}, R_0)$ où R_0 est constant et les éléments $L_{0,i}$ sont deux à deux distincts pour $i \in \{1, \dots, t\}$ et notons $(L_{3,i}, R_{3,i})$ pour $i \in \{1, \dots, t\}$ les messages obtenus à la sortie du troisième tour de Ladder-DES. Montrer que les valeurs $L_{3,i}$ pour $i \in \{1, \dots, t\}$ sont deux à deux distinctes.
2. Estimer la probabilité pour une permutation aléatoire $F : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ qu' étant données t entrées distinctes m_1, \dots, m_t de $\{0, 1\}^{2n}$, les valeurs tronquées des $F(m_1), \dots, F(m_t)$ à n bits soient toutes différentes.
3. En déduire une attaque à chiffrés choisis qui permet de déterminer la clé K_4 utilisée lors du quatrième tour.
4. En déduire une attaque sur le chiffrement Ladder-DES.

Solution

1. Notons $L_{i,j}$ et $R_{i,j}$ les messages obtenus à la sortie du j -ième tour de Ladder-DES pour $j \in \{1, 2, 3\}$ (après l'échange des parties droite et gauche du message). À la sortie du premier tour de Ladder-DES, nous avons

$$L_{1,i} = R_0 \text{ et } R_{1,i} = L_{0,i} \oplus \text{DES}_{K_1}(R_0)$$

Les valeurs $R_{1,i}$ sont donc deux à deux distinctes pour $i \in \{1, \dots, t\}$. À la sortie du second tour, nous avons

$$R_{2,i} = L_{1,i} \oplus \text{DES}_{K_2}(R_{1,i}) = R_0 \oplus \text{DES}_{K_2}(R_{1,i})$$

Comme DES est une permutation, les valeurs $\text{DES}_{K_2}(R_{1,i})$ pour $i \in \{1, \dots, t\}$ sont deux à deux distinctes et il en est de même des $R_{2,i}$. Puisqu'à la sortie du troisième tour, nous avons $L_{3,i} = R_{2,i}$ pour $i \in \{1, \dots, t\}$, nous avons bien montré le résultat demandé.

2. Les valeurs m_i pour $i \in \{1, \dots, t\}$ étant deux à deux distinctes et F étant une permutation aléatoire, les valeurs $F(m_i)$ pour $i \in \{1, \dots, t\}$ sont distinctes et uniformément distribuées dans $\{0, 1\}^{2n}$. Le nombre de tels t -uplets est égal à

$$2^{2n}(2^{2n}-1)(2^{2n}-2)\cdots(2^{2n}-(t-1)) = \prod_{i=0}^{t-1} (2^{2n}-i)$$

Le nombre de t -uplets pour lesquels les valeurs tronquées des $F(m_1), \dots, F(m_t)$ à n positions fixées soient toutes différentes est égal à

$$(2^n \cdot 2^n)((2^n - 1) \cdot 2^n)((2^n - 2) \cdot 2^n)\cdots((2^n - (t-1)) \cdot 2^n) = 2^{nt} \prod_{i=0}^{t-1} (2^n - i)$$

Nous obtenons donc la probabilité

$$p_{n,t} = \frac{2^{nt} \prod_{i=0}^{t-1} (2^n - i)}{\prod_{i=0}^{t-1} (2^{2n} - i)} = \frac{\prod_{i=0}^{t-1} (1 - i \cdot 2^{-n})}{\prod_{i=0}^{t-1} (1 - i \cdot 2^{-2n})} \simeq \frac{\exp(-t(t-1)/2^{n+1})}{\exp(-t(t-1)/2^{2n+1})}$$

3. L'attaquant demande le chiffrement de t messages de la forme $(L_{0,i}, R_0)$ où R_0 est constant et $L_{0,i}$ pour $i \in \{1, \dots, t\}$ sont deux à deux distincts. Il effectue ensuite une recherche exhaustive pour déterminer la valeur K_4 en effectuant un déchiffrement partiel du dernier tour pour les 2^{56} valeurs possibles de K_4 . Si la clé K_4 est la bonne, tous les déchiffrés partiels seront différents. Si la clé est incorrecte, en supposant que la permutation résultante se comporte comme une permutation aléatoire, les déchiffrés seront tous différents seulement avec probabilité $1 - p_{n,t}$. Pour éviter de devoir filtrer trop de mauvaises clés, il faut choisir une valeur de t telle que $p_{64,t}$ est inférieure à 2^{56} .

En choisissant t de l'ordre de 2^{36} , nous obtenons, d'après la question précédente,

$$p_{64,2^{36}} \simeq \exp(-2^{72}/2^{65})/\exp(-2^{72}/2^{129}) = \exp(-2^7)/\exp(-2^{-57}) \leq 2^{-184}$$

et le nombre de fausses clés trouvées par l'attaque sera en moyenne égal à $2^{56} \cdot 2^{-184} = 2^{-128} \simeq 0$.

4. La clé K_3 peut être trouvée par recherche exhaustive de façon identique et avec le même ensemble de textes clairs choisis. En effet, comme nous l'avons vu dans la question 1, les valeurs $R_{2,i}$ pour $i \in \{1, \dots, t\}$ sont deux à deux distinctes. Les clés K_1 et K_2 peuvent donc être obtenues également par recherche exhaustive en utilisant les distingueurs connus sur les schémas de Feistel à un et deux tours (*cf.* Exercice (2.4)).

Nous obtenons donc que

- l'attaque pour retrouver la clé K_4 demande le déchiffrement partiel du dernier tour pour 2^{36} chiffrés pour chaque clé ;
- l'attaque pour retrouver la clé K_3 demande le déchiffrement partiel du troisième tour de deux chiffrés pour chaque clé ;
- l'attaque pour retrouver la clé K_i avec $i \in \{1, 2\}$ demande le déchiffrement partiel du tour correspondant d'un seul chiffré pour chaque clé choisie ;

Le coût du déchiffrement d'un tour de Ladder-DES étant égal au quart du coût total d'un chiffrement Ladder-DES complet, nous obtenons un coût total équivalent à $1/4 \cdot 2^{56} \cdot (2^{36} + 2 + 1 + 1) \simeq 2^{90}$ chiffrements complets Ladder-DES.

Note

L'attaque précédente est due à E. BIHAM et a été présentée à la conférence *Fast Software Encryption 1997* [6].

CHIFFREMENT PAR FLOT

Le *chiffrement par flot* (*stream cipher*) également appelé *chiffrement par flux* ou *chiffrement à la volée*, est, après le chiffrement par bloc, l'autre grande famille de systèmes de chiffrement symétrique. Par opposition à ce dernier, un chiffrement par flot permet de traiter des données de longueur arbitraire sans avoir à les découper.

Dans un chiffrement par flot dit *synchrone*¹, le texte chiffré est obtenu en combinant par une opération de groupe (généralement le « ou exclusif » bit-à-bit) le message clair avec une suite secrète, de même longueur que le message et appelée *suite chiffrante*. Les algorithmes de chiffrement par flot reposent donc sur le même principe que le chiffrement de Vernam mais utilisent une suite *pseudo-aléatoire*, produite à partir d'une clé secrète par un *générateur pseudo-aléatoire* (au lieu d'un masque parfaitement aléatoire).

Chaque nouveau symbole transmis peut donc être chiffré ou déchiffré sans qu'il soit nécessaire d'attendre les bits suivants. De plus, les algorithmes de chiffrement par flot sont généralement plus rapides que ceux chiffrant par bloc. Leur utilisation est donc privilégiée dans les applications qui imposent de fortes contraintes sur la taille et la consommation électrique du circuit électronique dédié au chiffrement (comme dans les téléphones mobiles).

Nous étudierons les *registres à décalage à rétroaction linéaire* qui sont des dispositifs rapides (notamment en implantation matérielle) pour engendrer des suites ayant de bonnes qualités statistiques. Ces registres sont très souvent utilisés comme brique de base pour la construction de systèmes de chiffrement à flot au sein d'un dispositif plus complexe. Nous analyserons notamment la sécurité du générateur de Geffe (un exemple de *registre à rétroaction linéaire combinés*), du chiffrement par flot Toyocrypt (un exemple de *registre à rétroaction linéaire filtré*) et de certains *registres à décalage irrégulier*. Enfin, nous examinerons la sécurité des deux systèmes de chiffrement à flot A5/1 et RC4.

4.1 REGISTRES À DÉCALAGE À RÉTROACTION LINÉAIRE

Un *registre à décalage à rétroaction* (ou FSR de l'anglais *Feedback Shift Register*) est un dispositif permettant d'engendrer une suite chiffrante vérifiant une relation de

1. Il existe d'autres types de chiffrement par flot, notamment les systèmes *auto-synchronisants*, mais ils ne seront pas étudiés dans cet ouvrage.

récurrence. Un tel registre de longueur ℓ sur \mathbb{F}_2^n est formé d'un registre de ℓ cellules contenant chacune n bits (les $\ell \cdot n$ bits forment l'état interne du FSR). Ces cellules sont initialisées avec ℓ mots de n bits et à chaque unité de temps (réalisée par une horloge interne) chaque mot de n bits est décalé d'une cellule vers la droite. Le contenu de la cellule la plus à droite sort du registre et la cellule la plus à gauche reçoit en entrée un mot obtenu par combinaison du contenu du registre. Si la valeur de ce *mot de rétroaction* est obtenue par une combinaison linéaire des mots du registre, nous parlons de *registre à décalage à rétroaction linéaire* (ou LFSR de l'anglais *Linear Feedback Shift Register*). Un tel registre est représenté sur la figure 4.1.

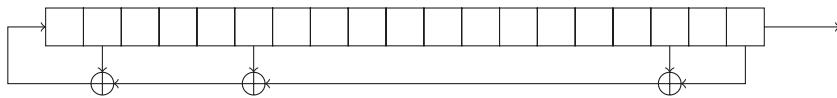


Figure 4.1- Exemple de registre à décalage à rétroaction linéaire

Dans la suite, nous étudierons des registres à décalage à rétroaction linéaire binaire (*i.e.* formés de cellules contenant $n = 1$ bit). Une suite chiffrante produite par un registre à décalage à rétroaction linéaire de longueur ℓ sur \mathbb{F}_2 est donc entièrement déterminée par le contenu initial du registre $(s_1, \dots, s_\ell) \in \mathbb{F}_2^\ell$ et les coefficients de rétroaction linéaire $(a_1, \dots, a_\ell) \in \mathbb{F}_2^\ell$ par la relation

$$s_{m+1} = a_1 s_{m-\ell+1} \oplus a_2 s_{m-\ell} \cdots \oplus a_\ell s_m$$

pour tout entier $m \geq \ell$. L'action du LFSR peut également être représentée sous la forme matricielle (pour $m \geq \ell$) :

$$\begin{bmatrix} s_{m-\ell+1} \\ s_{m-\ell+2} \\ \vdots \\ \vdots \\ s_{m-2} \\ s_{m-1} \\ s_m \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & & & \ddots & & \vdots & \vdots \\ \vdots & & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ a_1 & a_2 & a_3 & \dots & a_{\ell-2} & a_{\ell-1} & a_\ell \end{bmatrix} \cdot \begin{bmatrix} s_{m-\ell} \\ s_{m-\ell+1} \\ \vdots \\ \vdots \\ s_{m-3} \\ s_{m-2} \\ s_{m-1} \end{bmatrix}$$

Une telle suite est ultimement périodique de période inférieure à $2^\ell - 1$ (*i.e.* il existe deux entiers T et n_0 tel que $s_{i+T} = s_i$ pour tout entier $i \geq n_0$ avec $T \leq 2^\ell - 1$). Si de plus $a_\ell = 1$, la suite est périodique (*i.e.* $s_{i+T} = s_i$ pour tout entier $i \geq 0$). Le polynôme

$$f(X) = 1 + \sum_{i=1}^{\ell} a_i X^i \in \mathbb{F}_2[X]$$

est appelé *polynôme de rétroaction* du LFSR. Étant donnée une suite binaire ultimement périodique $s = (s_n)_{n \geq 0} \in \mathbb{F}_2^{\mathbb{N}}$, le *polynôme de rétroaction minimal* de s est le polynôme de plus bas degré parmi les polynômes de rétroaction de tous les LFSR possibles qui génèrent la suite $(s_n)_{n \geq 0}$. La *complexité linéaire* d'une suite ultimement périodique est le degré de son polynôme de rétroaction minimal.

Il est souvent utile d'associer à une suite $(s_n)_{n \geq 0} \in \mathbb{F}_2^{\mathbb{N}}$ la série formelle

$$s(X) = \sum_{n \geq 0} s_n X^n \in \mathbb{F}_2[[X]]$$

Exercice 4.1 LFSR et polynômes de rétroaction

- Soit $(s_n)_{n \geq 0}$ une suite engendrée par un LFSR avec un polynôme de rétroaction de degré ℓ

$$f(X) = 1 + \sum_{i=1}^{\ell} a_i X^i \in \mathbb{F}_2[X]$$

Montrer qu'il existe un polynôme $g(X) \in \mathbb{F}_2[X]$ avec $\deg g < \deg f$ tel que $s(X) = g(X)/f(X)$.

- Montrer que le polynôme de rétroaction minimal de $(s_n)_{n \geq 0}$ est l'unique polynôme $f_0(X) \in \mathbb{F}_2[X]$ tel qu'il existe $g_0 \in \mathbb{F}_2[X]$ avec $\deg(g_0) < \deg(f_0)$, $\text{pgcd}(g_0, f_0) = 1$ et $s(X) = g_0(X)/f_0(X)$.
- Montrer que si le polynôme de rétroaction minimal de $(s_n)_{n \geq 0}$ de degré ℓ est primitif (*i.e.* si l'une de ses racines engendre le groupe multiplicatif de $(\mathbb{F}_{2^\ell})^*$) alors la suite est de période maximale $2^\ell - 1$.

Solution

- Si un tel polynôme g existe, nous avons $g(X) = s(X) \cdot f(X)$, soit

$$g(X) = \left(\sum_{n \geq 0} s_n X^n \right) \cdot \left(1 + \sum_{i=1}^{\ell} a_i X^i \right)$$

En identifiant les coefficients, nous avons

$$g(X) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^i a_{i-j} s_j \right) X^i$$

avec $a_0 = 1$ et $a_i = 0$ pour $i \geq \ell + 1$. La relation de récurrence vérifiée par la suite $(s_n)_{n \geq 0}$ montre que pour $i \geq \ell$

$$\begin{aligned} \sum_{j=0}^i a_{i-j} s_j &= a_0 \cdot s_i \oplus a_1 \cdot s_{i-1} \oplus \cdots \oplus a_i \cdot s_0 \\ &= s_i \oplus a_1 \cdot s_{i-1} \oplus \cdots \oplus a_\ell \cdot s_{i-\ell} \\ &= 0 \end{aligned}$$

et nous obtenons le polynôme

$$g(X) = \sum_{i=0}^{\ell-1} \left(\sum_{j=0}^i a_{i-j} s_j \right) X^i$$

qui convient effectivement.

2. Soit f_0 le polynôme de rétroaction minimal de $(s_n)_{n \geq 0}$. D'après la question précédente, il existe un polynôme $g_0(X) \in \mathbb{F}_2[X]$ avec $\deg g_0 < \deg f_0$ tel que $s(X) = g_0(X)/f_0(X)$. Nous avons nécessairement $\text{pgcd}(g_0, f_0) = 1$ (sinon le polynôme $f_0/\text{pgcd}(g_0, f_0)$ est un polynôme de rétroaction de la suite $(s_n)_{n \geq 0}$ de degré inférieur à $\deg f_0$). Réciproquement, comme l'ensemble des polynômes de rétroaction est un idéal de l'anneau principal $\mathbb{F}_2[X]$, le générateur de cet idéal vérifie bien les propriétés demandées.
3. Soit f le polynôme de rétroaction minimal de la suite $(s_n)_{n \geq 0}$ supposé de degré ℓ . Le polynôme caractéristique de la matrice suivante est le polynôme réciproque de f : $\tilde{f}(X) = X^\ell f(X^{-1})$.

$$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & & \ddots & & & & \vdots \\ \vdots & & & \ddots & & & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ a_1 & a_2 & a_3 & \dots & a_{\ell-2} & a_{\ell-1} & a_\ell \end{bmatrix}$$

Les valeurs propres de cette matrice, appelée *matrice compagnon* de \tilde{f} , sont les racines de \tilde{f} . Elles sont toutes distinctes et d'ordre $2^\ell - 1$ puisque f est primitif. La matrice est donc d'ordre $2^\ell - 1$ et l'ordre de la matrice est la période de la suite.

L'exercice suivant montre qu'une suite produite par un LFSR a de bonnes propriétés statistiques.

Exercice 4.2 Propriétés statistiques d'une suite produite par un LFSR

Montrer que si la plus petite période d'un registre à décalage à rétroaction linéaire de longueur $\ell \geq 1$ sur \mathbb{F}_2 est de longueur maximale $2^\ell - 1$, alors dans toute suite finie d'éléments de la suite de longueur $2^\ell - 1$, chaque suite finie d'éléments de \mathbb{F}_2 de longueur k , pour $k \in \{1, \dots, \ell\}$ apparaît entre $2^{\ell-k}$ et $2^{\ell-k} + (\ell - k)$ fois si elle est non nulle et $2^{\ell-k} - 1$ et $2^{\ell-k} - 1 + (\ell - k)$ fois si elle est nulle.

Solution

Remarquons tout d'abord que dans une suite de période maximale $2^\ell - 1$, toute suite de ℓ éléments (non tous nuls) apparaît une et une seule fois par période. En effet, il y a $2^\ell - 1$ valeurs pour les registres par période et ces registres prennent au plus $2^\ell - 1$ valeurs. Ils prennent donc nécessairement toutes les valeurs une fois.

Comme tous les contenus possibles des registres de $\mathbb{F}_2^\ell \setminus \{(0, \dots, 0)\}$ apparaissent une et une seule fois, chaque suite de longueur k pour $k \in \{1, \dots, \ell\}$ apparaît exactement $2^{\ell-k}$ fois comme préfixe d'un des $2^\ell - 1$ registres différents s'il est non nul et $2^{\ell-k} - 1$ s'il est nul. Dans une suite de longueur $2^\ell - 1$ quelconque, le décalage pour atteindre un tel préfixe est au plus de $\ell - k$ et chaque suite finie d'éléments de \mathbb{F}_2 de longueur k , apparaît donc entre $2^{\ell-k}$ et $2^{\ell-k} + (\ell - k)$ fois si elle est non nulle et $2^{\ell-k} - 1$ et $2^{\ell-k} - 1 + (\ell - k)$ fois si elle est nulle.

Exercice 4.3 Reconstruction du polynôme de rétroaction minimal

- Soit $(s_n)_{n \geq 0}$ une suite ultimement périodique de \mathbb{F}_2 de complexité linéaire ℓ . Montrer que si l'on connaît 2ℓ termes consécutifs de $(s_n)_{n \geq 0}$, il est possible de calculer les coefficients du polynôme de rétroaction minimal de $(s_n)_{n \geq 0}$.
- En déduire un algorithme qui, étant donnée une suite binaire ultimement périodique $(s_n)_{n \geq 0} \in \mathbb{F}_2^{\mathbb{N}}$ de complexité linéaire inférieure à ℓ , retourne cette complexité linéaire et le polynôme de rétroaction linéaire de la suite en temps $O(\ell^4)$.

Solution

- Notons

$$f(X) = 1 + \sum_{i=1}^{\ell} a_i X^i \in \mathbb{F}_2[X]$$

le polynôme de rétroaction minimal de la suite $(s_n)_{n \geq 0}$. Nous avons la relation matricielle

$$\begin{bmatrix} s_i & s_{i+1} & s_{i+2} & \dots & s_{i+\ell-1} \\ s_{i+1} & s_{i+2} & s_{i+3} & \dots & s_{i+\ell} \\ \vdots & & & \ddots & \\ s_{i+\ell-2} & s_{i+\ell-1} & s_{i+\ell} & \dots & s_{i+2\ell-3} \\ s_{i+\ell-1} & s_{i+\ell} & s_{i+\ell+1} & \dots & s_{i+2\ell-2} \end{bmatrix} \cdot \begin{bmatrix} a_\ell \\ a_{\ell-1} \\ \vdots \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} s_{i+\ell} \\ s_{i+\ell+1} \\ \vdots \\ s_{i+2\ell-2} \\ s_{i+2\ell-1} \end{bmatrix}$$

pour tout $i \geq 0$. Il suffit donc de montrer que la matrice est inversible. Les colonnes de cette matrice sont les contenus du registre à décalage aux pas d'horloge $i, i+1, \dots, i+\ell-1$. Une combinaison linéaire nulle non triviale des colonnes entraînerait l'existence d'un polynôme non nul de degré inférieur à ℓ qui serait un polynôme de rétroaction de la suite. La matrice est donc inversible et les méthodes classiques d'algèbre linéaire (comme le pivot de Gauss) permettent de retrouver les coefficients de f .

- Il suffit d'exécuter l'algorithme précédent pour toutes les valeurs $i \in \{1, \dots, \ell\}$, en partant de la valeur $i = 1$. Chaque résolution de système linéaire à partir des termes s_0, \dots, s_{2i-1} de la suite donne un polynôme de rétroaction candidat et l'on peut tester s'il est correct avec les termes suivants de la suite. Chaque étape a un coût cubique en la longueur i de la suite traitée et le coût total est donc de l'ordre de $O(\ell^4)$.

Note

L'algorithme de Berlekamp-Massey [42] est un algorithme qui permet de déterminer la complexité linéaire d'une suite finie ultimement périodique, ainsi que le polynôme de rétroaction minimal qui permet de l'engendrer. Cet algorithme est quadratique en la longueur de la suite.

Même lorsque le polynôme de rétroaction d'un LFSR est primitif, la complexité linéaire de la suite produite est trop faible pour permettre une utilisation cryptographique de la suite produite. Pour cette raison, il est nécessaire de proposer des moyens d'augmenter la complexité d'une suite produite par un registre à décalage. Les trois méthodes principales qui ont été proposées sont :

- les *registres à rétroaction linéaire combinés* qui appliquent une fonction non linéaire aux bits de sortie de deux LFSR ou plus (cf. Figure 4.4)
- les *registres à rétroaction linéaire filtrés* qui appliquent une fonction non linéaire à certains bits de l'état interne d'un LFSR (cf. Figure 4.3)
- les *registres à décalage irrégulier* où l'horloge interne du LFSR est irrégulière et contrôlée par un autre registre à décalage.

4.2 CHIFFREMENT PAR FLOT PAR REGISTRES À DÉCALAGE IRRÉGULIER

Le générateur à signal d'arrêt (*stop-and-go generator* en anglais) est un exemple de registre à décalage irrégulier qui a été proposé par Th. BETH et F. PIPER en 1984 [5]. Il utilise la sortie d'un premier registre à décalage à rétroaction linéaire R_1 pour contrôler l'horloge d'un second registre à décalage à rétroaction linéaire R_2 . Le registre R_2 ne change d'état à l'instant t que si la sortie de R_1 est égale à 1 à l'instant $t-1$ (cf. Figure 4.2).

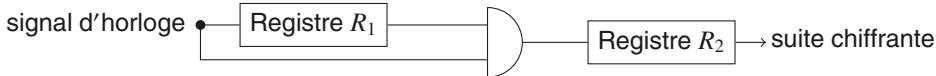


Figure 4.2 - Description du générateur à signal d'arrêt

Exercice 4.4 (avec programmation). Distinguier sur le générateur à signal d'arrêt

1. En supposant que les deux registres d'un générateur à signal d'arrêt produisent des séquences uniformément distribuées, calculer la probabilité que deux bits consécutifs produits par le générateur soient égaux.
2. En déduire un distinguier sur le générateur à signal d'arrêt
3. Un adversaire a intercepté le chiffré suivant :

$$c = 0000 \ 1111 \ 1101 \ 1010 \ 0111 \ 1101 \ 0101 \ 1001.$$

Il sait qu'il s'agit du chiffrement de l'un des trois textes suivants à l'aide d'un générateur à signal d'arrêt :

$$\begin{aligned} m_1 &= 0110 \ 1001 \ 0010 \ 1001 \ 0101 \ 1001 \ 0110 \ 0001 \\ m_2 &= 0110 \ 0010 \ 1111 \ 0100 \ 1010 \ 1011 \ 1111 \ 0101 \\ m_3 &= 1110 \ 1000 \ 0101 \ 1101 \ 1001 \ 1110 \ 1100 \ 0101. \end{aligned}$$

Donner le texte clair qui a été le plus probablement chiffré.

Solution

1. Si le bit à l'instant $t-1$ en sortie du registre R_1 est égal à 0, alors le registre R_2 n'est pas décalé et le bit de sortie à l'instant t est donc toujours égal au bit de sortie à l'instant $t-1$.

Si le bit à l'instant $t-1$ en sortie du registre R_1 est égal à 1, alors le registre R_2 est décalé et le bit de sortie à l'instant t est égal au bit de sortie à l'instant $t-1$ avec probabilité 1/2. Nous avons donc

$$\begin{aligned} \Pr[R_2[t] = R_2[t-1]] &= \Pr[R_2[t] = R_2[t-1]|R_1[t-1] = 0] \cdot \Pr[R_1[t-1] = 0] \\ &\quad + \Pr[R_2[t] = R_2[t-1]|R_1[t-1] = 1] \cdot \Pr[R_1[t-1] = 1] \\ &= 1 \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}. \end{aligned}$$

2. La probabilité que deux bits consécutifs soient égaux dans une suite aléatoire est égale à 1/2. Pour distinguer une suite aléatoire de longueur ℓ d'une suite produite par un générateur à signal d'arrêt de même longueur, il suffit de compter le nombre de changements de bits entre deux symboles consécutifs. Si ce nombre est proche de $\ell/2$,

alors la suite est probablement aléatoire, alors que s'il est proche de $\ell/4$ la suite est probablement générée par le générateur à signal d'arrêt.

Plus formellement, supposons que nous disposons d'une suite de longueur n avec t changements de bits entre deux symboles consécutifs et que le distingueur décide que

- la suite a été produite par un générateur à signal d'arrêt si $t \in [0, 3n/8[$;
- la suite est une suite aléatoire dans le cas contraire.

Si la suite a effectivement été produite par un générateur à signal d'arrêt nous avons (en appliquant les bornes de Chernoff) :

$$\Pr[t < 3n/8] = 1 - \Pr[t \geq 3n/8] \geq 1 - \Pr[|t - n/4| \leq n/8] \geq 1 - 2 \exp(-n/48)$$

Si la suite est une suite aléatoire, nous avons

$$\Pr[t < 3n/8] = \Pr[|t - n/4| < n/8] \leq \exp(-9n/128)$$

Si le distingueur dispose d'une suite suffisamment longue (par exemple, $n \geq 1024$), nous obtenons des probabilités très proches de 1 et de 0 (respectivement) et le distingueur a donc un avantage très proche de 1/2 (en appliquant le théorème de Bayes).

3. En calculant le « ou exclusif » bit à bit des messages clairs potentiels et du chiffré, nous obtenons

$$\begin{aligned} m_1 \oplus c &= 0110 \ 0110 \ 1111 \ 0011 \ 0010 \ 0100 \ 0011 \ 1000 \\ m_2 \oplus c &= 0110 \ 1101 \ 0010 \ 1110 \ 1101 \ 0110 \ 1010 \ 1100 \\ m_3 \oplus c &= 1110 \ 0111 \ 1000 \ 0111 \ 1110 \ 0011 \ 1001 \ 1100 \end{aligned}$$

Ces valeurs correspondent à la suite chiffrante générée par le générateur à signal d'arrêt. Le nombre de changements de bits entre deux symboles consécutifs de ces suites est égal à 14, 22 et 9 (respectivement). La valeur 9 est la plus proche de celle attendue d'après la question 1 (à savoir $32/4 = 8$). Le message qui a été le plus probablement chiffré est donc m_3 .

Un générateur par rétrécissement (*shrinking generator* en anglais) est un autre exemple de registre à décalage irrégulier utilisé pour le chiffrement par flot. Ce principe a été publié en 1993 par D. COPPERSMITH, H. KRAWCZYK et Y. MANSOUR [13] ; il repose sur l'utilisation de deux registres à décalage à rétroaction linéaire. Le premier, noté R_1 , produit des bits de sortie tandis que le second noté R_2 , contrôle quels bits de sortie apparaissent dans la suite chiffrante. Si le bit du registre R_2 est égal à 1 alors le bit produit par R_1 est ajouté à la suite chiffrante, alors qu'il est simplement mis de côté si le bit du registre R_2 est égal à 0.

Une variante intéressante est le générateur par auto-rétrécissement (*self-shrinking generator* en anglais). Ce système utilise un seul registre à décalage à rétroaction linéaire R et considère les bits de sortie de R deux par deux :

- si $R[2t] = 0$ et $R[2t + 1] = 0$, le générateur n'ajoute rien à la suite chiffrante ;
- si $R[2t] = 0$ et $R[2t + 1] = 1$, le générateur n'ajoute rien à la suite chiffrante ;
- si $R[2t] = 1$ et $R[2t + 1] = 0$, le générateur ajoute un 0 à la suite chiffrante ;
- si $R[2t] = 1$ et $R[2t + 1] = 1$, le générateur ajoute un 1 à la suite chiffrante.

Problème 4.5 Propriétés du générateur par auto-rétrécissement

Considérons un registre à décalage à rétroaction linéaire R de longueur ℓ sur \mathbb{F}_2 dont le polynôme de rétroaction est primitif et l'état initial n'est pas l'état nul. Soit $(s_n)_{n \geq 0}$ la suite produite par le générateur par auto-rétrécissement à partir de ce registre R .

1. Montrer que la suite $(s_n)_{n \geq 0}$ est périodique de période $2^{\ell-1}$ et qu'elle est équilibrée (*i.e.* qu'elle contient autant de 0 que de 1).
2. Montrer que la période de la suite $(s_n)_{n \geq 0}$ est minorée par $2^{\lfloor \ell/2 \rfloor}$
3. Montrer que la complexité linéaire de la suite $(s_n)_{n \geq 0}$ est minorée par $2^{\lfloor \ell/2 \rfloor - 1}$

Solution

1. Soit $\vec{a} = (a_0, a_1, \dots)$ une suite générée par un LFSR de longueur ℓ avec un polynôme de rétroaction primitif. La suite \vec{a} est donc périodique de période maximale $2^\ell - 1$. La suite auto-réduite est donc aussi périodique. Après $2(2^\ell - 1)$ sorties du LFSR, nous avons la suite des couples

$$(a_0, a_1), (a_2, a_3), \dots, (a_{2^\ell-2}, a_0), (a_1, a_2), \dots, (a_{2^{\ell-3}}, a_{2^\ell-2})$$

et le terme suivant est (a_0, a_1) . Dans ces $2(2^\ell - 1)$ bits de sortie, chaque couple (a_i, a_{i+1}) apparaît exactement une fois (pour $0 \leq i < 2^\ell - 1$). Dans chaque période du LFSR, chaque couple 01, 10, 11 apparaît exactement $2^{\ell-2}$ fois et le couple 00 apparaît $2^{\ell-2} - 1$ fois (*cf.* Exercice 4.2). Par conséquent $2^{\ell-1}$ est une période de la séquence réduite. De plus, les couples 10 et 11 ont le même nombre d'occurrences et la suite réduite est équilibrée.

2. Considérons le cas où ℓ est pair (le cas impair se traite de façon totalement similaire). Sur une période doublée de la suite initiale, tous les mots de longueur ℓ apparaissent au moins une fois. En particulier, les mots $(1, x_1, 1, x_2, \dots, 1, x_{\ell/2})$ apparaissent dans la suite avant réduction et les mots (x_1, \dots, x_ℓ) apparaissent dans la suite auto-réduite. La suite réduite contient donc au moins $2^{\ell/2}$ mots différents. Une suite de période p contient au plus p motifs différents pour chaque taille de fenêtre. La période de la suite auto-réduite est donc nécessairement supérieure à $2^{\ell/2}$.
3. D'après les questions précédentes, la période de la séquence auto-réduite divise $2^{\ell-1}$. Elle est donc égale à 2^λ pour un entier $\lambda \geq \lfloor \ell/2 \rfloor$. Soit $f(x)$ le polynôme de rétroaction minimal de la suite auto-réduite. Nous savons que $f(x)$ divise $x^{2^\lambda} - 1 = (x - 1)^{2^\lambda}$.

Le polynôme f est donc nécessairement de la forme $(x - 1)^L$ où L est la complexité linéaire de la suite réduite. Il faut montrer que $L > 2^{l-1}$.

Supposons par l'absurde que $L \leq 2^{l-1}$, alors le polynôme f divise $(x-1)^{2^{l-1}} = x^{2^{l-1}} - 1$. Dans ce cas, le polynôme $x^{2^{l-1}} - 1$ serait un polynôme de rétroaction de la suite réduite. La suite réduite vérifierait alors la relation $s_n = s_{n-2^{l-1}}$ ce qui contredit le fait que la période minimale est 2^l .

Note

Les meilleures attaques connues contre le générateur par auto-rétrécissement sont des attaques de type « deviner et déterminer » et ont toutes une complexité exponentielle en la longueur du LFSR utilisé (cf. l'article récent [20] et les références citées).

4.3 CHIFFREMENT PAR FLOT PAR REGISTRE FILTRÉ

Un registre filtré est un système de chiffrement par flot constitué d'un registre à décalage à rétroaction linéaire filtré par une fonction booléenne F appelée *fonction de filtrage* (comme décrit sur la figure 4.3). À chaque pas d'horloge, le registre est avancé d'un pas et certains bits de l'état interne du registre sont utilisés comme entrées de la fonction de filtrage qui produit ainsi un bit de la suite chiffrante.

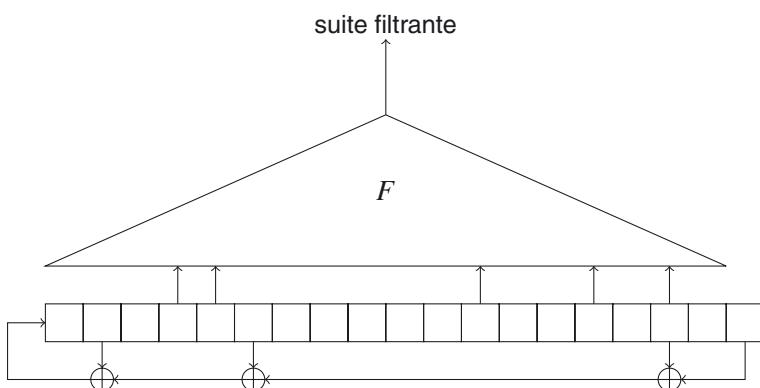


Figure 4.3 - Registre à rétroaction linéaire filtré

Dans cette section, nous allons étudier le système de chiffrement à flot par registre filtré appelé Toyocrypt qui a été proposé par K. SUGIMOTO en 2000 pour un processus de standardisation du gouvernement japonais appelé *Cryptrec*. Le système Toyocrypt utilise un unique registre à décalage à rétroaction linéaire de 128 bits et une fonction de filtrage non linéaire définie par

$$\begin{aligned}
 F(s_0, \dots, s_{127}) = & s_{127} + \sum_{i=0}^{62} s_i s_{\alpha_i} + s_{10} s_{23} s_{32} s_{42} \\
 & + s_1 s_2 s_9 s_{12} s_{18} s_{20} s_{23} s_{25} s_{26} s_{28} s_{33} s_{38} s_{41} s_{42} s_{51} s_{53} s_{59} + \prod_{i=0}^{62} s_i
 \end{aligned}$$

où $\{\alpha_0, \dots, \alpha_{62}\}$ est une permutation de l'ensemble $\{63, \dots, 125\}$. À chaque pas de l'horloge, un bit de suite chiffrante est alors produit en appliquant F au contenu de l'état interne (s_0, \dots, s_{127}) du registre. La clé de Toyocrypt est formée des 128 bits du contenu initial du registre et le polynôme de rétroaction utilisé (supposé primitif) est public.

Nous allons voir que le système Toyocrypt est vulnérable à plusieurs types d'attaques. La première de ces attaques est de type « deviner et déterminer ». Le principe est de deviner une partie des bits de l'état interne d'un système de chiffrement par flot et de les utiliser pour déterminer de façon certaine d'autres bits de l'état interne (à partir de la suite chiffrante et de ses relations avec l'état interne).

Exercice 4.6 Attaque « deviner et déterminer » sur Toyocrypt

- Montrer que si un attaquant connaît les bits s_0, \dots, s_{62} au pas d'horloge t et le bit associé de la suite chiffrante alors il obtient une relation linéaire liant les bits s_{63}, \dots, s_{127} au pas d'horloge t .
- En déduire une attaque de type « deviner et déterminer » de complexité équivalente à la résolution de 2^{96} systèmes linéaires 32×32 qui permet de retrouver l'état interne initial du registre à décalage (en utilisant 32 bits consécutifs de la suite chiffrante).

Solution

- En posant

$$\sigma = s_{10} s_{23} s_{32} s_{42} + s_1 s_2 s_9 s_{12} s_{18} s_{20} s_{23} s_{25} s_{26} s_{28} s_{33} s_{38} s_{41} s_{42} s_{51} s_{53} s_{59} + \prod_{i=0}^{62} s_i$$

qui ne dépend que des valeurs de s_0, \dots, s_{62} , le bit z de la suite chiffrante est donné par

$$z = s_{127} + \sum_{i=0}^{62} s_i \cdot s_{\alpha_i} + \sigma$$

qui est bien une relation linéaire (à coefficients connus) liant les bits s_{63}, \dots, s_{127} puisque $\{\alpha_0, \dots, \alpha_{62}\}$ est une permutation de l'ensemble $\{63, \dots, 125\}$.

2. Si un attaquant devine les valeurs des bits $s_0^{(0)}, \dots, s_{62}^{(0)}$ au pas d'horloge 0, le bit z_0 de la suite chiffrante lui donne une relation linéaire (à coefficients connus) liant les bits $s_{63}^{(0)}, \dots, s_{127}^{(0)}$. Au pas d'horloge suivant, s'il connaît les valeurs des bits $s_0^{(1)}, \dots, s_{62}^{(1)}$ et le bit z_1 de la suite chiffrante, il obtient une autre relation linéaire (à coefficients connus) liant les bits $s_{63}^{(1)}, \dots, s_{127}^{(1)}$. Puisque Toyocrypt utilise un LFSR, nous avons $s_{i+1}^{(1)} = s_i^{(0)}$ pour $i \in 0, \dots, 126$ et $s_0^{(1)}$ est une combinaison linéaire des valeurs $s_0^{(0)}, \dots, s_{127}^{(0)}$. En devinant, simplement la valeur de $s_0^{(1)}$, l'attaquant obtient une relation linéaire (à coefficients connus) liant les bits $s_{62}^{(0)}, \dots, s_{126}^{(0)}$.

Ainsi, en devinant les valeurs des bits $s_0^{(0)}, \dots, s_{62}^{(0)}$ de l'état initial et des bits $s_0^{(j)}$ pour $j \in \{1, \dots, 31\}$, l'attaquant obtient 32 relations linéaires liant les valeurs $s_{63}^{(0)}, \dots, s_{127}^{(0)}$. En résolvant, le système linéaire associé, il obtient aisément les 32 bits de clé manquants. L'attaquant doit donc résoudre pour chacun des 2^{96} vecteurs possibles $(s_0^{(0)}, \dots, s_{62}^{(0)}, s_0^{(1)}, \dots, s_0^{(31)}) \in \mathbb{F}_2^{96}$ un système linéaire 32×32 sur \mathbb{F}_2 .

L'approche précédente utilise le fait que si certaines variables de l'état interne sont connues, alors la fonction F utilisée dans Toyocrypt devient linéaire. Cependant, le degré élevé de la fonction F semble rendre impossible une attaque qui tenterait de résoudre directement les équations entre les bits de sorties et les bits de l'état interne initial. L'exercice suivant montre que même si F est de degré 63, il est possible d'exprimer les bits de la suite chiffrante par des équations cubiques en les bits de l'état interne du registre.

Exercice 4.7 Attaque algébrique sur Toyocrypt *

Posons

$$G(s_0, \dots, s_{127}) = s_{10}s_{23}s_{32}s_{42}$$

$$+ s_1s_2s_9s_{12}s_{18}s_{20}s_{23}s_{25}s_{26}s_{28}s_{33}s_{38}s_{41}s_{42}s_{51}s_{53}s_{59} + \prod_{i=0}^{62} s_i$$

1. Montrer que $G(s_0, \dots, s_{127}) \cdot (1 + s_{23}) = 0$ et $G(s_0, \dots, s_{127}) \cdot (1 + s_{42}) = 0$ pour tout $(s_0, \dots, s_{127}) \in \mathbb{F}_2^{128}$.
2. En déduire qu'il existe deux équations cubiques liant le bit z_t de la suite chiffrante au pas d'horloge t et le contenu de l'état interne du registre $(s_0^{(t)}, \dots, s_{127}^{(t)})$ au pas d'horloge t .

3. Supposons qu'un attaquant dispose de ℓ bits consécutifs de la suite chiffrante.

Montrer que si

$$2 \cdot \ell \geq \binom{128}{3} + \binom{128}{2} + \binom{128}{1}$$

l'attaquant peut obtenir le contenu de l'état interne initial du registre. Donner la complexité de l'attaque.

Solution

1. Nous avons

$$s_{10}s_{23}s_{32}s_{42} \cdot (1 + s_{23}) = s_{10}s_{23}s_{32}s_{42} + s_{10}s_{23}^2s_{32}s_{42}$$

et sur \mathbb{F}_2 , nous avons $s_{23}^2 = s_{23}$ indépendamment de la valeur de s_{23} . Nous avons donc

$$s_{10}s_{23}s_{32}s_{42} \cdot (1 + s_{23}) = s_{10}s_{23}s_{32}s_{42} + s_{10}s_{23}s_{32}s_{42} = 0$$

Puisque s_{23} apparaît dans chacun des trois monômes de la fonction G , nous obtenons $G(s_0, \dots, s_{127}) \cdot (1 + s_{23}) = 0$ et de même $G(s_0, \dots, s_{127}) \cdot (1 + s_{42}) \equiv 0$ pour tout $(s_0, \dots, s_{127}) \in \mathbb{F}_2^{128}$.

2. Nous avons

$$z_t = F(s_0^{(t)}, \dots, s_{127}^{(t)}) = s_{127}^{(t)} + \sum_{i=0}^{62} s_i^{(t)} s_{\alpha_i}^{(t)} + G(s_0^{(t)}, \dots, s_{127}^{(t)})$$

et donc nous obtenons la relation cubique

$$z_t \cdot (1 + s_{23}^{(t)}) = (1 + s_{23}^{(t)}) \cdot \left(s_{127}^{(t)} + \sum_{i=0}^{62} s_i^{(t)} s_{\alpha_i}^{(t)} \right)$$

Nous obtenons également la relation :

$$z_t \cdot (1 + s_{42}^{(t)}) = (1 + s_{42}^{(t)}) \cdot \left(s_{127}^{(t)} + \sum_{i=0}^{62} s_i^{(t)} s_{\alpha_i}^{(t)} \right)$$

3. Pour chaque bit de la suite chiffrante, l'attaquant dispose de deux relations cubiques liant le contenu de l'état interne du registre au pas d'horloge correspondant. L'état interne évoluant de façon linéaire à partir de l'état interne initial, nous obtenons à chaque fois deux relations cubiques liant le contenu de l'état interne initial. Le polynôme de rétroaction de Toyocrypt étant primitif, ces relations sont linéairement indépendantes.

Les variables apparaissant dans ces équations sont :

- les variables de degré 1, $s_i^{(0)}$ pour $i \in \{0, \dots, 127\}$;
- les variables de degré 2, $s_i^{(0)}s_j^{(0)}$ pour $i, j \in \{0, \dots, 127\}$ (avec $i \neq j$ puisque $s_i^{(0)2} = s_i^{(0)}$);
- les variables de degré 3, $s_i^{(0)}s_j^{(0)}s_k^{(0)}$ pour $i, j, k \in \{0, \dots, 127\}$ (avec $i \neq j, i \neq k$ et $j \neq k$).

En linéarisant le problème (*i.e.* en considérant chaque variable indépendamment sans prendre en compte leurs relations), l'attaquant obtient un système linéaire à

$$\binom{128}{3} + \binom{128}{2} + \binom{128}{1}$$

variables. Dès que le nombre d'équations 2ℓ est égal à cette valeur, il peut donc résoudre le système et obtenir le contenu de l'état initial du registre. Le nombre de variables est égal à $349632 < 2^{19}$ et en utilisant un algorithme cubique (comme le pivot de Gauss) pour résoudre le système linéaire, la complexité de l'attaque est de l'ordre de 2^{57} opérations élémentaires.

4.4 CHIFFREMENT PAR FLOT PAR REGISTRES COMBINÉS

Un système de chiffrement par flot par registres combinés est composé de $n \geq 2$ registres à décalage à rétroaction linéaire qui fonctionnent en parallèle et dont les sorties sont combinées au moyen d'une fonction booléenne à n variables (*cf.* Figure 4.4). La valeur de cette fonction à chaque pas d'horloge fournit un bit de la suite chiffrante.

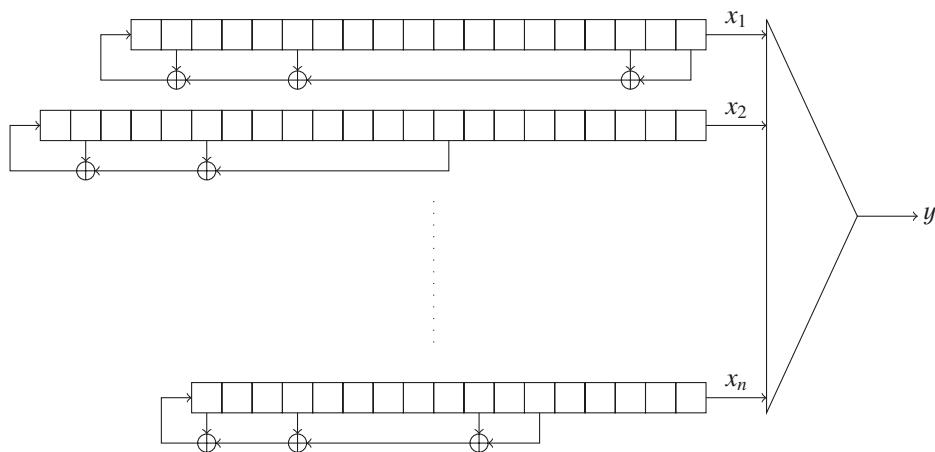


Figure 4.4 - Registres à rétroaction linéaire combinés

Le générateur de Geffe est un exemple de système de chiffrement par flot par registres combinés qui a été proposé en 1973 par P.R. GEFFE [27] (*cf.* Figure 4.5). Il est composé de trois LFSR de longueurs distinctes combinés par la fonction

$$F(x_1, x_2, x_3) = x_1 x_2 \oplus x_2 x_3 \oplus x_3$$

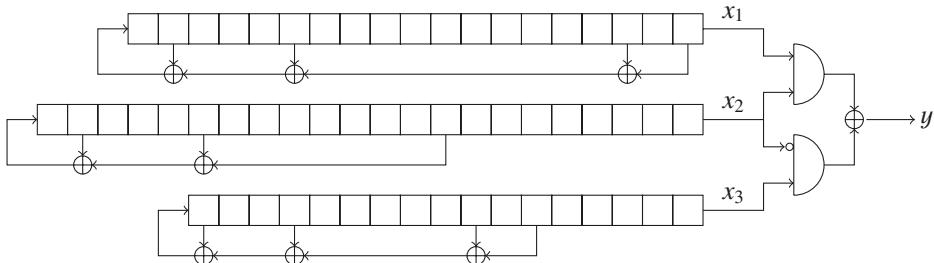


Figure 4.5 - Générateur de Geffe

Nous allons voir que le générateur de Geffe est vulnérable à plusieurs types d'attaques.

En 1985, T. SIEGENTHALER a proposé une méthode de cryptanalyse sur les systèmes de chiffrement par flot par registres combinés appelée *attaque par corrélation*. L'idée de cette méthode est d'exploiter l'existence d'une éventuelle corrélation entre la suite chiffrante et le contenu de certains registres. Un attaquant peut alors chercher séparément les valeurs initiales de chaque registre et retrouver la clé plus rapidement qu'en effectuant une recherche exhaustive. Le but de l'exercice suivant est de montrer que le générateur de Geffe est vulnérable à une telle attaque.

Exercice 4.8 Attaque par corrélation sur le générateur de Geffe

Considérons trois registres à décalage à rétroaction linéaire R_1 , R_2 et R_3 de longueur ℓ_1 , ℓ_2 et ℓ_3 (respectivement) et le générateur de Geffe associé. Notons $x_1(t)$, $x_2(t)$, $x_3(t)$ les sorties respectives de trois registres à décalage à rétroaction linéaire R_1 , R_2 et R_3 et $z(t)$ la suite chiffrante produite par le générateur de Geffe associé.

1. En supposant que les valeurs $x_i(t)$ suivent des lois de Bernoulli indépendantes de paramètre $1/2$, montrer que

$$\Pr(z(t) = x_1(t)) = \frac{3}{4} \text{ et } \Pr(z(t) = x_3(t)) = \frac{3}{4}$$

2. En déduire une attaque sur le générateur de Geffe.

Solution

1. Puisque $f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$, nous avons (en fonction de la valeur de $x_2(t)$) :

$$z(t) = \begin{cases} x_3(t) & \text{si } x_2(t) = 0 \\ x_1(t) & \text{si } x_2(t) = 1 \end{cases}$$

En particulier, nous avons

$$\begin{aligned}
 \Pr[z(t) = x_1(t)] &= \Pr[x_2(t) = 0] \Pr[z(t) = x_1(t)|x_2(t) = 0] \\
 &\quad + \Pr[x_2(t) = 1] \Pr[z(t) = x_1(t)|x_2(t) = 1] \\
 &= \frac{1}{2} \Pr[z(t) = x_1(t)|x_2(t) = 0] + \frac{1}{2} \Pr[z(t) = x_1(t)|x_2(t) = 1] \\
 &= \frac{1}{2} \Pr[x_3(t) = x_1(t)|x_2(t) = 0] + \frac{1}{2} \Pr[x_1(t) = x_1(t)|x_2(t) = 1] \\
 &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot 1 = \frac{3}{4}
 \end{aligned}$$

De même

$$\begin{aligned}
 \Pr[z(t) = x_3(t)] &= \frac{1}{2} \Pr[x_3(t) = x_3(t)|x_2(t) = 0] + \frac{1}{2} \Pr[x_3(t) = x_1(t)|x_2(t) = 1] \\
 &= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}
 \end{aligned}$$

2. L'attaque consiste à deviner successivement le contenu initial des registres R_1 et R_3 . Pour tous les contenus initiaux possibles du premier registre, l'attaquant génère une suite de longueur n pour ce registre. Si la suite chiffrante et la sortie du LFSR ne sont pas corrélées, le nombre moyen de valeurs communes sera le même qu'avec une suite aléatoire (soit en moyenne $n/2$) alors que pour le contenu initial correspond à l'état interne du générateur de Geffe, ce nombre sera proche de $3n/4$.

Une fois l'état initial de R_1 trouvé, l'attaquant peut répéter l'attaque pour obtenir l'état initial de R_3 . Une fois ces deux registres connus, l'attaquant peut retrouver simplement le contenu de R_2 en appliquant l'algorithme de l'exercice 4.3 ou l'algorithme de Berlekamp-Massey. La recherche exhaustive est l'étape la plus coûteuse de cette attaque. En notant ℓ_i la longueur du registre R_i , elle nécessite environ $(2^{\max(\ell_1, \ell_3)+1}) \cdot n$ itérations d'un registre de taille $\max(\ell_1, \ell_3)$.

La corrélation étant de bonne qualité, une petite valeur de n est suffisante pour distinguer les suites correctes des suites aléatoires (par exemple $n = 64$) et l'attaque est de l'ordre de $(2^{\max(\ell_1, \ell_3)+7}) \max(\ell_1, \ell_3)^3$ opérations élémentaires (au lieu de $2^{\ell_1+\ell_2+\ell_3}$ pour une recherche exhaustive).

L'attaque précédente utilise le fait que les valeurs des premier et troisième registres sont corrélées à la suite chiffrante sans utiliser d'information sur le contenu du second registre. L'exercice suivant montre que si le contenu du second registre est connu d'un attaquant, alors il peut facilement retrouver le contenu des deux autres registres.

Exercice 4.9 Attaque « deviner et déterminer » sur le générateur de Geffe

Considérons trois registres à décalage à rétroaction linéaire R_1 , R_2 et R_3 de longueur ℓ_1 , ℓ_2 et ℓ_3 (respectivement) et le générateur de Geffe associé. Montrer que si un attaquant connaît le contenu de ce second registre du générateur de Geffe, il peut retrouver le contenu des registres R_1 et R_2 avec une attaque de complexité de l'ordre de $\ell_1^3 + \ell_3^3$. En déduire une attaque de complexité de l'ordre de $2^{\ell_2}(\ell_1^3 + \ell_3^3)$ opérations élémentaires.

Solution

Si l'attaquant connaît le contenu du registre R_2 , il sait pour chaque bit de la suite filtrante s'il provient du premier registre ou du troisième. Chaque bit lui donne donc une équation linéaire sur le contenu d'état interne de l'un des registres (mais pas forcément consécutifs). Lorsqu'il a obtenu ℓ_1 (*resp.* ℓ_3) équations sur les états internes de R_1 (*resp.* R_3), l'attaquant peut simplement résoudre le système linéaire associé pour retrouver la valeur des contenus initiaux des deux registres.

Pour « deviner » l'état initial du registre R_2 , l'attaquant va simplement faire une recherche exhaustive sur tous les états possibles et pour chacun d'eux appliquer la résolution des systèmes linéaires. Lorsque le contenu des deux registres R_1 et R_3 est « déterminé », l'attaquant peut tester la validité de la clé obtenue en utilisant des bits supplémentaires de la suite chiffrante pour filtrer les clés incorrectes.

Le générateur de Geffe utilise une fonction de combinaison très simple et en particulier de petit degré algébrique. L'exercice suivant montre que l'utilisation d'une telle fonction dans un système par registres combinés entraîne des failles de sécurité.

Exercice 4.10 Attaque algébrique sur le générateur de Geffe

Considérons trois registres à décalage à rétroaction linéaire R_1 , R_2 et R_3 de longueur ℓ_1 , ℓ_2 et ℓ_3 (respectivement) et le générateur de Geffe associé (où le registre R_2 est celui qui contrôle le flux de sortie du générateur). Montrer que si l'attaquant dispose de plus de $\ell(\ell+1)/2$ bits de suite chiffrante, il peut retrouver le contenu des états internes en temps $O(\ell^6)$ où $\ell = \max(\ell_1 + \ell_2 + \ell_3)$.

Solution

Avec les notations de l'exercice 4.8, pour tout entier t , nous avons

$$z(t) = x_1(t)x_2(t) \oplus x_2(t)x_3(t) \oplus x_3(t)$$

Pour tout entier t , $x_1(t)$ s'exprime linéairement en fonction des valeurs $x_1(0), \dots, x_1(\ell_1 - 1)$ et un attaquant connaît les coefficients de cette équation linéaire puisqu'ils ne dépendent que du polynôme de rétroaction de R_1 :

$$x_1(t) = \varphi_1^{(t)}(x_1(0), \dots, x_1(\ell_1 - 1))$$

De même, $x_2(t)$ et $x_3(t)$ s'expriment linéairement en fonction des valeurs $x_2(0), \dots, x_1(\ell_2 - 1)$ et $x_3(0), \dots, x_3(\ell_3 - 1)$ (respectivement) :

$$x_2(t) = \varphi_2^{(t)}(x_2(0), \dots, x_2(\ell_2 - 1)) \text{ et } x_3(t) = \varphi_3^{(t)}(x_3(0), \dots, x_3(\ell_3 - 1))$$

La relation $z(t) = x_1(t)x_2(t) \oplus x_2(t)x_3(t) \oplus x_3(t)$, s'exprime sous la forme

$$z(t) = \sum_{i=1}^3 \sum_{j=1}^{\ell_i} a_{i,j} x_i(j) + \sum_{i_1=1}^3 \sum_{i_2=1}^3 \sum_{j_1=1}^{\ell_{i_1}} \sum_{j_2=1}^{\ell_{i_2}} b_{i_1, i_2, j_1, j_2} x_{i_1}(j_1) x_{i_2}(j_2)$$

où les coefficients $a_{i,j}$ et b_{i_1, i_2, j_1, j_2} sont publiquement calculables à partir des polynômes de rétroaction des registres R_1 , R_2 et R_3 . Dans le corps à deux éléments \mathbb{F}_2 , tout élément x vérifie $x^2 = x$. Le nombre de variables apparaissant dans l'expression de $z(t)$ est inférieur au nombre de variables linéaires ℓ ajouté au nombre de variables quadratiques $x_{i_1}(j_1)x_{i_2}(j_2)$ avec $(i_1, j_1) \neq (i_2, j_2)$, c'est-à-dire majoré par $\ell + \ell(\ell - 1)/2 = \ell(\ell + 1)/2$.

En supposant que ces équations sont linéairement indépendantes, l'attaquant peut résoudre le système linéaire correspondant s'il dispose de $\ell(\ell + 1)/2$ bits de la suite chiffrante $z(t)$. Résoudre ce système linéaire par une méthode cubique comme le pivot de Gauss requiert un temps de calcul de l'ordre de $O((\ell^2)^3)$ opérations élémentaires.

4.5 LE CHIFFREMENT PAR FLOT A5/1

Le système de chiffrement par flot A5/1 est utilisé dans le cadre des communications hertziennes en téléphonie mobile. Il utilise des clés de 64 bits (même si son implantation en téléphonie mobile n'utilise que 54 bits effectifs). Le système A5/1 est un système par registres à décalage linéaire combinés où le décalage des registres est irrégulier. Il est composé de trois LFSR (de longueurs 19, 22 et 23 bits) combinés par « ou exclusif ». L'horloge de chacun de ces registres est contrôlée par une fonction de transition particulière : le système calcule la fonction « majorité » de trois bits (appelés bits d'horloge) à des positions fixes, (un par registre) et seuls les registres ayant produit le bit qui a emporté la majorité sont alors décalés. Le système A5/1 est décrit dans la figure 4.6. Les polynômes de rétroaction utilisés par A5/1 sont donnés dans le tableau 4.1.

Registre	Longueur	Polynôme de rétroaction	Bit d'horloge
1	19	$x_{19} + x_{18} + x_{17} + x_{14} + 1$	8
2	22	$x_{22} + x_{21} + 1$	10
3	23	$x_{23} + x_{22} + x_{21} + x_8 + 1$	10

Tableau 4.1 – Polynômes de rétroaction pour les registres de A5/1

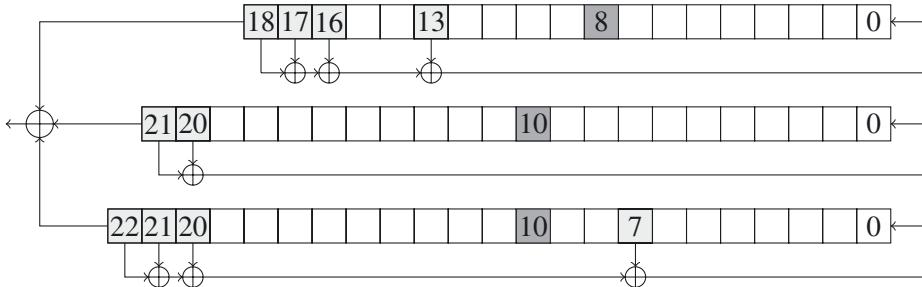


Figure 4.6 - Description du chiffrement par flot A5/1

Notons qu'il n'existe aucune description officielle de A5/1 (les détails de l'algorithme ont été publiés anonymement sur une liste de diffusion sur Internet, vraisemblablement suite à une rétro-ingénierie). L'état interne du registre est initialisé à partir de la clé de 64 bits et d'un vecteur d'initialisation de 22 bits (qui est simplement un compteur changé tous les 228 bits de la suite chiffrante). L'exercice suivant montre qu'il est relativement facile de retrouver l'état interne initial (et donc la clé) à partir d'un état interne après t applications de la fonction de transition de l'algorithme A5/1 (et de cette valeur $t \geq 0$).

Exercice 4.11 États internes de A5/1

Soit $E \in \{0, 1\}^{64}$ un état interne pour A5/1 (*i.e.* les 19 premiers bits correspondent au contenu de premier registre, les 22 suivants à celui du deuxième registre et les 23 derniers à ceux du troisième registre) et soit n_E le nombre d'états internes tel que l'application de la transition de A5/1 donne E .

- Montrer que si E est tiré uniformément aléatoirement dans $\{0, 1\}^{64}$, alors la variable aléatoire n_E vérifie

$$\Pr[n_E = 0] = 3/8$$

$$\Pr[n_E = 1] = 13/32$$

$$\Pr[n_E = 2] = \Pr[n_E = 3] = 3/32$$

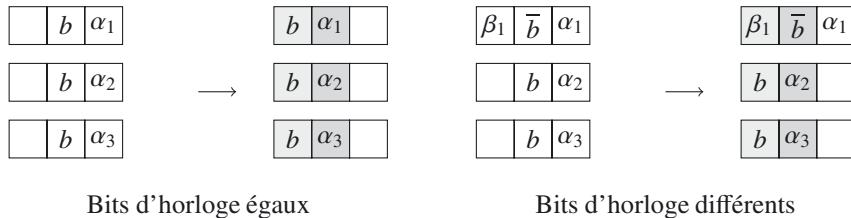
$$\Pr[n_E = 4] = 1/32.$$

- En déduire, étant donnés un état E^* et un entier $t \geq 1$, le nombre heuristique d'états pour lesquels t applications de la fonction de transition du chiffrement A5/1 produisent l'état E^* .

Solution

- Dans un état interne d'A5/1, pour déterminer le décalage des registres, nous avons deux configurations possibles : les trois bits d'horloge ont la même valeur b ou seule-

ment deux sont égaux à b et le troisième est égal à $1 - b$. Le diagramme suivant illustre l'évolution de l'état interne en fonction de ces deux configurations :



Le nombre d'états produisant un état E dépend donc uniquement des bits d'horloge h_1 , h_2 et h_3 aux positions 8, 10 et 10 dans les premier, second et troisième registres respectivement (en gris foncé sur le diagramme) et des bits g_1 , g_2 et g_3 situés à gauche, c'est-à-dire les bits aux positions 9, 11 et 11 dans les premier, second et troisième registres (en gris clair sur le diagramme).

Ainsi, si lors de l'étape précédente, les bits d'horloge sont différents, les deux registres « majoritaires » vont être décalés alors que celui dont le bit est unique va rester immobile. Par conséquent, il est impossible qu'une configuration du type $(h_1, h_2, h_3) = (0, 1, 0)$ et $(q_1, q_2, q_3) = (0, 0, 1)$ ait un antécédent.

En raisonnant ainsi pour les 64 choix possibles des vecteurs (h_1, h_2, h_3) et (g_1, g_2, g_3) , nous obtenons les nombres d'antécédents possibles n_E pour chaque configuration avec $g_1 = 0$ dans le tableau 4.2 (le nombre d'antécédent des configurations avec $g_1 = 1$ a évidemment la même distribution).

g_1	g_2	g_3	h_1	h_2	h_3	n_E
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	2
0	0	0	0	1	1	3
0	0	0	1	0	0	2
0	0	0	1	0	1	3
0	0	0	1	1	0	3
0	0	0	1	1	1	4
0	0	1	0	0	0	0
0	0	1	0	0	1	1
0	0	1	0	1	0	0
0	0	1	0	1	1	1
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	0	1	1	1	1	1
0	0	1	1	1	1	1

g_1	g_2	g_3	h_1	h_2	h_3	n_E
0	1	0	0	0	0	0
0	1	0	0	0	1	0
0	1	0	0	1	0	1
0	1	0	0	1	1	1
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	0	1
0	1	1	0	0	1	1
0	1	1	0	1	0	1
0	1	1	0	1	1	1
0	1	1	1	0	0	0
0	1	1	1	0	1	0
0	1	1	1	1	0	0
0	1	1	1	1	1	0

Tableau 4.2 - Nombres d'antécédents possibles en fonctions des vecteurs (h_1, h_2, h_3) et (q_1, q_2, q_3)

2. D'après la question précédente, le nombre d'antécédents d'un état E^* uniformément aléatoire par une transition du chiffrement A5/1 est égal à

$$\mathbb{E}(n_{E^*}) = \sum_{n=0}^4 n \cdot \Pr[n_{E^*} = n] = \frac{13}{32} + 2 \cdot \frac{3}{32} + 3 \cdot \frac{3}{32} + 4 \cdot \frac{1}{32} = 1$$

Heuristiquement, le nombre d'états pour lesquels t transitions du chiffrement A5/1 produit l'état E^* est donc égal à $\mathbb{E}(n_{E^*})^t = 1$.

En utilisant cette propriété, J.D. GOLIC a montré à la conférence *Eurocrypt 1997* [28]. que le système de chiffrement par flot A5/1 est vulnérable aux attaques par compromis temps-mémoire. Ce type d'attaque se trouve à mi-chemin entre une recherche exhaustive de l'état interne des registres et un stockage complet préalable de toutes les états internes possibles correspondant à une chaîne de la suite chiffrante.

Exercice 4.12 Attaque par compromis temps-mémoire sur A5/1

Soit $(s_n)_{n \in \{0, \dots, 63+\ell\}}$ une séquence de $64 + \ell$ bits consécutifs d'une suite chiffrante produite par un état interne de A5/1 inconnu. Calculer en fonction de ℓ et de T , une probabilité heuristique que l'algorithme 4.1 retourne un état interne E .

Algorithme 4.1 Attaque par compromis temps-mémoire sur A5/1

ENTRÉE: $(s_n)_{n \in \{0, \dots, 63+\ell\}}$

SORTIE: $E \in \{0, 1\}^{64}$ ou ÉCHEC

$L \leftarrow \emptyset$

pour i de 0 à T **faire**

$E_i \xleftarrow{\text{u.a.}} \{0, 1\}^{64}$

$\vec{a}_i = (a_0^{(i)}, \dots, a_{63}^{(i)}) \leftarrow \text{A5/1}(E_i)$

$L \leftarrow L \cup \{(E_i, \vec{a}_i)\}$

▷ liste triée selon la seconde coordonnée

fin pour

pour i de 0 à ℓ **faire**

si $(E, (s_n)_{n \in \{i, \dots, 63+i\}}) \in L$ **alors**

retourner E

fin si

fin pour

retourner ÉCHEC

En déduire une attaque et donner sa complexité en temps et en mémoire.

Solution

Nous présentons seulement une solution intuitive car l'analyse formelle de la probabilité de succès de l'algorithme est très proche de celle vue dans l'exercice 3.6.

Étant donnée une suite $\tilde{s} \in \mathbb{F}_2^{64}$, il existe en moyenne un état E qui produit la séquence \tilde{s} comme 64 premiers bits de suite chiffrante.

Dans la première étape de l'algorithme, une liste L est construite. Elle contient des états internes $E_i \in \{0, 1\}^{64}$ de A5/1 et les 64 premiers bits de la suite chiffrante associée $\vec{a}_i = (a_0^{(i)}, \dots, a_{63}^{(i)})$. Nous supposons que T est suffisamment petit par rapport à 2^{64} pour que le nombre de valeurs communes obtenues parmi toutes les suites chiffrantes obtenues soit petit et que la liste L contienne donc environ T couples.

Dans la seconde étape de l'algorithme, l'attaquant teste pour chaque sous-suite de longueur 64 de $(s_n)_{n \in \{0, \dots, 63+\ell\}}$ si l'une d'elles appartient à la liste L et si c'est le cas, il retourne l'un des états E correspondants. En supposant les sous-suites indépendantes², chacune d'elle apparaît dans la liste L avec probabilité $\#L/2^{64}$ et l'algorithme retourne donc un état E avec une probabilité proche de $(\ell \cdot T)/2^{64}$.

Si $\ell \cdot T = 2^{64}$, l'algorithme réussit avec une probabilité (heuristique) proche de 1. L'attaquant dispose alors d'un état interne E et d'une valeur t tels que E produit la suite chiffrante à partir de l'indice t . En inversant la fonction de transition comme dans l'exercice précédent, l'adversaire obtient l'état interne initial (et donc la clé).

La complexité de l'attaque est de $T \cdot 128$ bits en mémoire et T évaluations de A5/1 pour la première étape de l'algorithme et ℓ recherches dans la liste L dans la seconde partie de l'algorithme. Le coût de ces deux étapes est équilibré pour $\ell \approx T \approx 2^{32}$ (et pour cette valeur de T le nombre de valeurs communes obtenues parmi toutes les suites chiffrantes obtenues dans la première étape de l'algorithme est effectivement petit). L'attaquant doit alors pré-calculer 2^{32} valeurs de la suite chiffrante de A5/1 et les stocker (ce qui représente 2^{39} bits) et il doit obtenir 2^{32} bits consécutifs de la suite chiffrante attaquée.

Note

Le chiffrement par flot A5/1 réinitialise l'état interne lorsque 228 bits de suite chiffrante ont été produits. L'attaque précédente s'adapte cependant aisément en utilisant plusieurs suites chiffrantes correspondant à des vecteurs d'initialisation différents.

Problème 4.13 Attaque « deviner et déterminer » sur A5/1

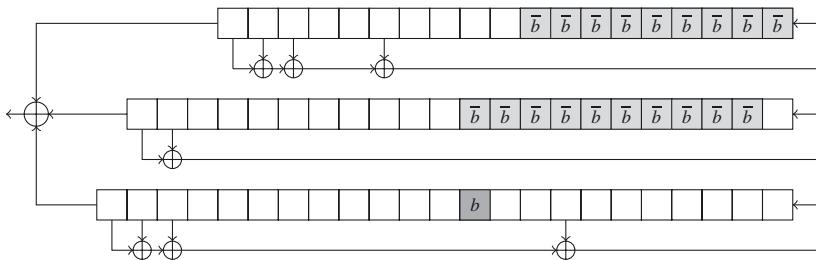
1. Supposons que pour une clé donnée le troisième registre R_3 utilisé dans A5/1 n'est pas décalé pendant 10 tours consécutifs. Montrer que si un attaquant devine 2 bits de R_3 , 9 bits de R_1 et un bit de R_2 bien choisis, alors il peut déduire le contenu complet des registres R_1 et R_2 à partir de la suite chiffrante.
2. Montrer que si un attaquant devine en plus 10 bits de R_3 il peut reconstruire le contenu complet de R_3 .
3. En déduire une attaque à clairs connus contre A5/1.

2. Il s'agit évidemment d'une approximation de la réalité.

Solution

1. Supposons que pendant 10 itérations, le registre R_3 n'est pas décalé. Si l'attaquant devine correctement la valeur b de $R_3[10]$, il obtient la valeur des bits contrôlant l'horloge des deux autres registres $R_1[8]$ et $R_2[10]$ (il est égal à $\bar{b} = 1 - R_3[10]$) et c'est également le cas des valeurs $R_2[i]$ pour $i \in \{1, \dots, 9\}$ et $R_1[i]$ pour $i \in \{0, \dots, 7\}$.

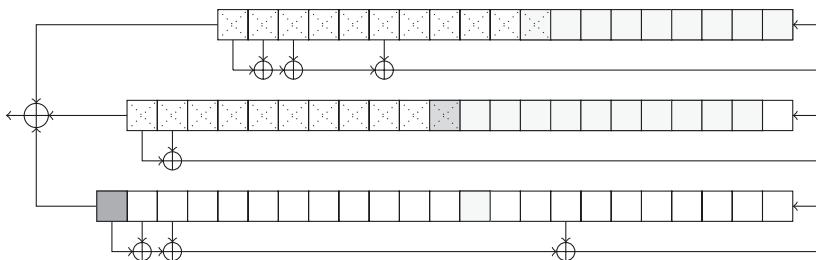
L'attaquant sait également que la valeur qui va entrer dans le registre R_1 au tour suivant est aussi égale à b . En indiquant en gris foncé le bit deviné et en gris clair les valeurs déduites, la situation est décrite par le diagramme suivant :



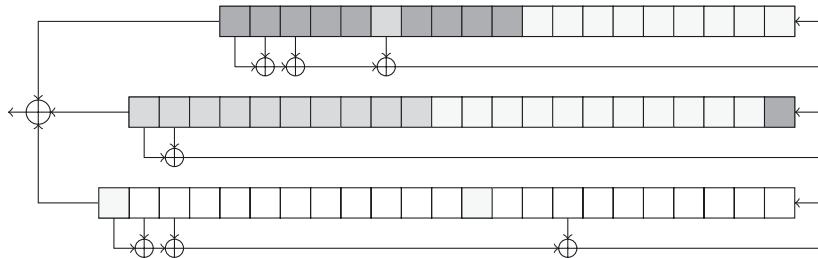
En devinant la valeur de $R_3[22]$ pour ces 10 tours, l'attaquant sait que le bit de la suite chiffrante est égal au « ou exclusif » de $R_3[22]$ et des bits de sorties des registres R_1 et R_2 . Il y a (au moins) 11 tours pour lesquels le même bit de sortie est utilisé pour R_3 , et l'attaquant obtient donc 11 équations linéaires liant les contenus des deux registres R_1 et R_2 . Plus précisément, il obtient les équations

$$z_{18-t} \oplus R_3[22] = R_1[t] \oplus R_2[t+3] \text{ pour } t \in \{8, \dots, 18\}$$

où z_0, \dots, z_{10} désignent les 11 bits consécutifs de la suite chiffrante produits pour ces 11 tours. En particulier, puisque $R_1[8]$ est connu, l'attaquant peut utiliser le onzième bit de la suite chiffrante pour obtenir la valeur de $R_2[11] = z_{10} \oplus R_1[8]$. Le diagramme suivant (où les cases barrées indiquent les cases pour lesquelles l'attaquant dispose de relations linéaires) illustre l'état des connaissances de l'attaquant :

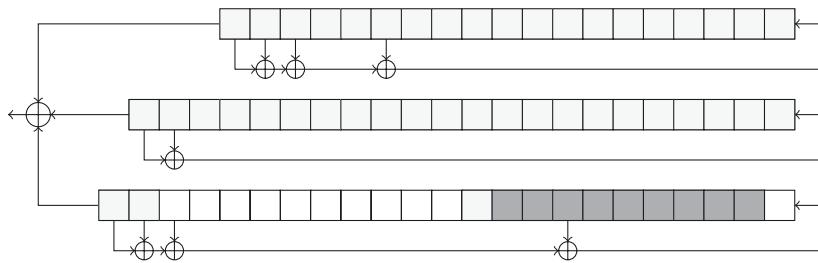


En devinant ensuite 9 bits parmi les 10 restants, à savoir les valeurs de $R_1[i]$ pour $i \in \{9, 10, 11, 12, 14, 15, 16, 17, 18\}$, et le bit $R_2[0]$, l'attaquant obtient tous les bits de R_1 et R_2 . En effet, il n'est pas nécessaire de deviner $R_1[13]$ puisqu'on sait que la valeur $R_1[18] \oplus R_1[17] \oplus R_1[16] \oplus R_1[13]$ va entrer dans $R_1[0]$ au prochain pas d'horloge (et donc est égale à \bar{b}).



L'attaquant doit donc deviner 12 bits pour retrouver tous les bits de \$R_1\$ et \$R_2\$.

2. Avec la connaissance des registres \$R_1\$ et \$R_2\$, l'attaquant peut décider à quel moment le registre \$R_3\$ va de nouveau avancer et en calculant le « ou exclusif » de la suite chiffrante avec les derniers bits de \$R_1\$ et \$R_2\$ à ce pas d'horloge, il obtient la valeur \$R_3[21]\$. Pour continuer à faire évoluer le registre \$R_3\$, l'attaquant doit deviner les valeurs de \$R_3[i]\$ pour \$i \in \{0, \dots, 9\}\$.



Chaque fois que le registre \$R_3\$ avance, nous obtenons un nouveau bit de \$R_3\$ et il suffit d'attendre 12 décalages supplémentaires pour retrouver le registre entier. Comme la probabilité que le registre avance à chaque étape est égale à 3/4, il suffit d'attendre 16 tours en moyenne pour obtenir l'état interne complet de \$R_3\$.

3. Après avoir deviné les douze bits nécessaires pour reconstruire les registres \$R_1\$ et \$R_2\$ et les dix bits supplémentaires de \$R_3\$, l'attaquant obtient l'état interne complet du générateur. Il teste sa validité en regardant si cet état est cohérent avec les bits suivants de la suite chiffrante. Le coût amorti de cette vérification est celui de la génération de deux bits de suite chiffrante en moyenne (une clé incorrecte sur deux sera éliminée par le premier bit de suite chiffrante, une clé incorrecte sur quatre par le second bit...).

Le coût complet de l'attaque est donc égal, en faisant une recherche exhaustive sur \$12 + 10 = 22\$ bits, à seize itérations de A5/1 pour reconstruire l'état interne entier et 2 itérations (en moyenne) pour supprimer les clés incorrectes. Le coût complet de l'attaque (lorsque l'instant à partir duquel le troisième registre \$R_3\$ n'est pas décalé pendant 10 tours est connu) est de \$2^{12} \cdot 2^{10} \cdot 2^4 \cdot 2 = 2^{27}\$ itérations de A5/1.

Cet instant est bien sûr inconnu de l'attaquant. Il doit répéter cette attaque jusqu'à ce que cet événement arrive. Cet événement est réalisé lorsque \$R_1[i]\$ et \$R_2[j]\$ sont tous égaux pour \$i \in \{-1, \dots, 8\}\$ et \$j \in \{1, \dots, 10\}\$, ce qui arrive avec probabilité \$2^{-20}\$ et l'attaquant doit donc répéter l'attaque \$2^{20}\$ fois pour retrouver un état interne (puis

l'état interne initial comme dans les exercices précédents). L'attaque demande donc 2^{20} bits de suite chiffrante (ce qui représente environ 2,36 minutes de conversation en téléphonie mobile) et un coût algorithmique équivalent à 2^{47} itérations de A5/1.

4.6 LE CHIFFREMENT PAR FLOT RC4

Le système de chiffrement par flot RC4 est un algorithme qui a été conçu en 1987 par R. RIVEST. Il a été utilisé dans différents protocoles et notamment le WEP (de l'anglais *Wired Equivalent Privacy*) utilisé pour sécuriser les réseaux sans fil de type Wi-Fi. Comme A5/1, il n'existe aucune description officielle de RC4.

Le système RC4 fonctionne de la façon suivante : une permutation S des 256 octets est construite à partir d'une clé. En utilisant des opérations très simples sur le contenu de ce tableau S (échange de cases, addition modulaire, ...), une suite chiffrante est ensuite générée. L'algorithme de génération de la suite filtrante (*cf.* Algorithme 4.2) prend en entrée le tableau S et un entier $n \geq 1$ et retourne une suite de n octets $(z_1, \dots, z_n) \in \{0, \dots, 255\}^n$ (*cf.* Figure 4.7).

Algorithme 4.2 Génération de la suite chiffrante de RC4

ENTRÉE: $S \in \mathfrak{S}_{\{0, \dots, 255\}}$, $n \in \mathbb{N}$

SORTIE: $(z_1, \dots, z_n) \in \{0, \dots, 255\}^n$

$i \leftarrow 0$

$j \leftarrow 0$

pour ℓ de 1 à n **faire**

$i \leftarrow i + 1 \bmod 256$

$j \leftarrow (j + S[i]) \bmod 256$

$S[i] \leftrightarrow S[j]$

▷ échange des deux valeurs $S[i]$ et $S[j]$

$z_\ell \leftarrow S[(S[i] + S[j]) \bmod 256]$

fin pour

retourner z_1, \dots, z_n

▷ suite chiffrante de n octets

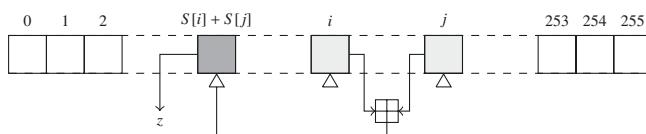


Figure 4.7 - Description du chiffrement par flot RC4

L'exercice suivant montre que les échanges dans l'algorithme de génération de la suite chiffrante sont essentiels pour assurer la sécurité de RC4.

Exercice 4.14 Cryptanalyse de RC4 sans opération d'échange *

Considérons une variante simplifiée de RC4 où l'opération d'échange de $S[i]$ et de $S[j]$ de la boucle principale de l'algorithme 4.2 n'est jamais effectuée.

1. Montrer que la suite chiffrante est périodique de période 512.
2. En déduire un moyen de reconstruire la permutation S à partir de la suite chiffrante.

Solution

1. En notant i_t et j_t les valeurs de i et j à la t -ième itération de la boucle, nous avons $i_{t+256} = i_t \bmod 256$ et

$$j_{t+256} = j_t + \sum_{\ell=0}^{255} S[i_{t+\ell}] = j_t + \sum_{\ell=0}^{255} S[\ell] \bmod 256$$

Puisque S est une permutation, nous avons

$$\sum_{\ell=0}^{255} S[\ell] = \sum_{\ell=0}^{255} \ell = (256 \cdot 255)/2 \equiv 128 \bmod 256$$

et nous obtenons $j_{t+256} = j_t + 128$. Nous avons donc

$$j_{t+512} = j_{t+256} + 128 = j_t + 256 \equiv j_t \bmod 256$$

et par conséquent

$$z_{t+512} = S[S[i_{t+512}] + S[j_{t+512}]] = S[S[i_t] + S[j_t]] = z_t$$

2. L'attaque consiste à deviner une partie des valeurs de la permutation S et à construire les autres à partir de la suite chiffrante. Si l'attaquant obtient une contradiction, il recommence avec d'autres valeurs initiales.

Dans l'exécution de l'algorithme RC4 le t -ième octet de la suite chiffrante dépend de j_t , $S[i_t]$ et $S[j_t]$. Si ces trois valeurs sont connues de l'attaquant et si $z_t \neq S[S[i_t] + S[j_t]] \bmod 256$, alors il obtient une contradiction.

Par ailleurs, si trois des quatre valeurs j_t , $S[i_t]$, $S[j_t]$ et $S^{-1}[z_t]$ sont connues de l'adversaire, il peut déterminer explicitement la quatrième en utilisant les relations :

$$j_t = S^{-1}(S^{-1}[z_t] - S[i_t]) \bmod 256$$

$$S[i_t] = S^{-1}[z_t] - S[j_t] \bmod 256$$

$$S[j_t] = S^{-1}[z_t] - S[i_t] \bmod 256$$

$$S^{-1}[z_t] = S[i_t] + S[j_t] \bmod 256$$

L'attaquant peut donc choisir des valeurs initiales de $S[1], \dots, S[v]$ pour $v \in \mathbb{N}$, afin de calculer les valeurs de j_t pour $t \in \{1, \dots, v\}$. Pendant ces calculs, il obtient de nouvelles valeurs de S en utilisant les relations précédentes.

Si l'attaquant n'obtient pas la valeur $S[v+1]$ pendant ce calcul, il ne peut pas obtenir la valeur de j_{v+1} . Il peut cependant essayer de reconstruire une valeur ultérieure de j_t (avec $t > v+1$) en utilisant la valeur de z_t et la quatrième relation. Il peut ainsi continuer l'attaque tout en testant si la valeur obtenue est cohérente avec les valeurs antérieures.

Dès qu'une contradiction est obtenue l'attaquant recommence avec un nouveau choix de $S[1], \dots, S[v]$. La valeur de v doit être suffisamment grande pour permettre de calculer plusieurs valeurs initiales pour j_t mais pas trop grande pour rendre la recherche exhaustive faisable. L'analyse théorique de cette attaque est difficile à réaliser mais elle fonctionne bien en pratique.

La suite chiffrante générée par RC4 est légèrement biaisée en faveur de certaines séquences d'octets. L'exercice suivant montre un exemple d'un tel biais et une attaque due à I. MANTIN et A. SHAMIR [41] qui s'applique si le système RC4 est utilisé pour transmettre le même message clair à plusieurs utilisateurs.

Exercice 4.15 Biais de la suite chiffrante produite par RC4

Notons $S_t[i]$ la valeur de la permutation S en l'octet i après le t -ième tour de boucle de l'algorithme (4.2).

1. Montrer que si $S_0[2] = 0$ et $S_0[1] \neq 2$, alors $z_2 = 0$.
2. En déduire que la distribution de z_2 n'est pas uniforme lorsque S est initialisée à une permutation aléatoire S_0 .
3. Supposons que le même texte clair est chiffré pour n destinataires ayant des clés RC4 différentes (avec n suffisamment grand). Proposer un algorithme qui retrouve le second octet du message clair.

Solution

1. Notons i_t et j_t les valeurs de i et j après le t -ième tour de boucle de l'algorithme (4.2). Posons $x = S_0[1]$, $y = S_0[x]$ et $i_0 = j_0 = 0$. Au premier tour de boucle, nous avons $i_1 = 1$ et $j_1 = 0 + S_0[i_1] = S_0[1] = x$, l'algorithme échange $S_0[1]$ et $S_0[x]$ pour obtenir S_1 . Seules ces deux valeurs sont modifiées et en particulier $S_1[2] = S_0[2] = 0$. L'algorithme retourne comme premier octet de la suite chiffrante

$$z_1 = S_1[S_1[1] + S_1[x] \bmod 256] = S_1[x + y]$$

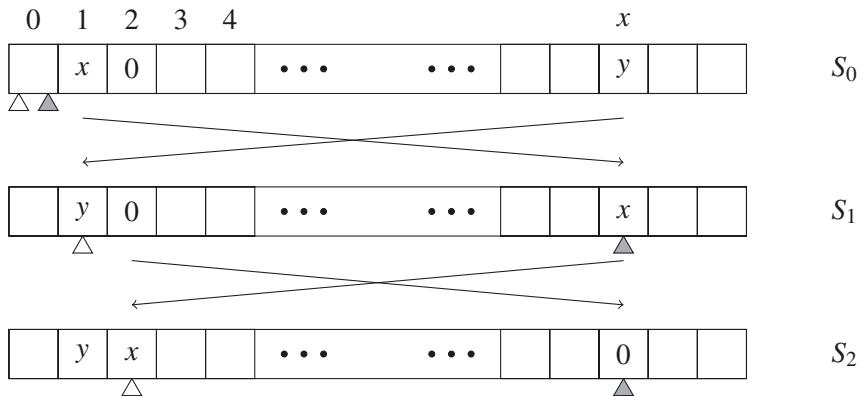
Pour le deuxième tour, nous obtenons avec $i_2 = 2$,

$$j_2 = j_1 + S_1[i_2] = x + S_1[i_2] = x + S_1[2] = x$$

L'algorithme échange $S_1[i_2] = S_1[2]$ et $S_1[j_2] = S_1[x]$ pour obtenir la permutation S_2 telle $S_2[2] = S_1[j_2] = x$ et $S_2[x] = S_2[j_2] = S_1[2] = 0$. Le deuxième octet de la suite chiffrante retourné par l'algorithme est donc

$$z_2 = S_2[S_2[2] + S_2[x] \bmod 256] = S_2[x] = 0$$

La figure suivante illustre ce raisonnement. Le triangle blanc indique la position de i et le triangle grisé la position de j .



- Pour une permutation aléatoire S_0 , la probabilité que $S_0[2] = 0$ et $S_0[1] \neq 2$ est égale à $(1/256) \cdot (254/256)$, et dans ce cas, le second octet généré par RC4 sera toujours égal à 0. Si l'on suppose que pour une permutation ne vérifiant pas cette hypothèse, la probabilité que $z_2 = 0$ est uniforme, nous obtenons

$$\begin{aligned} \Pr[z_2 = 0] &= \Pr[z_2 = 0 | S_0[2] = 0 \wedge S_0[1] \neq 2] \cdot \Pr[S_0[2] = 0 \wedge S_0[1] \neq 2] \\ &\quad + \Pr[z_2 = 0 | S_0[2] \neq 0 \vee S_0[1] = 2] \cdot \Pr[S_0[2] \neq 0 \vee S_0[1] = 2] \\ &= 1 \cdot \frac{255}{256^2} + \frac{1}{256} \frac{256^2 - 255}{256^2} \\ &\approx \frac{1}{128} \end{aligned}$$

qui est le double de la probabilité uniforme attendue.

- Supposons qu'un même texte clair est envoyé sous forme chiffrée à n utilisateurs en utilisant l'algorithme RC4 avec des clés différentes.

Pour $n/128$ chiffrés, en moyenne, la valeur du deuxième octet ne sera pas modifiée car nous aurons $z_2 = 0$ d'après la question précédente. Pour les autres chiffrés la valeur du deuxième octet sera modifiée par une autre valeur de z_2 (a priori uniformément distribuée parmi les 256 valeurs possibles). L'octet qui sera le plus fréquent parmi les deuxièmes octets de tous les chiffrés sera donc vraisemblablement le deuxième octet du texte clair.

Pour le chiffrement RC4, la génération de la permutation S à partir de la clé K est réalisée par l'algorithme 4.3. La longueur de la clé varie entre 16 et 256 bits. Elle est souvent choisie égale à $\ell = 16$ octets (128 bits).

Algorithme 4.3 Génération de la permutation initiale de RC4

ENTRÉE: $K \in \{0, \dots, 255\}^\ell, n \in \mathbb{N}$

SORTIE: $S \in \mathfrak{S}_{\{0, \dots, 255\}}$

```

pour  $i$  de 0 à 255 faire
   $S[i] \leftarrow i$ 
fin pour
 $j \leftarrow 0$ 
pour  $i$  de 0 à 255 faire
   $j \leftarrow (j + S[i] + K[i \bmod \ell]) \bmod 256$ 
   $S[i] \leftrightarrow S[j]$  ▷ échange des deux valeurs  $S[i]$  et  $S[j]$ 
fin pour
retourner  $S$  ▷ permutation initiale de RC4

```

La procédure de création de la permutation S à partir de la clé présente certaines faiblesses (notamment en raison de l'absence d'utilisation d'un vecteur d'initialisation séparé, en plus de la clé, qui rend le système totalement vulnérable à une attaque à un clair connu).

Une variante a été proposée en utilisant une clé k de 16 octets et un vecteur d'initialisation v de 16 octets. L'utilisateur construit le tableau K pour l'algorithme 4.3 en fixant les premiers octets aux valeurs du vecteur d'initialisation et les octets suivants aux valeurs des octets de la clé (répétés quinze fois). Autrement dit nous avons

$$K[i] = v[i] \text{ pour } 0 \leq i \leq 15 \text{ et } K[i] = k[i \bmod 16] \text{ pour } 16 \leq i \leq 255$$

L'exercice suivant montre que malgré ce choix, la procédure d'initialisation entraîne des failles de sécurité.

Problème 4.16 Attaque par recouvrement de clé sur RC4

Soit un entier ℓ vérifiant $16 \leq \ell \leq 32$ et soit $\kappa = K[\ell]$. Le but de cet exercice est de présenter un algorithme qui permet de retrouver κ en supposant connues les valeurs $K[i]$ pour $i < \ell$ (le vecteur d'initialisation v étant toujours supposé connu).

- Montrer que l'état de la permutation S et la valeur de j à la fin de la boucle d'indice $\ell - 1$ dans l'algorithme (4.3) peuvent être obtenus à partir des valeurs $K[i]$ pour $i < \ell$.

2. Supposons que $K[0] = \ell$ et $K[1] = 255$. Montrer que, si les valeurs de la permutation S aux positions 0, 1 et ℓ ne sont modifiées qu'une fois (*i.e.* aux tours d'indice 0, 1 et ℓ), alors nous avons

$$z_1 = S_{\ell-1}[j_{\ell-1} + S_{\ell-1}[\ell] + \kappa]$$

3. Proposer une estimation heuristique de la probabilité que trois octets d'indices α_1, α_2 et α_3 ne soient pas modifiés par les échanges des tours d'indice i pour $i > \max(\alpha_1, \alpha_2, \alpha_3)$.
4. Supposons qu'un attaquant dispose de textes clairs (dont les premiers octets sont connus) qui sont chiffrés de manière répétée avec la même clé et des vecteurs d'initialisation différents. Proposer une attaque qui révèle la clé à partir des chiffrés observés.

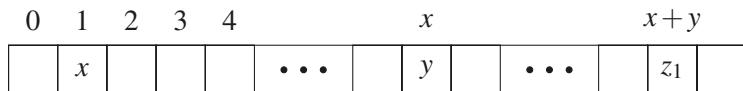
Solution

1. À partir des valeurs $K[i]$ pour $i < \ell$, un attaquant peut calculer récursivement les valeurs

$$j_t = (j_{t-1} + S[t] + K[t \bmod \ell]) \bmod 256$$

pour $t \in \{0, \dots, \ell\}$ et les valeurs de la permutation S correspondante.

2. La valeur du premier octet de la suite chiffrante z_1 dépend uniquement des valeurs $x = S[1]$ et $y = S[x]$ comme illustré sur le diagramme suivant :



Notons $S_j[t]$ et j_t les valeurs de la permutation S en l'octet i et de l'entier j après le t -ième tour de boucle de l'algorithme (4.3) et $S_0[t] = t$ pour tout $t \in \{0, \dots, 255\}$. Si $K[0] = \ell$ et $K[1] = 255$, nous avons pour les deux premiers tours :

$$j_1 = 0 + K[0] = \ell, S_1[\ell] = 0, S_1[0] = \ell \text{ et } S_1[t] = t \text{ pour } t \in \{0, \dots, 255\} \setminus \{0, \ell\}$$

et

$$\begin{aligned} j_2 &= \ell + S_1[1] + K[1] \bmod 256 = \ell + 1 + 255 \bmod 256 = \ell \\ S_2[\ell] &= S_1[1] = 1 \\ S_2[1] &= S_1[\ell] = 0 \end{aligned}$$

Si les valeurs de $S_2[1]$ et de $S_2[S_2[1]] = S_2[0]$ ne sont plus modifiées entre les tours 3 et $\ell - 1$, nous obtenons au ℓ -ième tour correspondant à $i = \ell$, et $S_\ell[j_\ell] = S_{\ell-1}[i] = S_{\ell-1}[\ell]$

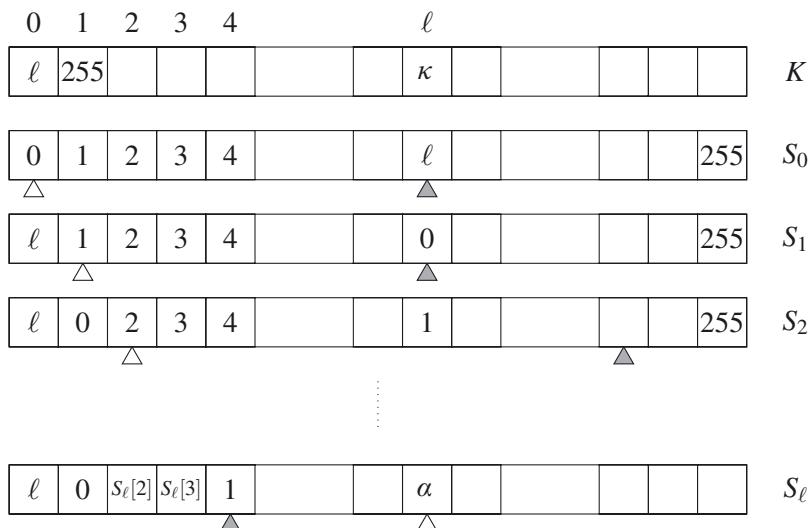
$$j_\ell = j_{\ell-1} + S_{\ell-1}[\ell] + K[\ell] = j_{\ell-1} + S_{\ell-1}[\ell] + \kappa$$

En posant $\alpha = S_{\ell-1}[j_\ell] = S_{\ell-1}[j_{\ell-1} + S_{\ell-1}[\ell] + \kappa]$, nous obtenons $S_\ell[\ell] = \alpha$.

Si les valeurs de $S_\ell[1] = 0$, $S_\ell[S_\ell[1]] = S_\ell[0]$ et $S_\ell[\ell]$ ne sont plus modifiées jusqu'à la fin de la génération de S , le premier octet de la suite chiffrante est bien égal à :

$$z_1 = S_\ell[S_\ell[1] + S_\ell[S_\ell[1]]] = S_\ell[0 + S_\ell[0]] = S_\ell[\ell] = \alpha = S_{\ell-1}[j_{\ell-1} + S_{\ell-1}[\ell] + \kappa]$$

La figure suivante illustre ce raisonnement. Le triangle blanc indique la position de i et le triangle grisé la position de j .



- La probabilité qu'un octet d'indice i reste inchangé après un échange aléatoire dans un tour $j > i$ est égale à $(1 - 1/256)$. En supposant que les tours se comportent de façon indépendante, la probabilité que l'octet reste inchangé après t échanges aléatoires dans des tours d'indice supérieur à i peut être estimée proche de $(1 - 1/256)^t$.

En faisant l'hypothèse simplificatrice que les trois octets sont indépendants, la probabilité que trois octets d'indice inférieur à j ne soient pas modifiés peut être estimée de l'ordre de $(1 - 1/256)^{3t}$. Dans le cas où t prend la valeur maximale 256 et en approximant $(1 - 1/256)^{256} \simeq e^{-1}$, nous obtenons que la probabilité que trois octets restent inchangés est de l'ordre de e^{-3} .

- D'après les questions précédentes, un attaquant peut obtenir les octets de la clé K successivement en utilisant la propriété du premier octet de la suite chiffrante.

Par exemple, pour obtenir le premier octet de la clé (*i.e.* $K[16]$), l'attaquant recherche parmi les textes clairs ceux utilisant un vecteur d'initialisation commençant par $K[0] = 16$ et $K[1] = 255$. Avec une probabilité proche de e^{-3} , l'octet z_1 est égal à

$$z_1 = S_{15}[j_{15} + S_{15}[16] + K[16]]$$

D'après la question 1, l'attaquant peut calculer les valeurs de $S_{15}[16]$, de j_{15} et de $S_{15}^{-1}[z_1]$ et une valeur candidate pour $K[16]$ (à partir du vecteur d'initialisation v). En répétant cette étape plusieurs fois, l'attaquant peut en déduire avec une très forte probabilité la valeur effective de $K[16]$.

Enfin, en itérant l'attaque pour tous les octets de clé $K[i]$ pour $i \in \{17, \dots, 32\}$, l'attaquant obtient la clé complète utilisée pour le chiffrement.

Note

Cette attaque a été présentée par S. FLUHRER, I. MANTIN et A. SHAMIR à la conférence *Selected Areas in Cryptography 2001* [24] et sert de base à l'attaque contre le protocole WEP.

PROBLÈME DU LOGARITHME DISCRET

5

Les concepts fondamentaux de la cryptographie asymétrique remontent à l'article fondateur de W. DIFFIE et M. HELLMAN paru en 1976, et RSA, le premier cryptosystème à clé publique, est apparu deux ans plus tard dans celui de R. L. RIVEST, A. SHAMIR et L. M. ADLEMAN. La cryptographie asymétrique repose de façon essentielle sur la notion de *fonction à sens unique*. Il s'agit d'une fonction facile à calculer mais pour laquelle il est impossible de calculer l'inverse en temps polynomial. Démontrer l'existence de fonctions à sens unique est un problème ouvert difficile mais la théorie des nombres est une source fascinante de problèmes algorithmiques qui peuvent être utilisés pour construire des fonctions (supposées) à sens unique.

Les protocoles cryptographiques à clé publique présentés dans la deuxième partie de ce livre reposent tous sur la difficulté du problème du logarithme discret dans un groupe abélien fini ou sur celle du problème de la factorisation des entiers. Dans ce chapitre, nous allons étudier les aspects algorithmiques et théoriques du problème du logarithme discret dans un groupe abélien fini.

Nous analyserons plusieurs algorithmes de complexité exponentielle qui fonctionnent dans n'importe quel groupe abélien (*i.e.* l'*algorithme de Shanks*, la *méthode ρ de Pollard*, l'*algorithme de Pohlig-Hellman* et la *méthode λ de Pollard*). Nous étudierons ensuite un algorithme sous-exponentiel (par *calcul d'indice*) pour résoudre le problème du logarithme discret dans le sous-groupe multiplicatif d'un corps fini d'ordre premier. Nous examinerons également des méthodes de résolution de variantes du problème du logarithme discret. Enfin, dans une dernière partie plus théorique, nous analyserons l'*interpolation polynomiale* du problème du logarithme discret dans le sous-groupe multiplicatif d'un corps fini d'ordre premier.

5.1 LOGARITHME DISCRET DANS UN GROUPE GÉNÉRIQUE

Une opération particulièrement importante en cryptographie à clé publique est l'*exponentiation discrète* dans un groupe abélien fini \mathbb{G} (généralement noté multiplicativement). Elle consiste, étant donné un élément g de \mathbb{G} et un entier $n \in \mathbb{N}$, à

calculer l'élément g^n de \mathbb{G} . Calculer cet élément demande *a priori* n multiplications dans le groupe \mathbb{G} mais on remarque facilement qu'il peut être obtenu en seulement $O(\log n)$ multiplications en décomposant l'entier n en base 2. En effet, si

$$n = \sum_{i=0}^{\ell-1} a_i 2^i \in \mathbb{N} \text{ avec } a_i \in \{0, 1\} \text{ pour } i \in \{0, \dots, \ell - 1\}$$

où ℓ est la taille en bits de l'entier n , nous avons

$$\begin{aligned} g^n &= \prod_{i=0}^{\ell-1} g^{a_i 2^i} = g^{a_0} (g^{a_1})^2 (g^{a_2})^4 (g^{a_3})^8 \dots (g^{a_{\ell-1}})^{2^{\ell-1}} \\ &= g^{a_0} \left(g^{a_1} \left(g^{a_2} \left(g^{a_3} \dots (g^{a_{\ell-1}})^2 \right)^2 \right)^2 \right)^2 \end{aligned}$$

Cette dernière expression peut être calculée en $\ell - 1$ élévarions au carré entrelacées avec des multiplications par g^{a_i} pour $i \in \{0, \dots, \ell - 2\}$. En moyenne, le nombre de a_i non nuls pour $i \in \{0, \dots, \ell - 2\}$ est égal à $(\ell - 1)/2$. Le nombre de multiplications nécessaires pour calculer g^n est donc égal à $3(\ell - 1)/2$ en moyenne et à $2(\ell - 1)$ dans le pire des cas. L'algorithme qui effectue ce calcul (*cf.* Algorithme 5.1) est appelé algorithme d'*exponentiation dichotomique* ou algorithme « éléver au carré et multiplier » (*square and multiply* en anglais).

Algorithme 5.1 Exponentiation discrète dichotomique

ENTRÉE: $g \in \mathbb{G}$, $n = \sum_{i=0}^{\ell-1} a_i 2^i \in \mathbb{N}$ avec $a_i \in \{0, 1\}$ pour $i \in \{0, \dots, \ell - 1\}$

SORTIE: $g^n \in \mathbb{G}$

```

 $h \leftarrow 1_{\mathbb{G}}$ 
pour  $i$  de  $\ell - 1$  à 0 faire
     $h \leftarrow h^2$                                  $\triangleright$  l'indice  $i$  varie en décroissant
    si  $a_i = 1$  alors
         $h \leftarrow h \cdot g$ 
    fin si
fin pour
retourner  $h$ 

```

Il existe de nombreuses variantes de cet algorithme en fonction du contexte et du groupe \mathbb{G} considéré. L'exercice suivant montre, par exemple, qu'il est généralisable pour calculer un produit de la forme $g_1^{n_1} \dots g_t^{n_t}$ dans \mathbb{G} (presque aussi efficacement qu'une seule exponentiation).

Exercice 5.1 Multi-exponentiation

Soit \mathbb{G} un groupe abélien (noté multiplicativement). Proposer un algorithme qui étant donnés t éléments g_1, \dots, g_t du groupe \mathbb{G} et des entiers positifs n_1, \dots, n_t calcule le produit $g_1^{n_1} \dots g_t^{n_t} \in \mathbb{G}$ en $O(\ell + 2^t)$ multiplications dans \mathbb{G} (où ℓ est la taille en bits de $\max(n_1, \dots, n_t)$).

Solution

Posons pour $j \in \{1, \dots, t\}$

$$n_j = \sum_{i=0}^{\ell-1} a_{i,j} 2^i \in \mathbb{N} \text{ avec } a_{i,j} \in \{0, 1\} \text{ pour } i \in \{0, \dots, \ell-1\}$$

Considérons tout d'abord le cas $t = 2$ et remarquons que

$$\begin{aligned} g_1^{n_1} g_2^{n_2} &= \prod_{i=0}^{\ell-1} g_1^{a_{i,1} 2^i} \prod_{i=0}^{\ell-1} g_2^{a_{i,2} 2^i} = \prod_{i=0}^{\ell-1} (g_1^{a_{i,1}} g_2^{a_{i,2}})^{2^i} \\ &= (g_1^{a_{0,1}} g_2^{a_{0,2}}) (g_1^{a_{1,1}} g_2^{a_{1,2}})^2 (g_1^{a_{2,1}} g_2^{a_{2,2}})^4 \dots (g_1^{a_{\ell-1,1}} g_2^{a_{\ell-1,2}})^{2^\ell} \\ &= (g_1^{a_{0,1}} g_2^{a_{0,2}}) \left(g_1^{a_{1,1}} g_2^{a_{1,2}} \left(g_1^{a_{2,1}} g_2^{a_{2,2}} \dots (g_1^{a_{\ell-1,1}} g_2^{a_{\ell-1,2}})^{2^2} \right)^2 \right)^2 \end{aligned}$$

Comme pour l'exponentiation dichotomique, cette dernière expression peut être calculée en $\ell - 1$ élévarions au carré entrelacées avec des multiplications par les éléments $g_1^{a_{i,1}} g_2^{a_{i,2}}$ pour $i \in \{0, \dots, \ell - 2\}$. Nous avons $a_{i,1} \in \{0, 1\}$ et $a_{i,2} \in \{0, 1\}$ pour tout $i \in \{0, \dots, \ell - 1\}$, donc le nombre d'éléments différents de la forme $g_1^{a_{i,1}} g_2^{a_{i,2}}$ pour $i \in \{0, \dots, \ell - 1\}$ est majoré par 4 :

$$1_{\mathbb{G}} = g_1^0 g_2^0, \quad g_1 = g_1^1 g_2^0, \quad g_2 = g_1^0 g_2^1 \quad \text{et} \quad g_{1,2} = g_1^1 g_2^1 = g_1 g_2$$

En pré-calculant cette dernière valeur (et en la stockant), le nombre de multiplications nécessaires pour calculer $g_1^{n_1} g_2^{n_2}$ est alors égal à $2(\ell - 1) + 1$ dans le pire des cas et à $7(\ell - 1)/4 + 1$ en moyenne (la multiplication par l'élément $1_{\mathbb{G}}$ étant réalisée une fois sur quatre en moyenne).

Pour généraliser cette approche pour un entier t arbitraire, il suffit de pré-calculer les valeurs

$$g_E = g_1^{b_1} \dots g_n^{b_n} \text{ avec } \begin{cases} b_i = 1 & \text{si } i \in E \\ b_i = 0 & \text{sinon} \end{cases}$$

pour tous les sous-ensembles E de $\{1, \dots, t\}$. En particulier $g_\emptyset = 1_{\mathbb{G}}$ et $g_{\{i\}} = g_i$ pour tout $i \in \{1, \dots, t\}$. Ce pré-calcul demande 2^t multiplications dans \mathbb{G} en construisant les g_E récursivement (*i.e.* en utilisant la formule $g_{E \cup \{i\}} = g_E \cdot g_i$ pour $i \notin E$).

Une fois ce pré-calcul réalisé, l'algorithme est identique à l'algorithme d'exponentiation dichotomique. Nous obtenons finalement l'algorithme 5.2 qui nécessite $2(\ell - 1) + 2^t$ multiplications dans \mathbb{G} dans le pire des cas.

Algorithme 5.2 Multi-exponentiation discrète dichotomique

ENTRÉE: $g_1, \dots, g_t \in \mathbb{G}$, $n_1, \dots, n_t \in \mathbb{N}$ avec $n_j = \sum_{i=0}^{\ell-1} a_{i,j} 2^i \in \mathbb{N}$ avec $a_{i,j} \in \{0, 1\}$ pour $i \in \{0, \dots, \ell - 1\}$ et $j \in \{1, \dots, t\}$

SORTIE: $g_1^{n_1} \dots g_t^{n_t} \in \mathbb{G}$

```

 $g_0 \leftarrow 1_{\mathbb{G}}$ 
pour  $j$  de 1 à  $t$  faire
    pour  $E \subseteq \{1, \dots, j - 1\}$  faire
         $g_{E \cup \{j\}} \leftarrow g_E \cdot g_j$ 
    fin pour
fin pour
 $h \leftarrow 1_{\mathbb{G}}$ 
pour  $i$  de  $\ell - 1$  à 0 faire
     $E_i \leftarrow \emptyset$ 
    pour  $j$  de 1 à  $t$  faire
        si  $a_{i,j} = 1$  alors
             $E_i \leftarrow E_i \cup \{j\}$ 
        fin si
    fin pour
     $h \leftarrow h^2$ 
     $h \leftarrow h \cdot g_{E_i}$ 
fin pour
retourner  $h$ 

```

▷ fin du précalcul

Note

Pour des compléments sur les algorithmes de multi-exponentiation, nous renvoyons le lecteur au survol très complet de R. M AVANZI [4].

Étant donné un groupe abélien \mathbb{G} , un élément $g \in \mathbb{G}$ et un élément h appartenant au sous-groupe multiplicatif engendré par g (noté $\langle g \rangle = \{g^i, i \in \mathbb{Z}\}$), le problème qui consiste à retrouver le plus petit entier positif x tel que $h = g^x$ est appelé *problème du logarithme discret*. En faisant une recherche exhaustive de cette valeur n , un algorithme peut retrouver ce logarithme discret $x = \log_g(h)$ en x multiplications dans le groupe \mathbb{G} . L'exercice suivant améliore cet algorithme trivial et montre comment il est possible de retrouver n en $O(\sqrt{|\mathbb{G}|})$ multiplications dans \mathbb{G} .

Exercice 5.2 Algorithme de Shanks

Considérons un groupe multiplicatif cyclique \mathbb{G} engendré par $g \in \mathbb{G}$ d'ordre connu q . Proposer un algorithme de résolution de logarithme discret par compromis temps-mémoire de complexité $O(\sqrt{q})$ opérations de groupe en temps et $O(\sqrt{q})$ éléments de groupe en mémoire.

Indication

On pourra remarquer que pour tout élément $h = g^x \in \langle g \rangle$, l'entier x s'écrit sous la forme $x = x_1 T + x_0$ avec $0 \leq x_1 < T$ et $0 \leq x_2 < T$ pour $T = \lceil \sqrt{q} \rceil + 1$.

Solution

Suivant l'indication, nous remarquons que l'égalité $h = g^x$ avec $x = x_1 T + x_0$ s'écrit également

$$h \cdot (g^{-T})^{x_1} = hg^{-x_1 T} = g^{x_0}$$

En appliquant un compromis temps-mémoire, il suffit dans un premier temps de calculer les T valeurs prises par le terme de droite de cette égalité (*i.e.* les éléments g^i pour $i \in \{0, \dots, T-1\}$) et de stocker les valeurs obtenues (g^i, i) dans une table de hachage indexée par les éléments de groupe.

Dans un second temps, l'algorithme va rechercher parmi les valeurs $h \cdot (g^{-T})^j$ pour $j \in \{0, \dots, T-1\}$ si l'une d'elles apparaît dans la table de hachage.

Le nombre de multiplications à effectuer est égal à :

- T pour la construction de la table des éléments g^i pour $i \in \{0, \dots, T-1\}$;
- $2(\lfloor \log(q-T) \rfloor + 1)$ pour le calcul de g^{-T} (puisque $g^{q-T} \cdot g^T = g^q = 1_{\mathbb{G}}$) ;
- T pour la recherche des éléments $h \cdot (g^{-T})^j$ pour $j \in \{0, \dots, T-1\}$.

soit un total de $2(T + \lfloor \log(q-T) \rfloor + 1) = O(T) = O(\sqrt{q})$. La complexité en mémoire est donnée par la table de hachage qui utilise $O(T) = O(\sqrt{q})$ éléments du groupe \mathbb{G} .

Algorithme 5.3 Algorithme de Shanks (ou « pas de bébé, pas de géant »)

ENTRÉE: $g, h \in \mathbb{G}, q = |\mathbb{G}|$

SORTIE: $x \in \{0, \dots, q-1\}$ tel que $h = g^x$

$T \leftarrow \lceil \sqrt{q} \rceil + 1$

$\Upsilon \leftarrow \emptyset$

pour i de 0 à $T-1$ **faire**

$x_i \leftarrow g^i$

$\Upsilon \leftarrow \Upsilon \cup (x_i, i)$

 ▷ 1 multiplication à chaque itération ($x_{i+1} \leftarrow x_i \cdot g$)

 ▷ table de hachage

fin pour

$k \leftarrow g^{q-T}$

 ▷ $O(\log(q-T))$ multiplications

pour j de 0 à $T-1$ **faire**

$y \leftarrow h \cdot k^j$

 ▷ 1 multiplication à chaque itération ($y \leftarrow y \cdot k$)

si $y = x_i$ avec $(x_i, i) \in \Upsilon$ **alors**

retourner $i + Tj \bmod q$

fin si

fin pour

Note

L'algorithme obtenu (cf. Algorithme 5.3) a été proposé par D. SHANKS en 1970 [59]. Il est appelé l'algorithme « *pas de bébé, pas de géant* » (la construction des valeurs g^i pour $i \in \{0, \dots, T-1\}$ étant les « *pas de bébé* », celle des $(g^{-T})^j$ pour $j \in \{0, \dots, T-1\}$ les « *pas de géant* »).

L'inconvénient majeur de l'algorithme précédent est sa complexité en mémoire. En 1978, J. M. POLLARD a proposé un algorithme probabiliste dont la complexité en multiplications dans \mathbb{G} est similaire mais qui ne requiert le stockage que d'un nombre constant d'éléments de \mathbb{G} [54].

Exercice 5.3 Algorithme ρ de Pollard

Considérons un groupe multiplicatif cyclique \mathbb{G} engendré par $g \in \mathbb{G}$ d'ordre premier connu q et soit h un élément de \mathbb{G} .

Soit $F : \mathbb{G} \rightarrow (\mathbb{Z}/q\mathbb{Z})$. Nous définissons une fonction $H : \mathbb{G} \rightarrow \mathbb{G}$ par $H(\alpha) = \alpha \cdot h \cdot g^{F(\alpha)}$ et l'algorithme 5.4 (appelé *algorithme ρ de Pollard*).

Algorithme 5.4 Algorithme ρ de Pollard (pour le logarithme discret)

ENTRÉE: $g, h \in \mathbb{G}$

SORTIE: $x \in \{0, \dots, q - 1\}$ tel que $h = g^x$ ou ÉCHEC

```

1:  $i \leftarrow 1$ 
2:  $x \leftarrow 0 ; \alpha \leftarrow h$ 
3:  $y \leftarrow F(\alpha) ; \beta \leftarrow H(\alpha)$ 
4: tant que  $\alpha \neq \beta$  faire
5:    $x \leftarrow x + F(\alpha) \text{ mod } q ; \alpha \leftarrow H(\alpha)$ 
6:    $y \leftarrow y + F(\beta) \text{ mod } q ; \beta \leftarrow H(\beta)$ 
7:    $y \leftarrow y + F(\beta) \text{ mod } q ; \beta \leftarrow H(\beta)$ 
8:    $i \leftarrow i + 1$ 
9: fin tant que
10: si  $i < q$  alors
11:   retourner  $(x - y)/i \text{ mod } q$ 
12: sinon
13:   retourner ÉCHEC
14: fin si
```

Considérons la suite $(\gamma_i)_{i \geq 1}$ définie par récurrence par $\gamma_1 = h$ et $\gamma_{i+1} = H(\gamma_i)$ pour $i \geq 1$.

- Montrer que dans la boucle **tant que** des lignes 4 à 9 de l'algorithme 5.4, nous avons

$$\alpha = \gamma_i = g^x h^i \text{ et } \beta = \gamma_{2i} = g^y h^{2i}$$

- Montrer que si cette boucle termine avec $i < q$ alors l'algorithme retourne le logarithme discret de h en base g .
- Soit j le plus petit entier tel que $\gamma_j = \gamma_k$ pour un entier $k < j$. Montrer que $j \leq q + 1$ et que la boucle termine avec $i < j$.
- Montrer que si F est une fonction aléatoire, alors le temps moyen d'exécution de l'algorithme est en $O(q^{1/2})$ multiplications dans \mathbb{G} .

Solution

- Il suffit de vérifier par récurrence que

$$\alpha = \gamma_i = g^x h^i \text{ et } \beta = \gamma_{2i} = g^y h^{2i} \quad (5.1)$$

pour chaque valeur de i .

Lorsque $i = 1$, nous avons $x = 0$, $\alpha = h$, $y = F(h)$ et $\beta = H(h) = g^y h^2$ (*cf.* les lignes 1 à 3 de l'algorithme 5.4) et les égalités (5.1) sont bien vérifiées.

Notons x_i , α_i , y_i , β_i les valeurs prises par x , α , y et β dans la boucle principale de l'algorithme 5.4 (en fonction de la valeur de i). Supposons que les égalités (5.1) sont vérifiées pour une valeur $i \geq 1$. En appliquant la boucle principale de l'algorithme ρ de Pollard (*i.e.* les lignes 4 à 9), nous obtenons $x_{i+1} \equiv x_i + F(\alpha_i) \pmod{q}$ et

$$\begin{aligned} \alpha_{i+1} &= H(\alpha_i) = \alpha_i \cdot h \cdot g^{F(\alpha_i)} = (g^{x_i} \cdot h^i) \cdot h \cdot g^{F(\alpha_i)} = g^{x_i+F(\alpha_i)} \cdot h^{i+1} = g^{x_{i+1}} \cdot h^{i+1} \\ &= H(\alpha_i) = H(\gamma_i) = \gamma_{i+1} \end{aligned}$$

Nous avons également

$$\begin{aligned} \beta_{i+1} &= H(H(\beta_i)) = H(\beta_i \cdot h \cdot g^{F(\beta_i)}) = \beta_i \cdot h^2 \cdot g^{F(\beta_i)} \cdot g^{F(H(\beta_i))} \\ &= g^{y_i} \cdot g^{F(\beta_i)} \cdot g^{F(H(\beta_i))} \cdot h^{2i+2} \\ y_{i+1} &= y_i + F(\beta_i) + F(H(\beta_i)) \\ \text{et } \beta_{i+1} &= H(H(\beta_i)) = H(H(\gamma_{2i})) = H(\gamma_{2i+1}) = \gamma_{2i+2} \end{aligned}$$

d'où le résultat.

- Lorsque la boucle de l'algorithme termine avec $i < q$, nous avons

$$g^x h^i = \alpha = \beta = g^y h^{2i} \quad \text{soit} \quad h^i = g^{x-y}$$

Puisque q est premier, la condition $i < q$ assure que i est premier avec q et donc inversible dans $(\mathbb{Z}/q\mathbb{Z})$. En posant $u = (x - y)/i \pmod{q}$, nous obtenons bien $h = g^u$.

- La suite $(\gamma_n)_{n \in \mathbb{N}}$ est à valeurs dans le groupe \mathbb{G} qui est un ensemble fini de cardinal q . Par le principe des tiroirs, il existe donc un indice $k < j$ pour lequel $\gamma_j = \gamma_k$ avec $j < q + 1$.

La suite $(\gamma_n)_{n \in \mathbb{N}}$ étant définie par la récurrence $\gamma_{n+1} = H(\gamma_n)$ pour tout entier n , nous avons $\gamma_{j+t} = \gamma_{k+t}$ pour tout entier $t \geq 0$ et la suite des valeurs de $(\gamma_n)_{n \in \mathbb{N}}$ peut être représentée comme sur la figure 5.1.

L'algorithme ρ de Pollard calcule simultanément les valeurs $(\gamma_n)_{n \in \mathbb{N}}$ et $(\gamma_{2n})_{n \in \mathbb{N}}$ et retourne la plus petite valeur i telle que $\gamma_i = \gamma_{2i}$.

Puisque nous avons $\gamma_{j+t} = \gamma_{k+t}$ pour tout entier $t \geq 0$, il suffit de trouver une valeur t telle que $j + t = 2(k + t)$. La valeur $t = j - 2k$ convient et nous donne une valeur de i égale à $k + (j - 2k) = j - k$ qui appartient bien à l'ensemble $\{1, \dots, j - 1\}$.

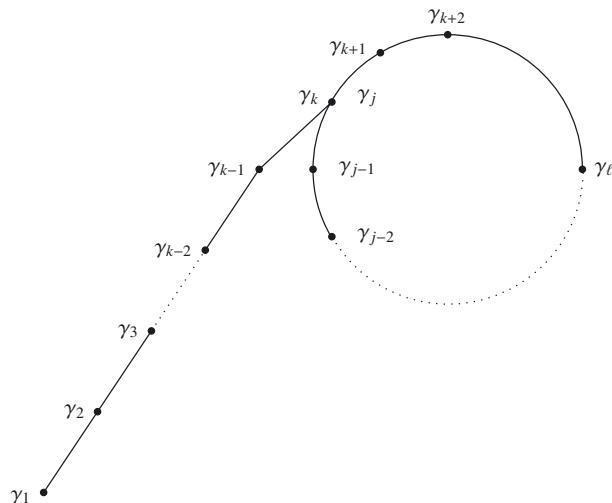


Figure 5.1– Représentation de la suite $(\gamma_n)_{n \in \mathbb{N}}$ dans l'algorithme ρ de Pollard

4. Si la fonction F est une fonction aléatoire de \mathbb{G} , la fonction H est une fonction aléatoire de \mathbb{G} dans \mathbb{G} et le nombre espéré d'éléments de la suite $(\gamma_n)_{n \in \mathbb{N}}$ pour obtenir deux valeurs communes est de l'ordre de $j \simeq \sqrt{\pi q}/2$ (d'après le paradoxe des anniversaires). Comme chaque étape de la boucle de l'algorithme ne demande qu'un nombre constant de multiplications dans le groupe, nous obtenons bien le résultat demandé.

Note

Le nom de l'algorithme ρ de Pollard vient de la forme de la figure 5.1. Pour obtenir de bons résultats avec cet algorithme, il faut que la fonction F se comporte comme une fonction aléatoire. Une méthode possible pour implanter cette fonction F est de partitionner le groupe \mathbb{G} en un nombre fini de sous-ensembles G_1, \dots, G_n (indépendamment de la structure de groupe) est de définir $F(y) = i$ si $y \in G_i$ (pour $i \in \{1, \dots, n\}$).

La technique utilisée pour détecter la collision dans la suite $(\gamma_i)_{i \geq 1}$ a été proposée par R. FLOYD en 1967 [23]. Elle trouve d'autres applications en cryptographie (notamment la recherche de collisions pour les fonctions de hachage ou la méthode ρ de Pollard pour la factorisation que nous verrons au chapitre suivant).

L'algorithme ρ de Pollard présenté résout le problème du logarithme discret dans un groupe d'ordre premier. L'exercice suivant montre que ce n'est pas vraiment une restriction. En effet, nous allons voir que dans un groupe \mathbb{G} d'ordre n dont la décomposition en facteurs premiers est connue, il est possible de réduire le problème du logarithme discret dans \mathbb{G} au problème du logarithme discret dans chacun de ses sous-groupes d'ordre premier.

Exercice 5.4 Algorithme de Pohlig-Hellman

Soit \mathbb{G} un groupe cyclique d'ordre fini n dont la décomposition en facteurs premiers est connue.

- Montrer que si $n = pq$ est le produit de nombres premiers distincts alors il existe un algorithme qui résout le problème du logarithme discret dans \mathbb{G} en $O(\sqrt{p} + \sqrt{q} + \log(n))$ multiplications dans \mathbb{G} .
- Montrer que si $n = p^e$ où p est un nombre premier et $e \geq 2$ est un entier alors il existe un algorithme qui résout le problème du logarithme discret dans \mathbb{G} en $O(e(\sqrt{p} + \log(n)))$ multiplications dans \mathbb{G} .
- En déduire que si $n = q_1^{e_1} \cdots q_k^{e_k}$ où les q_i sont des nombres premiers deux à deux distincts), alors il existe un algorithme qui résout le problème du logarithme discret en $O\left(\sum_{i=1}^k e_i(\sqrt{q_i} + \log(n))\right)$ opérations dans le groupe \mathbb{G} .

Solution

- Soient g et h deux éléments de \mathbb{G} avec $g \neq 1_{\mathbb{G}}$. Le logarithme discret de h en base g est l'élément x de $(\mathbb{Z}/n\mathbb{Z})$ tel que $h = g^x$. Si la factorisation de $n = pq$ est connue et si les valeurs de $(x \bmod p)$ et $(x \bmod q)$ sont connues, il est facile de reconstruire x par le théorème chinois des restes.
En posant $x = x_0p + x_1$ avec $x_0, x_1 \in \mathbb{N}$ et $x_1 < p$, nous avons $x_1 \equiv x \bmod p$ et pour obtenir x_1 il suffit de remarquer que $h^q = (g^x)^q = (g^q)^x$ et que l'élément g^q est d'ordre p . Nous avons alors

$$h^q = (g^q)^{x_0p+x_1} = g^{x_0pq+qx_1} = (g^{pq})^{x_0} \cdot (g^q)^{x_1} = (g^q)^{x_1}$$

puisque $g^{pq} = g^n = 1_{\mathbb{G}}$.

L'entier x_1 est donc le logarithme discret de h^q en base g^q . En appliquant l'un des algorithmes de résolution de logarithme discret vu précédemment, il est ensuite possible de retrouver x_1 en $O(\sqrt{p})$ multiplications dans \mathbb{G} . Le coût total pour obtenir $(x \bmod p)$ est donc $O(\sqrt{p} + \log(q))$ multiplications dans \mathbb{G} (le terme $O(\log(q))$ provient des deux exponentiations discrètes nécessaires pour le calcul de h^q et g^q).

De même, il est possible de retrouver la valeur de $(x \bmod q)$ en $O(\sqrt{q} + \log(p))$ multiplications dans \mathbb{G} et après application du théorème chinois des restes, l'algorithme retourne x en $O(\sqrt{p} + \sqrt{q} + \log(n))$ multiplications dans \mathbb{G} .

- Notons encore x le logarithme discret de h en base g avec $g, h \in \mathbb{G}$. En décomposant x en base p , nous avons

$$x = x_0 + x_1p + \cdots + x_{e-1}p^{e-1} \text{ avec } x_i \in \{0, \dots, p-1\} \text{ pour } i \in \{0, \dots, e-1\}.$$

L'approche est similaire à celle de la question précédente mais en calculant cette fois les valeurs x_i successivement pour $i \in \{0, \dots, e-1\}$. Pour obtenir la valeur de $x_0 = (x \bmod p)$, l'algorithme calcule le logarithme discret de $h^{p^{e-1}}$ en base $g^{p^{e-1}}$. Cet élément engendre un sous-groupe d'ordre p et le calcul de x_0 demande $O(\sqrt{p} + \log(n))$ multiplications dans le groupe \mathbb{G} .

Pour déterminer la valeur de x_1 , remarquons que

$$hg^{-x_0} = g^{x_1 p + \dots + x_{e-1} p^{e-1}} = (g^p)^{x_1 + \dots + x_{e-1} p^{e-2}}$$

En élevant cette égalité à la puissance p^{e-2} , nous obtenons $(hg^{-x_0})^{p^{e-2}} = (g^{p^{e-1}})^{x_1}$ et la valeur de x_1 s'obtient donc comme la valeur du logarithme discret de $(hg^{-x_0})^{p^{e-2}}$ en base $g^{p^{e-1}}$ (ce qui demande de nouveau $O(\sqrt{p} + \log(n))$ multiplications dans le groupe \mathbb{G}).

En itérant cette approche, nous voyons que le logarithme discret de l'élément

$$(hg^{-(x_0 + px_1 + \dots + p^{e-1}x_{e-1})})^{p^{e-\ell-1}}$$

en base $g^{p^{e-1}}$ est égal à x_ℓ pour tout $\ell \in \{2, \dots, e-1\}$. Nous pouvons donc construire un algorithme qui calcule la valeur de x en $O(e(\sqrt{p} + \log(n)))$ multiplications dans le groupe \mathbb{G} .

3. Il suffit de combiner les méthodes des deux questions précédentes pour calculer la valeur de x modulo les $q_i^{e_i}$ pour $i \in \{1, \dots, k\}$ puis de reconstituer la valeur de x en appliquant le théorème chinois des restes.

Notons $\text{LOGARITHMEDISCRET}(h, g)$ le logarithme discret de h en base g (calculé par exemple par l'algorithme de Shanks 5.3 ou l'algorithme ρ de Pollard 5.4) et $\text{RESTESCHINOIS}((x_1, n_1), (x_2, n_2), \dots, (x_t, n_t))$ la valeur $x \in (\mathbb{Z}/n\mathbb{Z})$ telle que $x \equiv x_i \pmod{n_i}$ pour $i \in \{1, \dots, t\}$ et $n = n_1 n_2 \dots n_t$. L'algorithme 5.5 résout le problème du logarithme discret en $O\left(\sum_{i=1}^k e_i(\sqrt{q_i} + \log(n))\right)$ opérations dans le groupe \mathbb{G} .

Algorithme 5.5 Algorithme de Pohlig-Hellman

ENTRÉE: $g, h \in \mathbb{G}$ et $(q_1, e_1, \dots, q_k, e_k)$ tel que $|\mathbb{G}| = q_1^{e_1} \dots q_k^{e_k}$

SORTIE: $x \in \{0, \dots, n-1\}$ tel que $h = g^x$

- 1: **pour** i de 1 à k **faire**
 - 2: $x_i \leftarrow 0$
 - 3: $g_i \leftarrow g^{n/q_i}$
 - 4: **pour** j de 1 à $e_i - 1$ **faire**
 - 5: $\alpha \leftarrow (h \cdot g^{-x_i})^{q_i^{e_i-j}}$
 - 6: $t \leftarrow \text{LOGARITHMEDISCRET}(\alpha, g_i)$
 - 7: $x_i \leftarrow x_i + t \cdot p^{j-1}$
 - 8: **fin pour**
 - 9: **fin pour**
 - 10: $x \leftarrow \text{RESTESCHINOIS}((x_1, q_1^{e_1}), (x_2, q_1^{e_2}), \dots, (x_k, q_1^{e_k}))$
 - 11: **retourner** x
-

Note

L'algorithme précédent (*cf.* Algorithme 5.5) a été proposé en 1978 par S. POHLIG et M. HELLMAN [51].

L'algorithme précédent montre en particulier que si l'ordre d'un groupe \mathbb{G} n'a que des « petits » facteurs premiers alors le problème du logarithme discret est facile à résoudre dans \mathbb{G} . L'exercice suivant donne un exemple d'un tel groupe comme groupe multiplicatif d'un corps fini $(\mathbb{Z}/p\mathbb{Z})^*$ engendré par l'élément 2.

Exercice 5.5 (avec programmation).

Application de l'algorithme de Pohlig-Hellman

Appliquer l'algorithme de Pohlig-Hellman pour calculer le logarithme discret de h en base g dans $(\mathbb{Z}/p\mathbb{Z})^*$ où

$$p = 27150900575903233409044168443612036065960862633229685601064230$$

$$g = 2$$

$$h = 12001254138198740617926080580880036900575413150567106504361875$$

Solution

Le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ est d'ordre

$$p - 1 = 2 \cdot 1057149 \cdot 1060743 \cdot 1307755 \cdot 1799693 \cdot 1886999^4 \cdot 1947545 \cdot 2083094.$$

Notons x le logarithme discret de h en base g . En appliquant l'algorithme de Shanks pour calculer la valeur de x modulo chaque nombre premier qui divise $p - 1$ avec multiplicité 1, nous trouvons

$$\begin{aligned} x &\equiv 0 \pmod{2} \\ x &\equiv 687885 \pmod{1057149} \\ x &\equiv 580752 \pmod{1060743} \\ x &\equiv 425323 \pmod{1307755} \\ x &\equiv 227959 \pmod{1799693} \\ x &\equiv 1939788 \pmod{1947545} \\ x &\equiv 555174 \pmod{2083094} \end{aligned}$$

En calculant la valeur de x modulo les puissances de 1886999, nous obtenons successivement

$$\begin{aligned} x &\equiv 364822 \pmod{1886999} \\ x &\equiv 1518236359245 \pmod{1886999^2} \\ x &\equiv 5033773420633774922 \pmod{1886999^3} \\ x &\equiv 9217980899967389466337028 \pmod{1886999^4} \end{aligned}$$

et finalement, en appliquant le théorème chinois des restes, nous obtenons la valeur de x suivante :

$$x = 74725294124582391378954045445229610641904889036329056996872188$$

5.2 PROBLÈME DU LOGARITHME DISCRET DANS $(\mathbb{Z}/p\mathbb{Z})^*$

Dans l'article fondateur de la cryptographie à clé publique, W. DIFFIE et M. HELLMAN ont suggéré d'utiliser la difficulté supposée du problème du logarithme discret dans certains groupes pour créer différentes primitives cryptographiques. Ils ont proposé d'utiliser le groupe multiplicatif d'un corps fini $(\mathbb{Z}/p\mathbb{Z})^*$ où p est un nombre premier. Dans cette section, nous allons voir qu'il existe des algorithmes sous-exponentiels pour résoudre le problème du logarithme discret dans ces groupes. Ces algorithmes appartiennent à la famille des algorithmes dits de *calcul d'indice* qui reposent sur l'existence d'éléments *friables* (*smooth*, en anglais) dans le groupe considéré.

Définition 5.1 Entiers friables

Soit $M \geq 0$ un nombre réel positif. Un entier $n \in \mathbb{N}$ est dit *M-friable* si tous les diviseurs premiers de n sont inférieurs à M .

La complexité des algorithmes de logarithme discret dans $(\mathbb{Z}/p\mathbb{Z})^*$ repose de façon essentielle sur le nombre d'entiers M -friables inférieurs à p pour une valeur de M bien choisie.

Définition 5.2 Fonction de Dickman-De Bruijn

Soient x, y deux nombres réels positifs. Notons

$$\Psi(x, y) = \#\{n \in \mathbb{N}, n \leq x \text{ et } n \text{ est } y\text{-friable}\}.$$

La fonction $\Psi : \mathbb{R}_+ \times \mathbb{R}_+ \longrightarrow \mathbb{N}$ est appelée *fonction de Dickman-De Bruijn*.

Il existe des estimations très fines de cette fonction suivant les valeurs de x et y . Dans l'exercice suivant, nous allons en démontrer une par des méthodes élémentaires en admettant le théorème des nombres premiers (où pour tout nombre réel positif x , $\pi(x)$ désigne le nombre d'entiers premiers inférieurs à x) :

Théorème 5.1

Lorsque x tend vers $+\infty$, nous avons $\pi(x) \sim \frac{x}{\ln(x)}$ (i.e. $\pi(x)\ln(x)/x$ tend vers 1 lorsque x tend vers $+\infty$).

Notons $(p_n)_{n \geq 1}$ la suite des nombres premiers (avec $p_1 = 2, p_2 = 3, \dots$). Le théorème des nombres premiers s'énonce aussi sous la forme $p_n \sim n \ln n$ (i.e. $p_n/(n \ln n)$ tend vers 1 lorsque n tend vers $+\infty$).

Exercice 5.6 Entiers friables

1. Soient $n \geq 1$ et $N \geq 1$ deux entiers. Montrer que le nombre de suites d'entiers $e = (e_1, \dots, e_n)$ telles que $e_1 + \dots + e_n \leq N$ est égal au coefficient binomial $\binom{N+n}{n}$.

Indication

On pourra représenter une telle suite en coloriant les entiers de 1 à $N+n$ en deux couleurs : les e_1 premiers en blanc, suivi d'un noir, suivi de e_2 blancs et ainsi de suite.

2. Soient $n \geq 1$ un entier et $\lambda > 2p_n$ un entier impair ; posons $N = \lfloor \ln \lambda / \ln p_n \rfloor$. Déduire de la question précédente que la probabilité qu'un entier choisi uniformément aléatoirement dans $[1, \lambda]$ soit p_n -friable est supérieure à $(N^n / \lambda n!)$.
3. En appliquant le théorème des nombres premiers, donner une minoration de la fonction de Dickman-De Bruijn $\Psi(x^k, x)$ pour $x \geq 0$ suffisamment grand et $k \in \mathbb{N}$.

Solution

1. Avec $N = 3$ et $n = 3$, nous voyons que le nombre de partitions en trois parts des entiers 0, 1, 2 et 3 est bien égal aux nombres de suites obtenues en suivant l'indication de l'énoncé :

	1	2	3	4	5	6		1	2	3	4	5	6		1 + 0 + 0
0 + 0 + 0	●	●	●	○	○	○		○	●	●	●	○	○		1 + 0 + 0
0 + 0 + 1	●	●	○	●	○	○		○	●	●	○	●	○		1 + 0 + 1
0 + 0 + 2	●	●	○	○	●	○		○	●	●	○	○	●		1 + 0 + 2
0 + 0 + 3	●	●	○	○	○	●		○	●	○	●	●	○		1 + 1 + 0
0 + 1 + 0	●	○	●	●	●	○		○	●	○	●	○	●		1 + 1 + 1
0 + 1 + 1	●	○	●	○	●	○		○	●	○	○	●	●		1 + 2 + 0
0 + 1 + 2	●	○	●	○	○	●		○	○	●	●	●	○		2 + 0 + 0
0 + 2 + 0	●	○	○	●	●	●		○	○	●	●	○	●		2 + 0 + 1
0 + 2 + 1	●	○	○	●	●	○	●	○	○	●	○	●	●		2 + 1 + 0
0 + 3 + 0	●	○	○	○	●	●	●	○	○	○	●	●	●		3 + 0 + 0

Plus généralement, considérons l'ensemble $E_{N,n}$ constitués des sous-ensembles de $\{1, \dots, N+n\}$ à n éléments (correspondant aux entiers coloriés en noir). Posons f l'application injective

$$f : \left\{ \begin{array}{ccc} E_{N,n} & \longrightarrow & \mathbb{N}^n \\ \{\alpha_1, \dots, \alpha_n\} & \longmapsto & (\alpha_1 - 1, \alpha_2 - \alpha_1 - 1, \dots, \alpha_n - \alpha_{n-1} - 1) \end{array} \right.$$

où nous supposons $\alpha_1 < \dots < \alpha_n$ dans la définition de f . Pour tout élément de $E_{N,n}$, la somme des éléments de $f(\{\alpha_1, \dots, \alpha_n\})$ est égal à $\alpha_n - n$ où $\alpha_n = \max\{\alpha_1, \dots, \alpha_n\}$. La somme des éléments de $f(\{\alpha_1, \dots, \alpha_n\})$ est donc bien majorée par N .

Réciprocement, toute suite d'entiers $e = (e_1, \dots, e_n)$ telle que $e_1 + \dots + e_n \leq N$ est bien obtenue comme image par f en posant

$$\alpha_1 = e_1 + 1$$

$$\alpha_2 = e_2 + \alpha_1 + 1$$

⋮

$$\alpha_n = e_n + \alpha_{n-1} + 1$$

de sorte que $\alpha_n = e_1 + \dots + e_n + n \leq N + n$ et $f(\{\alpha_1, \dots, \alpha_n\}) = (e_1, \dots, e_n)$. Le cardinal recherché est donc égal au cardinal de $E_{N,n}$ c'est-à-dire au coefficient binomial $\binom{N+n}{n}$.

2. Un entier t est p_n -friable s'il est de la forme

$$t = p_1^{e_1} p_2^{e_2} \cdots p_n^{e_n} \quad \text{avec } e_1, \dots, e_n \in \mathbb{N}.$$

En notant $N = \lfloor \ln \lambda / \ln p_n \rfloor$, une condition suffisante pour que t appartienne à l'intervalle $[1, \lambda]$ est $e_1 + \dots + e_n \leq N$. En effet, par la croissance de la suite (p_i) , nous avons

$$p_1^{e_1} p_2^{e_2} \cdots p_n^{e_n} \leq p_n^{e_1 + \dots + e_n} \leq p_n^N \leq \lambda.$$

Le nombre d'entiers p_n -friables inférieur à λ est donc minoré par le nombre de suite (e_1, \dots, e_n) tels que $e_1 + \dots + e_n \leq N$, c'est-à-dire le coefficient binomial $\binom{N+n}{n}$ d'après la question précédente. Cet entier vérifie

$$\binom{N+n}{n} = \frac{(N+n)!}{N!n!} \geq \frac{N^n}{n!}$$

et la probabilité qu'un entier de $[1, \lambda]$ soit p_n -friable est donc supérieure à $(N^n/\lambda n!)$.

3. D'après la question précédente, nous avons

$$\Psi(x^k, x) \geq \left(\frac{\pi(x) + k}{\pi(x)} \right) = \left(\frac{\pi(x) + k}{k} \right) \geq \frac{\pi(x)^k}{k!} \geq \left(\frac{\pi(x)}{k} \right)^k$$

et en appliquant le théorème des nombres premiers, nous obtenons pour x suffisamment grand

$$\Psi(x^k, x) \geq \left(\frac{x}{k \ln x} \right)^k$$

Il existe des estimations plus fines de $\Psi(x, y)$ comme le développement asymptotique suivant [29] :

$$\Psi(x^k, x) = x\rho(k) + (1 - \gamma) \frac{x\rho(k-1)}{\log x} + O\left(x\rho(k) \frac{\log^2 k}{\log^2 y}\right)$$

où γ est la constante d'Euler, $\rho(k) = 1$ pour $0 \leq k \leq 1$, $\rho(k) = 1 - \log(k)$ pour $1 \leq k \leq 2$ et

$$\rho(u) = \frac{1}{u} \int_{u-1}^u \rho(t) dt, \text{ pour } u > 1$$

L'estimation de l'exercice précédent sera cependant suffisante pour nos besoins. En posant

$$L_N(\alpha, c) = \exp(c(\ln N)^\alpha (\ln \ln N)^{1-\alpha})$$

pour $\alpha \in [0, 1]$ et $c > 0$, nous obtenons en particulier que

$$\Psi(N, L_N(1/2, c))/N \simeq (L_N(1/2, 1/2c))^{-1}$$

pour N suffisamment grand.

L'exercice suivant présente et analyse un algorithme sous-exponentiel de résolution de logarithme discret dans un sous-groupe \mathbb{G} de $(\mathbb{Z}/p\mathbb{Z})^*$. Soit γ un générateur de \mathbb{G} et $\alpha \in \mathbb{G}$. L'algorithme utilise l'ensemble $\mathcal{B} = \{2, 3, 5, \dots, p_k\}$ constitués des k premiers nombres premiers (*i.e.* $p_1 = 2, p_2 = 3, \dots$) appelé *base de facteurs* et l'idée générale (due à M. KRAITCHIK) est la suivante :

- dans une première étape, l'algorithme recherche des relations de la forme

$$\alpha^{r_i} \cdot \gamma^{s_i} \cdot h_i \equiv p_1^{e_{i,1}} \cdots p_k^{e_{i,k}} \pmod{p} \quad (5.2)$$

avec $r_i, s_i \in (\mathbb{Z}/q\mathbb{Z})$ et $h_i \notin \mathbb{G}$ pour $i \in \{1, \dots, k\}$;

- dans une seconde étape, l'algorithme cherche à combiner ces relations (en appliquant des techniques d'algèbre linéaire) pour obtenir une relation de la forme $\alpha^r \gamma^s = 1$ et ainsi retrouver le logarithme discret de α en base γ .

Cet algorithme est appelé algorithme par *calcul d'indice* (*index calculus* en anglais).

Problème 5.7 Méthode de Kraitchik – Calcul d'indice *

Soient p un nombre premier et q un diviseur premier de $p - 1$. Notons \mathbb{G} le sous-groupe de $(\mathbb{Z}/p\mathbb{Z})^*$ d'ordre q et \mathbb{H} le sous-groupe de $(\mathbb{Z}/p\mathbb{Z})^*$ d'ordre $(p - 1)/q$. Notons γ un générateur de \mathbb{G} et supposons que $q^2 \nmid (p - 1)$.

1. Montrer que $\mathbb{G} \cap \mathbb{H} = \{1\}$.

2. Montrer que si r est tiré uniformément aléatoirement dans $(\mathbb{Z}/q\mathbb{Z})$ et h est tiré uniformément aléatoirement dans \mathbb{H} , alors l'élément $\gamma^r \cdot h$ est uniformément distribué dans $(\mathbb{Z}/p\mathbb{Z})^*$.
3. Proposer un algorithme probabiliste qui génère un élément uniformément distribué dans \mathbb{H} .

Soit T un paramètre dont la valeur sera choisie pour optimiser l'efficacité de l'algorithme. Notons $p_1 = 2, p_2 = 3, \dots, p_k$ les nombres premiers inférieurs à T .

4. Exprimer la probabilité que $\alpha^r \cdot \gamma^s \cdot h \bmod p$ se factorise sous la forme (5.2) si r et s sont tirés uniformément aléatoirement dans $(\mathbb{Z}/q\mathbb{Z})$ et h est tiré uniformément aléatoirement dans \mathbb{H} .
5. Proposer un algorithme qui étant donné un triplet $(r, s, h) \in (\mathbb{Z}/q\mathbb{Z})^2 \times \mathbb{H}$ vérifie si $\alpha^r \cdot \gamma^s \cdot h \bmod p$ est de la forme (5.2) en $O(k \log(p))$ opérations élémentaires. En déduire la complexité d'un algorithme pour générer $(k+1)$ relations de la forme (5.2).
6. Supposons connues $(k+1)$ relations de la forme (5.2). Montrer qu'il existe un vecteur (v_1, \dots, v_{k+1}) tel que

$$\begin{bmatrix} e_{1,1} & e_{2,1} & \dots & e_{k,1} & e_{k+1,1} \\ e_{1,2} & e_{2,2} & \dots & e_{k,2} & e_{k+1,2} \\ \vdots & & \vdots & & \vdots \\ e_{1,k} & e_{2,k} & \dots & e_{k,k} & e_{k+1,k} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \\ v_{k+1} \end{bmatrix} \in (q \cdot (\mathbb{Z}/p\mathbb{Z}))^k$$

et calculer la probabilité que $\sum_{i=1}^{k+1} v_i s_i \equiv 0 \bmod q$.

7. En déduire un algorithme de résolution du problème du logarithme discret dans \mathbb{G} de complexité

$$O\left(\frac{p-1}{\Psi(p-1, T)} k(\log^3(p) + k \log(p)) + k^3(\log(p))\right)$$

opérations élémentaires.

8. Montrer que pour $T = L_p(1/2, \sqrt{2}/2 + o(1))$, nous obtenons une complexité (sous-exponentielle) de l'ordre de $L_p(1/2, 3\sqrt{2} + o(1))$.

Solution

1. Soit $\theta \in \mathbb{G} \cap \mathbb{H}$. Nous avons $\theta^q \equiv 1 \bmod p$ et $\theta^{(p-1)/q} \equiv 1 \bmod p$, donc l'ordre de θ divise q et $(p-1)/q$. Comme q est premier, l'ordre de θ est égal à 1 ou q et comme par hypothèse q^2 ne divise pas $(p-1)$, nous obtenons que $\theta = 1$.

2. Le morphisme de groupe défini par

$$\begin{aligned}\mathbb{G} \times \mathbb{H} &\longrightarrow (\mathbb{Z}/p\mathbb{Z})^* \\ (g, h) &\longmapsto g \cdot h\end{aligned}$$

est injectif d'après la question précédente. Puisque $|\mathbb{G}| \times |\mathbb{H}| = (p-1)$, il est également surjectif. Pour une valeur de $r \in (\mathbb{Z}/q\mathbb{Z})$ tirée uniformément aléatoirement, γ^r est uniformément distribué dans \mathbb{G} . Si h est tiré uniformément aléatoirement dans \mathbb{H} , l'image du couple (γ^r, h) par ce morphisme est donc uniformément distribuée dans $(\mathbb{Z}/p\mathbb{Z})^*$.

3. Le sous-groupe \mathbb{H} est formé des éléments d'ordre $(p-1)/q$; il s'agit donc de l'image du morphisme

$$\begin{aligned}(\mathbb{Z}/p\mathbb{Z})^* &\longrightarrow (\mathbb{Z}/p\mathbb{Z})^* \\ x &\longmapsto x^q\end{aligned}$$

et tout élément de \mathbb{H} a q antécédents par ce morphisme. Pour générer un élément aléatoire de \mathbb{H} , il suffit donc de tirer uniformément aléatoirement un élément de $(\mathbb{Z}/p\mathbb{Z})^*$ et de calculer son image par ce morphisme (*i.e.* sa puissance q -ième).

4. D'après la question 2, l'élément construit est un élément uniformément distribué de $\{1, \dots, p-1\}$. De plus, il s'écrit sous la forme (5.2) s'il est T -friable. La probabilité recherchée est donc par définition $\Psi(p-1, T)/(p-1)$.
5. L'algorithme pour vérifier si $\alpha^r \cdot \gamma^s \cdot h \bmod p$ est de la forme (5.2) consiste simplement à diviser cette valeur par les k nombres premiers p_1, \dots, p_k tant que c'est possible. À chaque division, la taille de l'entier divisé diminue et la complexité totale est bien de l'ordre de $O(k \log(p))$ opérations élémentaires.

L'algorithme complet pour générer les $(k+1)$ relations de la forme (5.2) requiert de construire des éléments aléatoires $\alpha^r \cdot \gamma^s \cdot h$ puis de tester qu'ils vérifient bien la relation (5.2) en les factorisant par divisions successives. Un couple fournit une relation avec probabilité $\Psi(p-1, T)/(p-1)$ et le coût moyen de la génération de $(k+1)$ relations est :

$$O\left(\frac{p-1}{\Psi(p-1, T)} (k(\log^3(p) + k \log(p)))\right)$$

6. L'existence du vecteur \vec{v} est immédiate car la matrice a k lignes et $k+1$ colonnes.

Pour un ensemble de $k+1$ relations obtenues à partir des vecteurs \vec{s}^*, \vec{r}^* dans $(\mathbb{Z}/q\mathbb{Z})^{k+1}$ et $\vec{h}^* \in \mathbb{H}^{k+1}$, le même ensemble de relations est obtenu pour un choix arbitraire de vecteur \vec{s} dans $(\mathbb{Z}/q\mathbb{Z})^{k+1}$ en adaptant le vecteur $\vec{r} = \vec{r}^* - x\vec{s} + \vec{s}^*$ et en conservant $\vec{h}^* \in \mathbb{H}^{k+1}$.

La variable aléatoire \vec{s} est donc indépendante de l'ensemble de relations obtenues et donc du vecteur \vec{v} . La probabilité que le produit scalaire $\vec{s} \cdot \vec{v} \equiv 0 \bmod q$ soit nul est donc la probabilité uniforme $\Pr[\vec{s} \cdot \vec{v} \equiv 0 \bmod q] = 1/q$.

7. L'algorithme consiste simplement à construire les $k+1$ relations comme indiqué dans les questions précédentes, puis à construire le vecteur \vec{v} en résolvant un système linéaire. Lorsque ce vecteur \vec{v} existe, nous obtenons le logarithme discret de α en base γ .

En effet, nous avons

$$\alpha^{r_i} \cdot \gamma^{s_i} \cdot h_i \equiv p_1^{e_{i,1}} \cdots p_k^{e_{i,k}} \pmod{p}$$

pour tout $i \in \{1, \dots, k+1\}$. En élevant chacune de ces équations à la puissance v_i et en les multipliant, nous obtenons une relation de la forme

$$\alpha^r \cdot \gamma^s \cdot h \equiv p_1^{e_{1,q}} \cdots p_k^{e_{k,q}} \pmod{p}$$

avec $r = \vec{r} \cdot \vec{v}$, $s = \vec{s} \cdot \vec{v}$, $h \in \mathbb{H}$ et e_1, \dots, e_k des entiers. Puisque \mathbb{H} est l'image dans $(\mathbb{Z}/p\mathbb{Z})^*$ de l'élévation à la puissance q , nous obtenons l'existence d'un élément h' tel que $\alpha^r \gamma^s = h'$ et d'après la question 1, nous obtenons $\alpha^r \gamma^s = 1$. Si $s \neq 0$ (ce qui arrive avec probabilité $1 - 1/q$ d'après la question précédente), nous obtenons le logarithme discret $x \equiv -r/s \pmod{q}$.

Le coût total de l'algorithme est la somme du coût de la génération des relations (vu à la question 5) et de l'algèbre linéaire. En supposant que nous utilisons un algorithme cubique (comme une réduction de Gauss), nous obtenons le coût total suivant :

$$O\left(\frac{p-1}{\Psi(p-1, T)} k (\log^3(p) + k \log(p)) + k^3 (\log(p))\right)$$

Algorithme 5.6 Méthode de Kraitchik – Calcul d'indice

ENTRÉE: $\gamma, \alpha \in (\mathbb{Z}/p\mathbb{Z})^*$ et une base de facteurs $\mathcal{B} = \{2, 3, 5, \dots, p_k\}$

SORTIE: $x \in (\mathbb{Z}/q\mathbb{Z})$ tel que $\alpha = \gamma^x \pmod{p}$ ou ÉCHEC.

$i \leftarrow 0$

$\mathcal{L} \leftarrow \emptyset$

répéter

$i \leftarrow i + 1$

répéter

$r_i \xleftarrow{\text{u.a.}} (\mathbb{Z}/q\mathbb{Z}); s_i \xleftarrow{\text{u.a.}} (\mathbb{Z}/q\mathbb{Z})$

$g_i \xleftarrow{\text{u.a.}} (\mathbb{Z}/p\mathbb{Z})^*; h_i \xleftarrow{\text{u.a.}} g_i^q$

$m_i \leftarrow \alpha^{r_i} \cdot \gamma^{s_i} \cdot h_i \pmod{p}$

$\triangleright h_i$ uniforme dans \mathbb{H}

jusqu'à m_i est p_k -friable $\triangleright m_i = p_1^{e_{i,1}} \cdots p_k^{e_{i,k}}$ par divisions successives

$\vec{e}_i \leftarrow (e_{i,1}, \dots, e_{i,k})$

$\mathcal{L} \leftarrow \mathcal{L} \cup \{(r_i, s_i, \vec{e}_i)\}$

jusqu'à $i = k+1$

Trouver $\vec{v} \in (\mathbb{Z}/q\mathbb{Z})^{k+1}$ tel que $v_1 \vec{e}_1 + v_2 \vec{e}_2 + \cdots + v_{k+1} \vec{e}_{k+1} \in (q \cdot (\mathbb{Z}/p\mathbb{Z}))^k$.

\triangleright par élimination gaussienne

$r \leftarrow \sum_{i=1}^{k+1} v_i r_i \pmod{q}; s \leftarrow \sum_{i=1}^{k+1} v_i s_i \pmod{q}$

si $s \not\equiv 0 \pmod{q}$ **alors**

retourner $-r/s \pmod{q}$

sinon

retourner ÉCHEC

fin si

8. D'après l'estimation de l'exercice précédent, si $T = L_p(1/2, \sqrt{2}/2 + o(1))$, nous avons

$$\Psi(p-1, T)/p \simeq L_p(1/2, \sqrt{2}/2 + o(1))^{-1}$$

Par le théorème des nombres premiers, nous avons $k \simeq T \ln T$ et nous obtenons une complexité égale à

$$L_p(1/2, \sqrt{2}/2 + o(1)) \cdot L_p(1/2, \sqrt{2}/2 + o(1))^2 + L_p(1/2, \sqrt{2}/2 + o(1))^3$$

soit la complexité sous-exponentielle annoncée (les termes polynomiaux étant absorbés dans le $o(1)$) :

$$L_p(1/2, 3\sqrt{2}/2 + o(1))$$

Note

La complexité de l'algorithme précédent peut être améliorée en utilisant un algorithme plus efficace pour tester le caractère friable des nombres générés (par exemple avec l'algorithme ρ de Pollard que nous verrons au chapitre suivant) et en utilisant des techniques d'algèbre linéaire spécifique (puisque la matrice construite à partir des relations obtenues est très creuse). Nous obtenons avec ces modifications, un algorithme de complexité $L_p(1/2, \sqrt{2} + o(1))$ tel qu'il a été formalisé en 1979 par L. M. ADLEMAN [1]. Il existe des variantes et des améliorations (heuristiques) de cet algorithme dans tous les corps finis.

5.3 PROBLÈMES ALGORITHMIQUES LIÉS AU LOGARITHME DISCRET

Le premier exercice montre la propriété d'*auto-réductibilité* du problème du logarithme discret dans un groupe fixé \mathbb{G} . Cette propriété assure que s'il existe un algorithme qui fonctionne en temps polynomial pour une fraction polynomiale des instances du problème dans \mathbb{G} , alors il existe un algorithme probabiliste qui résout toutes les instances du problème en temps polynomial espéré.

Exercice 5.8 Auto-réductibilité du problème du logarithme discret

Soit \mathbb{G} un groupe fini cyclique et g un générateur de \mathbb{G} . Considérons un algorithme \mathcal{A} qui prend en entrée un élément de \mathbb{G} et retourne un entier, en temps τ (dans le pire des cas) où τ représente au moins le coût d'une exponentiation dans \mathbb{G} .

Supposons qu'il existe un sous-ensemble E de \mathbb{G} avec $\#E \geq \epsilon |\mathbb{G}|$ et $\epsilon \in]0, 1]$ pour lequel lorsque \mathcal{A} est exécuté sur un élément $h \in E$, l'entier retourné par \mathcal{A} est le logarithme discret de h en base g . Considérons l'algorithme \mathcal{B} défini à partir de \mathcal{A} dans l'algorithme 5.7.

Algorithme 5.7 Algorithme \mathcal{B}

ENTRÉE: $g, h \in \mathbb{G}$

SORTIE: $x \in (\mathbb{Z}/q\mathbb{Z})$ tel que $h = g^x \bmod p$.

```

tant que VRAI faire
     $c \leftarrow \overset{u.a.}{(\mathbb{Z}/q\mathbb{Z})}$ 
     $h' \leftarrow g^c$ 
     $w \leftarrow \mathcal{A}(h \cdot h')$ 
    si  $g^w = h \cdot h'$  alors
        retourner  $w - c \bmod q$ 
    fin si
fin tant que

```

Montrer que l'algorithme \mathcal{B} résout le problème du logarithme discret dans \mathbb{G} en temps espéré $O(\tau/\epsilon)$.

Solution

Remarquons tout d'abord que lorsque l'algorithme \mathcal{B} termine son exécution il retourne toujours la valeur correcte du logarithme discret de h en base g . En effet, si $g^w = h \cdot g^c$, le logarithme discret de h en base g est bien $w - c \bmod q$.

Le temps d'exécution de l'algorithme \mathcal{B} est essentiellement égal au temps d'exécution de l'algorithme \mathcal{A} multiplié par le nombre d'itérations de la boucle **tant que ... faire**. Cette boucle se termine dès que \mathcal{A} retourne la bonne valeur du logarithme discret pour $h \cdot h'$ ce qui par hypothèse se produit dès que $h \cdot h' \in E$. L'élément h' est tiré uniformément aléatoirement dans \mathbb{G} , donc la probabilité que $h \cdot h' \in E$ est indépendante de h et égale à ϵ . Le nombre espéré d'itérations de la boucle **tant que** est donc égal à $1/\epsilon$ d'où le résultat.

Nous avons vu dans la section 5.1 l'algorithme de Shanks qui résout le problème du logarithme discret dans un groupe \mathbb{G} en temps $O(\sqrt{|\mathbb{G}|})$. Cet algorithme s'adapte pour résoudre le problème du logarithme discret en temps $O(\sqrt{T})$ si nous savons que le logarithme discret appartient à un intervalle de longueur T . En raison du compromis temps-mémoire, cette méthode a malheureusement également une complexité en mémoire de $O(\sqrt{T})$ éléments. La méthode ρ de Pollard ne permet pas *a priori* de tirer avantage de la connaissance de l'intervalle d'appartenance du logarithme discret pour résoudre le problème en temps $O(\sqrt{T})$ et en mémoire $O(1)$. Dans l'exercice suivant, nous étudions un autre algorithme dû également à J. M. POLLARD qui atteint ces complexités.

Exercice 5.9 Algorithme λ de Pollard

Considérons un groupe multiplicatif cyclique \mathbb{G} engendré par $g \in \mathbb{G}$ d'ordre premier connu q et soit h un élément de \mathbb{G} . Nous supposons que le logarithme discret α de h en base g vérifie $\alpha \in [0, T]$ pour une valeur entière T connue.

Soit $F : \mathbb{G} \rightarrow S = \{s_1, \dots, s_\ell\} \subset \mathbb{N}$. Nous définissons une fonction $H : \mathbb{G} \rightarrow \mathbb{G}$ par $H(\alpha) = \alpha \cdot g^{F(\alpha)}$.

1. Considérons les deux suites $(x_n)_{n \in \mathbb{N}}$ et $(y_n)_{n \in \mathbb{N}}$ formées d'éléments de \mathbb{G} et définies par

$$\begin{aligned} x_0 &= g^u \quad \text{et} \quad x_{n+1} = H(x_n) \text{ pour } n \in \mathbb{N} \\ y_0 &= h \quad \text{et} \quad y_{n+1} = H(y_n) \text{ pour } n \in \mathbb{N}. \end{aligned}$$

Montrer que si l'on connaît deux valeurs i et j telles que $x_i = y_j$ alors il est possible calculer le logarithme discret de h en base g en $i + j$ applications de la fonction H et en mémoire constante.

2. Proposer un algorithme pour détecter une telle collision de valeur utilisant une mémoire constante. Analyser heuristiquement sa complexité.

Solution

1. Supposons que $x_i = y_j$ pour $i, j \in \mathbb{N}$. Nous avons

$$x_i = H^i(x_0) = g^u \cdot g^{F(x_0)} \cdot g^{F(x_1)} \cdots g^{F(x_{i-1})}$$

et

$$y_j = H^j(y_0) = h \cdot g^{F(y_0)} \cdot g^{F(y_1)} \cdots g^{F(y_{j-1})}$$

Nous obtenons donc

$$\log_g(h) = u + \sum_{k=0}^{i-1} F(x_k) - \sum_{t=0}^{j-1} F(y_t) \bmod q$$

Cette valeur peut être calculée en appliquant la fonction H dans \mathbb{G} ($i + j$) fois en mettant à jour la valeur du logarithme de la formule précédente à chaque étape en ajoutant $F(x_k)$ et en soustrayant $F(y_t)$.

2. Il suffit de remarquer que si $x_i = y_j$ pour $i, j \in \mathbb{N}$, alors $x_{i+t} = y_{i+t}$ pour tout entier $t \in \mathbb{N}$. Il suffit donc dans un premier temps de calculer les valeurs de la suite $(x_n)_{n \in \mathbb{N}}$ pour $n \in \{0, \dots, N\}$ (pour $N \in \mathbb{N}$ un paramètre à déterminer) et de calculer à chaque étape les logarithmes discrets associés

$$d_n = u + \sum_{i=0}^{n-1} F(x_i) \bmod q$$

L'algorithme stocke uniquement la valeur x_N et l'entier d_N (en plus de l'ensemble S et des éléments de groupe g^{s_i} pour $i \in \{1, \dots, \ell\}$).

Dans un deuxième temps, il calcule les valeurs de la suite $(y_n)_{n \in \mathbb{N}}$ pour $n \in \{0, \dots, M\}$ (pour $M \in \mathbb{N}$ un paramètre à déterminer) et les logarithmes discrets associés

$$e_n = \sum_{t=0}^{n-1} F(y_t)$$

tels que pour tout $n \in \{0, \dots, M\}$, $y_n = h \cdot g^{e_n}$. À chaque étape, il teste s'il y a une collision $y_n = x_N$ pour $n \in \{0, \dots, M\}$, et si c'est le cas, l'algorithme peut retourner comme logarithme discret la valeur $d_N - e_n \bmod q$.

Les suites $(x_n)_{n \in \mathbb{N}}$ et $(y_n)_{n \in \mathbb{N}}$ sont représentées sur la figure (5.2).

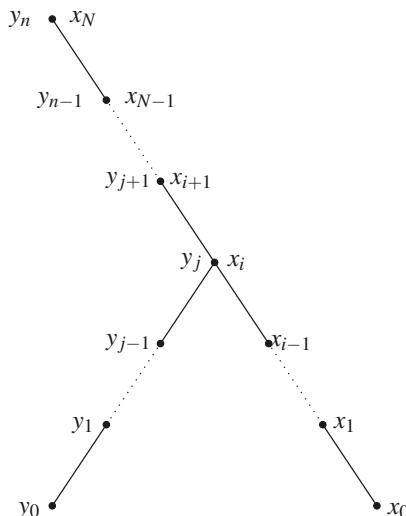


Figure 5.2 - Représentation des suites $(x_n)_{n \in \mathbb{N}}$ et $(y_n)_{n \in \mathbb{N}}$ dans l'algorithme λ de Pollard

L'algorithme retourne le logarithme discret s'il existe un entier $n \in \{0, \dots, M\}$ tel que $x_N = y_n$. La valeur moyenne du logarithme discret de x_N en base g pour une fonction aléatoire F sera de l'ordre de $u + sN$ où $s = (s_1 + \dots + s_\ell)/\ell$. La valeur moyenne du logarithme discret de l'élément y_n en base g sera de l'ordre de $\alpha + sn$. En choisissant u dans l'intervalle $[0, \sqrt{T}]$ et N et s de l'ordre de \sqrt{T} , nous obtenons $O(\sqrt{T})$ valeurs de logarithmes discrets pour x_i et pour y_j séparées en moyenne d'une distance s . Par conséquent, une fois que y_j est supérieur à $x_0 = u$, la probabilité d'une collision $x_i = y_j$ se produise pour $i \in \{0, \dots, N\}$ et $j \in \{0, \dots, M\}$ est de l'ordre de $s^{-1} = O(T^{-1/2})$. Avec N et M de l'ordre de \sqrt{T} , la probabilité d'une telle collision est constante et en répétant le calcul des deux suites un nombre constant de fois (en changeant la valeur initiale u), l'algorithme retourne le logarithme discret de h en base g en temps espéré $O(\sqrt{T})$ (en supposant les valeurs $g^{s_1}, \dots, g^{s_\ell}$ précalculées

ce qui demande $O(\ell \log(T))$ opérations de groupe). La complexité en mémoire est $O(\ell)$ éléments de groupe. Avec $\ell = O(1)$, nous obtenons l'algorithme (5.8) qui résout le problème du logarithme discret en temps $O(\sqrt{T})$ multiplications dans \mathbb{G} et en mémoire $O(1)$.

Algorithme 5.8 Algorithme λ de Pollard

ENTRÉE: $g, h \in \mathbb{G}$ avec $h = g^x$ et $x \in [0, T]$, $T, s_1, \dots, s_\ell \in \mathbb{N}$, $F : \mathbb{G} \rightarrow \{s_1, \dots, s_\ell\}$

SORTIE: $x \in \{0, \dots, T\}$ tel que $h = g^x$

```

pour  $i$  de 1 à  $\ell$  faire
   $g_i \leftarrow g^{s_i}$ 
fin pour                                 $\triangleright$  fin du précalcul des valeurs  $g^{s_1}, \dots, g^{s_\ell}$ 
tant que VRAI faire
   $u \xleftarrow{u.a.} \{0, 1, \dots, T\}$ 
   $d \leftarrow u$ 
   $x \leftarrow g^u$ 
  pour  $i$  de 1 à  $\lfloor \sqrt{T} \rfloor$  faire
     $d \leftarrow d + F(x)$ 
     $x \leftarrow x \cdot g_{F(x)}$ 
  fin pour                                 $\triangleright$  fin du calcul de  $x = x_{\lfloor \sqrt{T} \rfloor}$ 
   $y \leftarrow h$ 
   $e \leftarrow 0$ 
  pour  $i$  de 1 à  $2\lfloor \sqrt{T} \rfloor$  faire
    si  $x = y$  alors
      retourner  $d - e$ 
    fin si
     $e \leftarrow e + F(x)$ 
     $y \leftarrow y \cdot g_{F(x)}$ 
  fin pour
fin tant que

```

Note

L'algorithme de l'exercice précédent appelé *algorithme λ de Pollard* (en raison de la forme de la figure 5.2) ou « méthode des kangourous » a été proposé par J. M. POLLARD en 1978 [54, 55].

Le nombre de multiplications effectuées par l'algorithme d'exponentiation dichotomique 5.1 dépend du nombre de 1 dans le développement en base 2 de l'exposant considéré. Il a donc été suggéré pour rendre les protocoles cryptographiques plus efficaces d'utiliser des clés secrètes où ce nombre, le *poids de Hamming* de l'exposant, est relativement petit. Pour se prémunir d'attaque par recherche exhaustive de la clé secrète, il est nécessaire que le nombre de tels exposants soit suffisamment grand. Le nombre d'exposants de ℓ bits de poids w est donné par le coefficient binomial $\binom{\ell}{w}$. L'exercice suivant montre une adaptation de l'algorithme de Shanks pour résoudre le

problème du logarithme discret de petit poids de Hamming en $O(\ell \binom{\lceil \ell/2 \rceil}{\lceil w/2 \rceil})$ exponentiations dans le groupe considéré.

L'algorithme utilise la notion de système de décomposition.

Définition 5.3

Soient N un entier et ℓ et w deux entiers pairs. Un *système de décomposition* de type (N, ℓ, w) est un couple (X, \mathcal{C}) tel que

- X est un ensemble de cardinal ℓ ;
- \mathcal{C} est une famille de N sous-ensembles de X de cardinaux $\ell/2$;
- pour tout sous ensemble A de X de cardinal w , il existe un sous-ensemble $C \in \mathcal{C}$ tel que $\#(A \cap C) = w/2$.

Problème 5.10 Logarithme discret de petit poids de Hamming

Considérons un groupe multiplicatif cyclique \mathbb{G} engendré par $g \in \mathbb{G}$ d'ordre connu q un nombre premier de ℓ bits (*i.e.* $2^{\ell-1} < q < 2^\ell$). Soit w un entier dans $\{1, \dots, \ell\}$. Nous supposerons que ℓ et w sont pairs.

1. Donner un algorithme pour calculer le logarithme discret dans \mathbb{G} d'un élément h dont le poids de Hamming du logarithme discret est égal à w en $O(\binom{\ell}{w/2})$ exponentiations dans le groupe et dont la complexité en mémoire est $O(\binom{\ell}{w/2})$ éléments de groupe.
2. Montrer que s'il existe un système de décomposition de type (N, ℓ, w) alors le problème du logarithme discret de poids de Hamming égal à w peut être résolu en $O(N \binom{\ell/2}{w/2})$ exponentiations dans le groupe et en stockant $O(\binom{\ell/2}{w/2})$ éléments de groupe.
3. Posons $X = (\mathbb{Z}/\ell\mathbb{Z})$ et considérons la famille $\mathcal{C} = \{C_i, 0 \leq i \leq \ell/2 - 1\}$, définie par

$$C_i = \{i + j \bmod \ell, 0 \leq j \leq \ell/2 - 1\} \quad \text{pour } 0 \leq i \leq \ell/2 - 1$$

Montrer que (X, \mathcal{C}) est un système de décomposition de type $(\ell/2, \ell, w)$.

4. Conclure.

Solution

1. Notons

$$x = \sum_{i=0}^{\ell-1} x_i 2^i \in \mathbb{N} \text{ avec } x_i \in \{0, 1\} \text{ pour } i \in \{0, \dots, \ell - 1\}$$

5.3. Problèmes algorithmiques liés au logarithme discret

le logarithme discret de h en base g . Par définition, nous avons $w = \#\{i \in \{0, \dots, \ell - 1\}, x_i = 1\}$.

L'élément x se décompose donc sous la forme d'une somme de deux entiers de ℓ bits dont le poids de Hamming est $w/2$. Il y a $\binom{w}{w/2}$ représentations possibles pour un entier. Il suffit de modifier l'algorithme de Shanks pour rechercher deux entiers de ℓ bits et de poids de Hamming $w/2$ tels que

$$h = g^{x_1+x_2} = g^{x_1}g^{x_2}, \text{ soit } h \cdot g^{-x_2} = g^{x_1}$$

En notant $S_{\ell, w/2}$ l'ensemble des entiers de ℓ bits et de poids de Hamming $w/2$, nous obtenons immédiatement l'algorithme suivant qui nécessite au plus $O(\binom{\ell}{w/2})$ exponentiations dans le groupe et le stockage de $O(\binom{\ell}{w/2})$ éléments de groupe.

Algorithme 5.9 Algorithme de Shanks pour le logarithme discret de petit poids de Hamming

ENTRÉE: $g, h \in \mathbb{G}$, $q = |\mathbb{G}|$

SORTIE: $x \in \{0, \dots, q - 1\}$ tel que $h = g^x$

$\Upsilon \leftarrow \emptyset$

pour x dans $S_{\ell, w/2}$ **faire**

$g_x \leftarrow g^x$

$\Upsilon \leftarrow \Upsilon \cup \{(g_x, x)\}$

▷ Table de hachage

fin pour

pour x dans $S_{\ell, w/2}$ **faire**

$y \leftarrow h \cdot g^{-x}$

si $y = g_{x'}$ avec $(g_{x'}, x') \in \Upsilon$ **alors**

retourner $x + x' \bmod q$

fin si

fin pour

2. Considérons l'ensemble $X = \{0, \dots, \ell - 1\}$ et notons encore

$$x = \sum_{i=0}^{\ell-1} x_i 2^i \in \mathbb{N} \text{ avec } x_i \in \{0, 1\} \text{ pour } i \in \{0, \dots, \ell - 1\}$$

le logarithme discret de h en base g . Posons

$$A = \{i \in \{0, \dots, \ell - 1\}, x_i = 1\}$$

Par hypothèse, nous avons $\#A = w$. Considérons un système de décomposition de type (N, ℓ, w) , avec $\mathcal{C} = \{C_1, \dots, C_N\}$ une famille de sous-ensembles de X de cardinaux $\ell/2$.

Il existe un entier $i \in \{1, \dots, N\}$ tel que $\#(A \cap C_i) = w/2$. Posons A_i l'ensemble $A \setminus C_i$ et B_i l'ensemble $A_i \cap C_i$. Nous avons $A = A_i \cup B_i$, $A_i \cap B_i = \emptyset$ et $\#A_i = \#B_i = w/2$. Notons a_i et b_i les entiers dont le développement binaire sont donnés par A_i et B_i (respectivement). Nous avons

$$a_i = \sum_{n \in A_i} 2^n, \quad b_i = \sum_{n \in B_i} 2^n \quad \text{et} \quad x = a_i + b_i.$$

En considérant tous les entiers de la forme b_i et tous les entiers associés possibles de la forme a_i , nous pouvons construire l'algorithme 5.10 similaire à l'algorithme de Shanks (qui nécessite $O(N(\ell/2))$ exponentiations dans \mathbb{G} et le stockage $O(\ell/2)$ éléments de groupe).

Algorithme 5.10 Algorithme de logarithme discret basé sur les systèmes de décomposition

ENTRÉE: $g, h \in \mathbb{G}, q = |\mathbb{G}|$

SORTIE: $x \in \{0, \dots, q-1\}$ tel que $h = g^x$

pour i de 1 à N **faire**

$\Upsilon \leftarrow \emptyset$

pour B un sous-ensemble de C_i de cardinal $w/2$ **faire**

$b \leftarrow \sum_{n \in B} 2^n$

$g_b \leftarrow g^b$

$\Upsilon \leftarrow \Upsilon \cup \{(g_b, b)\}$

▷ Table de hachage

fin pour

pour A un sous-ensemble de $X \setminus C_i$ de cardinal $w/2$ **faire**

$a \leftarrow \sum_{n \in A} 2^n$

$y \leftarrow hg^{-a}$

si $y = g_b$ avec $(g_b, b) \in \Upsilon$ **alors**

retourner $a + b \bmod q$

fin si

fin pour

fin pour

3. Considérons la famille $\mathcal{C} = \{C_i, 0 \leq i \leq \ell/2 - 1\}$, définie par

$$C_i = \{i + j \bmod \ell, 0 \leq j \leq \ell/2 - 1\} \text{ pour } 0 \leq i \leq \ell/2 - 1$$

Soit A un sous-ensemble de $X = (\mathbb{Z}/\ell\mathbb{Z})$ de cardinal w . Notons pour tout $i \in (\mathbb{Z}/\ell\mathbb{Z})$,

$$\nu(i) = \#(C_i \cap A) - \#(X \setminus C_i \cap A)$$

la différence entre le nombre d'éléments de A dans C_i et ceux en dehors de C_i . Nous devons montrer qu'il existe un entier $i \in \{0, \dots, \ell/2 - 1\}$ tel que $\nu(i) = 0$.

Si $\nu(0) = 0$, le résultat est démontré. Supposons que $\nu(0) \neq 0$. En remarquant que

- $\nu(\ell/2) = -\nu(0)$;
- $|\nu(i+1) - \nu(i)| \in \{-2, 0, 2\}$ pour tout $i \in (\mathbb{Z}/\ell\mathbb{Z})$;
- $\nu(i)$ est pair pour tout $i \in (\mathbb{Z}/\ell\mathbb{Z})$,

il existe nécessairement un entier $i \in \{0, \dots, \ell/2 - 1\}$ tel que $\nu(i) = 0$.

4. Il suffit de combiner l'algorithme de la question 2 avec le système de décomposition de la question 3 pour obtenir un algorithme de logarithme discret qui nécessite $O(\ell/2(\ell/2))$ exponentiations dans le groupe et dont la complexité en mémoire est $O(\ell/2)$ éléments de groupe.

Note

La notion de système de décomposition et l'algorithme précédent ont été proposés par D. COPPERSMITH en 1997 (*cf.* [62]).

5.4 INTERPOLATION POLYNOMIALE DE LOGARITHME DISCRET

Toute fonction définie sur un ensemble fini peut s'exprimer sous la forme d'une fonction polynomiale en utilisant le polynôme d'interpolation de Lagrange. En particulier, pour tout nombre premier p et tout générateur g de $(\mathbb{Z}/p\mathbb{Z})^*$, il existe un polynôme $P : (\mathbb{Z}/p\mathbb{Z})^* \rightarrow (\mathbb{Z}/(p-1)\mathbb{Z})$ tel que $P(g^x) \equiv x \pmod{p-1}$ pour tout $x \in (\mathbb{Z}/(p-1)\mathbb{Z})$. Une formule explicite de ce polynôme a été présentée par A. L. WELLS JR. en 1984 [33, 50].

Exercice 5.11 Polynôme d'interpolation du logarithme discret

Soit p un nombre premier impair et soit g un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$.

1. Soit $n > 0$ un entier. Montrer que

$$\sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z})} \alpha^n = \begin{cases} 0 & \text{si } n \not\equiv 0 \pmod{p-1} \\ -1 & \text{sinon} \end{cases}$$

2. Soit $n \in \{0, \dots, p-1\}$ un entier. Montrer que

$$\sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{1\}} \frac{\alpha^n}{1-\alpha} \equiv n \pmod{p}$$

3. En déduire que pour tout entier x de $\{0, \dots, p-1\}$, en posant $h = g^x$, nous avons

$$x = \sum_{j=1}^{p-2} \frac{h^j}{1-g^j} \pmod{p}$$

Solution

1. Pour $n \equiv 0 \pmod{p-1}$, nous avons pour tout $\alpha \in (\mathbb{Z}/p\mathbb{Z})^*$, $\alpha^n \equiv 1 \pmod{p}$ et

$$\sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z})} \alpha^n = \sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z})^*} \alpha^n = \sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z})^*} 1 = p-1 = -1 \pmod{p}$$

Si $n \not\equiv 0 \pmod{p-1}$, en notant g un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$, nous avons

$$\sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z})} \alpha^n = \sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z})^*} \alpha^n = \sum_{i=1}^{p-1} (g^i)^n = \frac{g^{n(p-1)} - 1}{g^n - 1} = 0 \pmod{p}$$

2. Pour $n = 0$, nous avons

$$\sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{1\}} \frac{1}{1 - \alpha} = \sum_{\beta \in (\mathbb{Z}/p\mathbb{Z})^*} \beta \equiv 0 \pmod{p}$$

d'après la question précédente.

Pour $n \in \{1, \dots, p-1\}$, en effectuant la décomposition

$$\frac{X^n}{1-X} = \frac{X^n - X^{n-1} + X^{n-1} - X^{n-2} + X^{n-2} - \dots - X + X}{1-X}$$

nous obtenons

$$\begin{aligned} \sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{1\}} \frac{\alpha^n}{1 - \alpha} &= \sum_{j=1}^n \sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{1\}} \frac{\alpha^{j-1}(\alpha - 1)}{1 - \alpha} \\ &= - \sum_{j=1}^n \sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{1\}} \alpha^{j-1} \\ &= - \sum_{j=1}^n \left(\sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z})} \alpha^{j-1} - 1 \right) \\ &\equiv n \pmod{p} \end{aligned}$$

en appliquant la question précédente.

3. En remarquant que $\alpha^x = h^j$ pour $\alpha = g^j$, nous avons immédiatement

$$x = \sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{1\}} \frac{\alpha^x}{1 - \alpha} = \sum_{\alpha \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{0, 1\}} \frac{\alpha^x}{1 - \alpha} = \sum_{j=1}^{p-2} \frac{h^j}{1 - g^j} \pmod{p}$$

La formule précédente est inutile pour résoudre en pratique le problème du logarithme discret dans $(\mathbb{Z}/p\mathbb{Z})^*$ car le polynôme obtenu est de degré $p-2$ (*i.e.* de degré presque maximal). Cependant, s'il existait un ensemble $S \subset (\mathbb{Z}/q\mathbb{Z})^*$ de cardinal $\epsilon(p-1)$ pour $\epsilon \in]0, 1]$ pour lequel un polynôme de petit degré interpole le logarithme discret dans S , alors en appliquant l'auto-réductibilité du problème, nous obtiendrions un algorithme efficace pour le résoudre. L'exercice suivant montre que de tels polynômes sont peu susceptibles d'exister [14].

Exercice 5.12 Interpolation polynomiale de logarithme discret – Borne inférieure

Soient p un nombre premier impair et g un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$. Soient $S \subset (\mathbb{Z}/p\mathbb{Z})^*$ un ensemble de cardinal s et $f(X) \in (\mathbb{Z}/p\mathbb{Z})[X]$ un polynôme de degré n tel que

$f(g^x) \equiv x \pmod{p}$ pour tout $x \in (\mathbb{Z}/(p-1)\mathbb{Z})$ tel que $g^x \in S$. Le but de l'exercice est de montrer que le degré n est grand (en fonction du cardinal de S) :

$$n \geq \frac{s(s-1)}{2(p-2)}$$

1. En considérant l'ensemble

$$D = \{a \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{0, 1\} \mid a \equiv g_1 g_2^{-1} \pmod{p}, g_1, g_2 \in S\}$$

montrer qu'il existe $a \in D$ tel que le nombre r d'éléments $x \in \{1, \dots, p-1\}$ vérifiant

$$x \equiv f(g^x) \pmod{p} \text{ et } \alpha x \equiv f(ag^x) \pmod{p}$$

est supérieur à $s(s-1)/(p-2)$ (où α désigne le logarithme discret de a en base g).

2. Montrer que le nombre de racines de l'un des polynômes $h_1(X) = f(aX) - f(X) - \alpha$ ou $h_2(X) = f(aX) - f(X) - \alpha - 1$ dans $(\mathbb{Z}/p\mathbb{Z})$ est supérieur à $r/2$.
3. Conclure.

Solution

1. Nous avons immédiatement $\#D \leq p-2$ et il suffit d'appliquer le principe des tiroirs pour obtenir l'existence d'un $a \in D$ tel que le nombre de représentations $\{(g_1, g_2) \in S^2, a \equiv g_1 g_2^{-1} \pmod{p}\}$ est supérieur à $s(s-1)/D$. En appliquant la définition de f , nous obtenons immédiatement le résultat.
2. Considérons une représentation $a \equiv g_1 g_2^{-1} \pmod{p}$ avec g_1 et $a \cdot g_2$ dans S . Notons x_1 et x_2 les logarithmes discrets de g_1 et g_2 en base g . Puisque le logarithme discret de $g_1 \equiv ag_2 \pmod{p}$ en base g est égal à la somme des logarithmes discrets de a et de g_2 en base g modulo l'ordre de g , nous obtenons $x_1 = \alpha + x_2$ ou $x_1 = \alpha + x_2 - p + 1 = \alpha + x_2 + 1 \pmod{p}$.

Nous avons donc toujours

$$f(g_1) = f(ag_2) = \alpha + f(g_2) \pmod{p}$$

ou

$$f(g_1) = f(ag_2) = \alpha + f(g_2) + 1 \pmod{p}$$

Par conséquent, au moins un des deux polynômes $h_1(X) = f(aX) - f(X) - \alpha$ ou $h_2(X) = f(aX) - f(X) - \alpha - 1$ a au moins $r/2$ racines modulo p .

3. En raison du choix de D aucun de ces deux polynômes n'est nul dans $(\mathbb{Z}/p\mathbb{Z})[X]$. En effet,

$$h_1(0) = -\alpha \not\equiv 0 \pmod{p}$$

puisque $a \neq 1$, et

$$h_2(0) = -\alpha - 1 \not\equiv 0 \pmod{p}$$

puisque $\alpha \in \{2, \dots, p-1\}$. Les degrés de ces polynômes sont donc supérieurs à leur nombre de racines et le degré de l'un d'eux est supérieur à $r/2$. Nous obtenons donc $n \geq (s(s-1))/(2(p-2))$.

FACTORISATION DES ENTIERS ET PRIMALITÉ

La fonction à sens unique (conjecturale) la plus souvent citée est la multiplication des entiers. Il est en effet très facile de multiplier des entiers de grande taille mais il n'existe pas d'algorithme connu qui puisse inverser cette opération efficacement. Le cryptosystème à clé publique proposé par R. L. RIVEST, A. SHAMIR et L. M. ADLEMAN repose de façon essentielle sur la difficulté de cette inversion.

Le problème de la factorisation des entiers est un problème de théorie des nombres très ancien. En 1801, C. F. GAUSS écrivait dans *Disquisitiones Arithmeticae* que *distinguere nombres primi et numeros compostos, et decomponere ces derniers en facteurs primi, est un des problemes les plus importants et les plus utiles en arithmetique.* Le but de ce chapitre est de présenter les méthodes utilisées pour tester la primalité d'un nombre entier et factoriser des entiers composés.

Nous examinerons plusieurs algorithmes permettant de tester le caractère premier d'un nombre entier (notamment le *test de Fermat*, le *test de Solovay-Strassen* et le *test de Miller-Rabin*). Nous étudierons également certaines propriétés des *nombres de Carmichael* et des *nombres de Fermat*.

Nous analyserons ensuite plusieurs méthodes de factorisation de complexité exponentielle et sous-exponentielle, et les techniques algorithmiques associées. Nous examinerons des méthodes spécifiques de factorisation qui sont efficaces seulement pour certains types d'entiers (par exemple, la *méthode de Fermat* ou la *méthode $p - 1$ de Pollard*) et des techniques génériques comme la *méthode de Pollard-Strassen* ou la *méthode de Dixon*.

6.1 TESTS DE PRIMALITÉ

Notons \mathbb{P} l'ensemble des nombres premiers. Étant donné un entier $n \in \mathbb{N} \setminus \mathbb{P}$, exhiber un facteur de n donne une preuve facilement vérifiable que n n'est pas premier (nous utiliserons le mot *composé*). L'ensemble $\mathbb{N} \setminus \mathbb{P}$ défini donc un langage décidable en temps polynomial par une machine de Turing non déterministe (*i.e.* un langage de la classe de complexité NP). En 1975, V. PRATT [56] a montré que c'est également le cas du langage défini par l'ensemble \mathbb{P} en montrant que si n est un nombre premier, il existe une preuve (appelée *certificat de primalité* de n) qui peut être vérifiée en temps polynomial en la longueur de l'entier n .

Exercice 6.1 Certificats de primalité de Pratt

- Montrer qu'un entier n est premier si et seulement s'il existe un entier $a \in \mathbb{Z}$ tel que $a^{n-1} \equiv 1 \pmod{n}$ mais $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ pour tout diviseur premier q de $n - 1$.
- En déduire que tout nombre premier admet un certificat de primalité polynomial (en sa longueur binaire).

Solution

- L'entier n est premier si et seulement si $(\mathbb{Z}/n\mathbb{Z})^*$ est cyclique d'ordre $n - 1$ donc si et seulement s'il existe un élément $a \in (\mathbb{Z}/n\mathbb{Z})^*$ d'ordre $n - 1$, c'est-à-dire un entier $a \in \mathbb{Z}$ tel que $a^{n-1} \equiv 1 \pmod{n}$ mais $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ pour tout diviseur premier q de $n - 1$.
- La liste d'entiers (a, q_1, \dots, q_t) est un certificat de primalité de n si
 - $a^{n-1} \equiv 1 \pmod{n}$;
 - $a^{(n-1)/q_i} \not\equiv 1 \pmod{n}$ pour tout $i \in \{1, \dots, t\}$;
 - $n - 1 = q_1 \dots q_t$.
 - q_i est un nombre premier pour tout $i \in \{1, \dots, t\}$;

La question précédente montre que tout nombre premier possède un certificat de primalité et il est immédiat que la vérification des trois premières propriétés précédentes peut se réaliser en temps polynomial. Pour prouver la primalité de n , il suffit donc de fournir par récurrence un certificat de primalité des q_i pour $i \in \{1, \dots, t\}$.

Par récurrence, nous pouvons montrer qu'un certificat de primalité complet pour n nécessite moins de $(6 \log n - 4)$ entiers inférieurs à n . Cette propriété est vérifiée pour $n = 2$ et $n = 3$. Supposons la propriété vérifiée pour tout nombre premier $p < n$ pour $n > 3$ impair. Avec les notations précédentes, nous avons $n - 1 = q_1 \dots q_t$ avec $t \geq 2$ et

$$\sum_{k=1}^t (6 \log(q_k) - 4) + t + 1 = 6 \log(n - 1) - 3t + 1 \leq 6 \log(n) - 4$$

La vérification d'un certificat de primalité (complet) pour n peut donc se faire en temps polynomial.

Rappelons le « petit théorème de Fermat » qui est à la base de tous les tests de primalité que nous allons voir dans cette section :

Théorème 6.1 Petit théorème de Fermat

Si p est un nombre premier et si a est un entier non divisible par p alors

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

En particulier, si pour un entier n , il existe un entier a inférieur à n tel que $a^{n-1} \not\equiv 1 \pmod{n}$ alors n est un nombre composé. Le *test de primalité de Fermat* (cf. Algorithme 6.1) repose sur cette idée simple.

Algorithme 6.1 Test de primalité de Fermat

ENTRÉE: $n \in \mathbb{N}$, $a \in \mathbb{N}$.

SORTIE: COMPOSÉ OU PROBABLEMENT PREMIER

```

 $b \leftarrow a^{n-1} \pmod{n}$                                 ▷ par exponentiation dichotomique
si  $b = 1$  alors
    retourner PROBABLEMENT PREMIER
sinon
    retourner COMPOSÉ
fin si
```

Ce test de primalité est malheureusement insuffisant car pour toute valeur a , il existe des entiers composés n tels que $a^{n-1} \equiv 1 \pmod{n}$ (et l'algorithme peut retourner PROBABLEMENT PREMIER alors que l'entrée n est un nombre composé). Un tel nombre composé est appelé un nombre *pseudo-premier de Fermat en base a*. Les premiers nombres pseudo-premiers de Fermat en base 2 sont 341, 561, 645, 1105, 1387, 1729, 1905 ... L'exercice suivant montre qu'il existe une infinité de tels nombres pour toute base a .

Exercice 6.2 Nombres pseudo-premiers de Fermat en base a

Soit $a \geq 2$ un entier.

- Montrer $n = (a^{2p} - 1)/(a^2 - 1)$ est un nombre entier composé si $p \geq 3$ est un nombre impair.
- Montrer que si p est un nombre premier impair ne divisant pas $a^2 - 1$, alors $2p$ divise $n - 1$.
- En déduire qu'il existe une infinité de nombres pseudo-premiers de Fermat en base a .

Solution

- Nous avons

$$n = \frac{a^{2p} - 1}{a^2 - 1} = \frac{a^p - 1}{a - 1} \cdot \frac{a^p + 1}{a + 1}$$

Puisque $a - 1$ divise $a^p - 1$ et $a + 1$ divise $a^p + 1$ (pour p impair), n est un nombre entier composé.

2. Si p un nombre premier impair, par le petit théorème de Fermat, nous avons $a^p \equiv a \pmod{p}$ et donc $a^{2p} \equiv a^2 \pmod{p}$. Donc p divise $a^{2p} - a^2$ mais ne divise pas $a^2 - 1$ par hypothèse. Donc p divise $n - 1 = (a^{2p} - a^2)/(a^2 - 1)$. De plus $n - 1 = a^{2p-2} + a^{2p-4} + \dots + a^2$ est la somme d'un nombre pair de termes de même parité. Donc 2 divise $n - 1$ et $2p$ divise $n - 1$.
3. D'après la question précédente, si p est un nombre premier impair ne divisant pas $a^2 - 1$, $a^{2p} - 1$ est un diviseur de $a^{n-1} - 1$ mais $a^{2p} - 1$ est un multiple de n , donc $a^{n-1} \equiv 1 \pmod{n}$. La suite des nombres premiers ne divisant pas $a^2 - 1$ étant infinie, nous obtenons une famille infinie de nombres pseudo-premiers de Fermat en base a .

Un nombre de Carmichael est un entier qui est pseudo-premier de Fermat pour toute base, c'est-à-dire un entier naturel n non nul et composé tel que $a^{n-1} \equiv 1 \pmod{n}$ pour tout entier $a > 0$ premier avec n . L'exercice suivant montre que les nombres de Carmichael vérifient des conditions arithmétiques assez strictes connues sous le nom de *critère de Korselt*.

Problème 6.3 Nombres de Carmichael – Critère de Korselt

1. Montrer qu'un nombre de Carmichael est nécessairement impair.
2. Montrer que si p est un nombre premier alors $(\mathbb{Z}/p^t\mathbb{Z})^*$ est un groupe cyclique pour tout entier $t \geq 1$.
- Soient n un nombre de Carmichael, p un facteur premier de n et $\ell \geq 1$ tel que $p^\ell \mid n$ mais $p^{\ell+1} \nmid n$.
 3. Montrer qu'il existe un entier a tel que a est un générateur de $(\mathbb{Z}/p^\ell\mathbb{Z})^*$ et a est premier avec n/p^ℓ .
 4. En déduire que $\ell = 1$ et que $p - 1$ divise $n - 1$.
 5. Réciproquement, montrer que si n est un entier composé sans facteur carré, et tel que pour tout entier p divisant n , $p - 1$ divise $n - 1$ alors n est un nombre de Carmichael.
 6. En déduire qu'un nombre de Carmichael a au moins trois diviseurs premiers.

Solution

1. Pour un entier n pair, nous avons

$$(n - 1)^n \equiv (-1)^n \equiv 1 \not\equiv n - 1 \pmod{n}$$

et un nombre de Carmichael est donc nécessairement impair.

2. Nous savons que le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ est cyclique engendré par $\alpha \in (\mathbb{Z}/p\mathbb{Z})^*$. Nous allons construire à partir de α un entier γ tel que $\gamma \equiv \alpha \pmod{p}$ et γ est un générateur de $(\mathbb{Z}/p^t\mathbb{Z})^*$.

Notons m l'ordre de l'entier α dans $(\mathbb{Z}/p^t\mathbb{Z})^*$. Nous avons $\alpha^m \equiv 1 \pmod{p^t}$ et donc $\alpha^m \equiv 1 \pmod{p}$. Comme α est d'ordre $p-1$ modulo p , nous obtenons que $p-1$ divise m . L'élément $\alpha^{m/(p-1)}$ dans $(\mathbb{Z}/p^t\mathbb{Z})^*$ est donc d'ordre exactement $p-1$.

Montrons que l'élément $\beta = 1 + p$ est d'ordre p^{t-1} dans $(\mathbb{Z}/p^t\mathbb{Z})^*$. En calculant les puissances p -ième successives de β , nous obtenons

$$\begin{aligned}\beta &= 1 + p \\ \beta^p \pmod{p^t} &= (1 + p)^p \equiv 1 + p^2 \pmod{p^3} \\ \beta^{p^2} \pmod{p^t} &= (1 + p)^{p^2} \equiv 1 + p^3 \pmod{p^4} \\ &\vdots \\ \beta^{p^{t-2}} &= (1 + p)^{p^{t-2}} \equiv 1 + p^{t-1} \pmod{p^t} \\ \beta^{p^{t-1}} &= (1 + p)^{p^{t-1}} \equiv 1 \pmod{p^t}\end{aligned}$$

Ces puissances p -ième sont donc toutes distinctes et l'ordre de β est exactement égal à p^{t-1} . En notant γ le produit de $\alpha^{m/(p-1)}$ et β , l'ordre de γ est égal au plus petit commun multiple des ordres de $\alpha^{m/(p-1)}$ et β , soit $(p-1)p^{t-1}$ et nous avons donc montré que $(\mathbb{Z}/p^t\mathbb{Z})^*$ est cyclique.

3. Puisque p est premier, $(\mathbb{Z}/p^\ell\mathbb{Z})^*$ est cyclique. Soit γ un générateur de $(\mathbb{Z}/p^\ell\mathbb{Z})^*$. Par hypothèse, p^ℓ et n/p^ℓ sont premiers entre eux donc, par le théorème chinois des restes, il existe un entier a tel que $a \equiv \gamma \pmod{p^\ell}$ et $a \equiv 1 \pmod{n/p^\ell}$. Un tel entier a est premier avec n et est un générateur de $(\mathbb{Z}/p^\ell\mathbb{Z})^*$.
4. Puisque n est un nombre de Carmichael, nous avons

$$a^{n-1} \equiv 1 \pmod{n} \text{ donc } a^{n-1} \equiv 1 \pmod{p^\ell}.$$

L'ordre de a modulo p^ℓ est égal à $(p-1)p^{\ell-1}$, donc $(p-1)p^{\ell-1}$ divise $n-1$. En particulier si $\ell \geq 2$, p divise n et p divise $n-1$ et on obtient une contradiction. Donc $\ell = 1$ et on a bien que $(p-1)$ divise $(n-1)$.

5. Réciproquement, soit n un entier sans facteur carré tel que pour tout nombre premier p divisant n , $p-1$ divise $n-1$. Pour tout nombre premier p divisant n et tout entier a premier avec n , nous avons $a^{p-1} \equiv 1 \pmod{p}$ et donc $a^{n-1} \equiv 1 \pmod{p}$. Tout diviseur premier de n divise $a^{n-1} - 1$ et comme n est sans facteur carré, nous en déduisons que n divise $a^{n-1} - 1$, et donc que n est un nombre de Carmichael.
6. Supposons que $n = pq$ avec p et q deux nombres premiers impairs (et $p \neq q$). D'après le critère de Korselt, $p-1$ divise $n-1 = pq-1 = (p-1)q + q-1$. Donc $p-1$ divise $q-1$ et de même $q-1$ divise $p-1$, donc $p=q$. Un nombre de Carmichael a donc nécessairement au moins trois diviseurs premiers.

Le critère de Korselt suggère que les nombres de Carmichael sont relativement rares et l'exercice suivant montre que c'est le cas empiriquement.

Exercice 6.4 (avec programmation). Recherche de nombres de Carmichael

- Déduire du critère de Korselt la liste des nombres de Carmichael inférieurs à 10^5 .
- Montrer qu'il n'existe qu'un nombre fini de nombres de Carmichael de la forme pqr avec $p < q < r$ pour p fixé. Trouver tous les nombres de Carmichael de la forme $3pq$ et de la forme $5pq$ avec p et q premiers.

Solution

- Une recherche informatique (en utilisant le critère de Korselt) donne la courte liste suivante :

$$\begin{array}{ll}
 561 = 3 \cdot 11 \cdot 17 & 15841 = 7 \cdot 31 \cdot 73 \\
 1105 = 5 \cdot 13 \cdot 17 & 29341 = 13 \cdot 37 \cdot 61 \\
 1729 = 7 \cdot 13 \cdot 19 & 41041 = 7 \cdot 11 \cdot 13 \cdot 41 \\
 2465 = 5 \cdot 17 \cdot 29 & 46657 = 13 \cdot 37 \cdot 97 \\
 2821 = 7 \cdot 13 \cdot 31 & 52633 = 7 \cdot 73 \cdot 103 \\
 6601 = 7 \cdot 23 \cdot 41 & 62745 = 3 \cdot 5 \cdot 47 \cdot 89 \\
 8911 = 7 \cdot 19 \cdot 67 & 63973 = 7 \cdot 13 \cdot 19 \cdot 37 \\
 10585 = 5 \cdot 29 \cdot 73 & 75361 = 11 \cdot 13 \cdot 17 \cdot 31
 \end{array}$$

Il n'existe donc que 16 nombres de Carmichael inférieurs à 10^5 (alors qu'il y a 9592 nombres premiers inférieurs à cette borne).

- Soit n un nombre de Carmichael avec trois facteurs premiers $n = pqr$. D'après le critère de Korselt, $p - 1$ divise $n - 1 = pqr - 1 = (p - 1)qr + qr - 1$, donc $p - 1$ divise $qr - 1$ et en appliquant le même raisonnement à q et r , nous obtenons l'existence d'entiers h_1, h_2 et h_3 tels que

$$\begin{aligned}
 qr - 1 &= h_1(p - 1) \\
 pr - 1 &= h_2(q - 1) \\
 pq - 1 &= h_3(r - 1)
 \end{aligned}$$

Puisque $r - 1 > q$ (car q est nécessairement impair), nous avons $qh_3 < pq$ et donc $h_3 < p$. De plus, puisque r est premier, nous avons $r \neq pq$ et donc $h_3 \neq 1$. Pour p fixé, il n'existe donc qu'un nombre fini de valeurs de h_3 possibles.

Nous avons

$$h_2(q - 1) = p(r - 1) + (p - 1) = \frac{p}{h_3}(pq - 1) + (p - 1)$$

et donc

$$h_2 h_3 (q - 1) = p(pq - 1) + h_3(p - 1) = p(p(q - 1) + (p - 1)) + h_3(p - 1)$$

Finalement, nous obtenons

$$(h_2 h_3 - p^2)(q - 1) = (p + h_3)(p - 1)$$

et

$$q - 1 = \frac{(p - 1)(p + h_3)}{h_2 h_3 - p^2}$$

Il n'y a qu'un nombre fini de diviseurs de $(p - 1)(p + h_3)$ et donc qu'un nombre fini de valeurs pour h_2 et pour q . La relation $pq - 1 = h_3(r - 1)$ nous donne l'unique valeur de r possible et nous avons bien montré qu'il n'y a qu'un nombre fini de nombres de Carmichael de la forme pqr avec $p < q < r$ pour p fixé.

En particulier, une recherche informatique montre que 561 est le seul nombre de Carmichael de la forme $3pq$ et que 1105 et 2465 sont les deux seuls nombres de Carmichael de la forme $5pq$.

Même si les nombres de Carmichael sont rares, W. R. ALFORD, A. GRANVILLE et C. POMERANCE ont démontré en 1994 qu'il en existe une infinité [3]. Il est donc nécessaire d'utiliser des critères de primalité plus stricts que le « petit théorème de Fermat ».

Soit n un entier positif. Un entier a est un *carré modulo n* (ou est un *résidu quadratique modulo n*) s'il existe $b \in \mathbb{N}$ tel que $a \equiv b^2 \pmod{n}$. Cette propriété dépend uniquement de la classe de a modulo n et lorsque n est un nombre premier impair p , le *critère d'Euler* donne une caractérisation simple des carrés modulo p .

Théorème 6.2 Critère d'Euler

Si p est un nombre premier et si a est un entier non divisible par p , alors $a^{(p-1)/2} \equiv 1 \pmod{p}$ si a est un carré modulo p et $a^{(p-1)/2} \equiv -1 \pmod{p}$ sinon.

Dans ce cas, nous utiliserons le *symbole de Legendre* pour noter si a est un résidu quadratique modulo p .

Définition 6.1 Symbole de Legendre

Soit p un nombre premier impair et $a \in \mathbb{Z}$. Le symbole de Legendre de a modulo p est défini par :

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{si } p \text{ divise } a \\ 1 & \text{si } a \text{ est un résidu quadratique modulo } p \\ -1 & \text{si } a \text{ n'est pas un résidu quadratique modulo } p \end{cases}$$

Le critère d'Euler s'énonce alors, pour $n = p$ un nombre premier impair, sous la forme

$$\forall a \in (\mathbb{Z}/n\mathbb{Z})^*, \quad a^{(n-1)/2} = \left(\frac{a}{n}\right) \text{ mod } n \quad (6.1)$$

Le *symbole de Jacobi* est une extension formelle du symbole de Legendre dont les propriétés permettent de calculer plus facilement des symboles de Legendre.

Définition 6.2 Symbole de Jacobi

Soit $n \geq 2$ un entier impair dont la décomposition en facteurs premiers est $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_k}\right)^{\alpha_k} \cdots \left(\frac{a}{p_k}\right)^{\alpha_k}$$

Si pour deux entiers a et n (avec n non premier), $\left(\frac{a}{n}\right) = -1$ nous assure que a n'est pas un résidu quadratique modulo n , nous ne pouvons pas déduire en général de $\left(\frac{a}{n}\right) = 1$ que a est un résidu quadratique modulo n .

Pour tout nombre premier p impair, nous avons

$$\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2} \text{ et } \left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$$

La valeur d'un symbole de Jacobi quelconque peut être calculée très efficacement (*i.e.* en temps quadratique) en utilisant de façon répétée la *loi de réciprocité quadratique*.

Théorème 6.3 Loi de réciprocité quadratique

Soient m et n deux nombres premiers entre eux et impairs. Nous avons

$$\left(\frac{m}{n}\right) \cdot \left(\frac{n}{m}\right) = (-1)^{(n-1)(m-1)/4}$$

En utilisant la relation (6.1), nous obtenons un nouveau critère de primalité vérifié pour tout nombre premier.

Algorithme 6.2 Test de primalité d'Euler

ENTRÉE: $n \in \mathbb{N}$ impair, $a \in \mathbb{N}$.

SORTIE: COMPOSÉ OU PROBABLEMENT PREMIER

```

 $b_1 \leftarrow a^{(n-1)/2} \text{ mod } n$                                  $\triangleright$  par exponentiation dichotomique
 $b_2 \leftarrow \left(\frac{a}{n}\right)$                                           $\triangleright$  par la loi de réciprocité quadratique
si  $b_1 = b_2$  alors
    retourner PROBABLEMENT PREMIER
sinon
    retourner COMPOSÉ
fin si
    
```

Étant donné un entier a , un nombre composé peut vérifier cette égalité. Dans ce cas, il est appelé un nombre *pseudo-premier d'Euler en base a* . L'exercice suivant montre qu'il n'existe pas de nombre pseudo-premier d'Euler en base a pour toute base a .

Exercice 6.5 Test de primalité de Solovay-Strassen

1. Démontrer le critère d'Euler
 2. Montrer que si $n \geq 3$ impair n'est pas premier alors pour (au moins) la moitié des $a \in (\mathbb{Z}/n\mathbb{Z})^*$, nous avons
- $$\left(\frac{a}{n}\right) \not\equiv a^{(n-1)/2} \pmod{n}$$
3. En déduire pour tout entier T un algorithme qui, prenant en entrée un entier n , retourne Composé ou Premier en $O(T \log^3 n)$ opérations binaires, de sorte que
 - si l'algorithme retourne Composé, alors n est toujours un nombre composé ;
 - si l'algorithme retourne Premier, alors la probabilité que n soit composé est inférieure à 2^{-T} .

Solution

1. Soit p un nombre premier et g un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$. Pour tout élément $a \in (\mathbb{Z}/p\mathbb{Z})^*$, il existe un unique entier $x \in (\mathbb{Z}/(p-1)\mathbb{Z})$ tel que $a = g^x \pmod{p}$ et a est un carré modulo p si et seulement si x est pair. Nous avons donc

$$a^{(p-1)/2} \equiv g^{x(p-1)/2} \pmod{p} \equiv (-1)^x \pmod{p} \equiv \left(\frac{a}{p}\right)$$

2. Soit $n \geq 3$ un nombre composé. Posons

$$J_n = \left\{ a \in (\mathbb{Z}/n\mathbb{Z})^*, \left(\frac{a}{n}\right) \equiv a^{(p-1)/2} \pmod{n} \right\}.$$

J_n est un sous-groupe de $(\mathbb{Z}/n\mathbb{Z})^*$ et il suffit de montrer que c'est un sous-groupe propre de $(\mathbb{Z}/n\mathbb{Z})^*$ pour montrer le résultat. Supposons par l'absurde que $J_n = (\mathbb{Z}/n\mathbb{Z})^*$. Nous avons en particulier que n est un nombre de Carmichael.

Soit p un diviseur premier de n . Puisque n est sans facteur carré, n/p est premier avec p . Soit $g \in (\mathbb{Z}/p\mathbb{Z})^*$ un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$. Soit $a \in (\mathbb{Z}/n\mathbb{Z})^*$ tel que

$$\begin{cases} a \equiv g \pmod{p} \\ a \equiv 1 \pmod{n/p} \end{cases}$$

qui existe d'après le théorème chinois des restes. Nous avons

$$\begin{aligned} \left(\frac{a}{n}\right) &= \left(\frac{a}{p}\right)\left(\frac{a}{n/p}\right) = \left(\frac{a \bmod p}{p}\right)\left(\frac{a \bmod n/p}{n/p}\right) \\ &= \left(\frac{g}{p}\right)\left(\frac{1}{n/p}\right) \\ &= \left(\frac{g}{p}\right) = -1 \end{aligned}$$

Supposons que $a \in J_n$. Nous avons $a^{(n-1)/2} \equiv -1 \pmod{n}$ et comme (n/p) divise n , nous avons $a^{(n-1)/2} \equiv -1 \pmod{n/p}$ ce qui contredit $a \equiv 1 \pmod{(n/p)}$.

Donc J_n est un sous-groupe propre de $(\mathbb{Z}/n\mathbb{Z})^*$, donc de cardinal au plus $\varphi(n)/2$ où $\varphi(n) = \#(\mathbb{Z}/n\mathbb{Z})^*$ est la fonction indicatrice d'Euler de n . Par conséquent, pour au moins la moitié des $a \in (\mathbb{Z}/n\mathbb{Z})^*$, nous avons

$$\left(\frac{a}{n}\right) \not\equiv a^{(n-1)/2} \pmod{n}$$

3. L'algorithme consiste simplement à itérer le test de primalité d'Euler (*cf.* algorithme 6.2) T fois en tirant uniformément aléatoirement une nouvelle base a pour chaque itération. D'après la question précédente, si n est composé, il existe au moins $\varphi(n)/2$ éléments de $(\mathbb{Z}/n\mathbb{Z})^*$ pour lesquels le test d'Euler va échouer. La probabilité que tous les tests réussissent si n est composé est donc inférieure à $(1/2)^T$. Le temps d'exécution de l'algorithme est $O(T \log^3 n)$. L'algorithme obtenu est le test de primalité de Solovay-Strassen qui a été proposé en 1977 par R. SOLOVAY et V. STRASSEN [61].

Algorithme 6.3 Test de primalité de Solovay-Strassen

ENTRÉE: $n \in \mathbb{N}$, $a \in \mathbb{N}$.

SORTIE: COMPOSÉ ou PREMIER

pour i de 1 à T **faire**

$a \xleftarrow{u.a.} (\mathbb{Z}/n\mathbb{Z})^*$

$b_1 \leftarrow a^{(n-1)/2} \pmod{n}$ ▷ par exponentiation dichotomique

$b_2 \leftarrow \left(\frac{a}{n}\right)$ ▷ par la loi de réciprocité quadratique

si $b_1 \neq b_2$ **alors**

retourner COMPOSÉ

fin si

fin pour

retourner PREMIER

Le test de primalité de Fermat repose sur le fait que $a^{p-1} \equiv 1 \pmod{p}$ pour tout nombre premier p . Le test de primalité de Solovay-Strassen utilise le fait que $a^{(p-1)/2} \equiv \pm 1 \pmod{p}$. Nous allons maintenant étudier le *test de primalité de Miller-Rabin* qui utilise de façon itérée l'idée que si $x^2 \equiv 1 \pmod{p}$ alors $x \equiv \pm 1 \pmod{p}$. Il a

été proposé par G. L. MILLER dans une version déterministe [48] (qui est polynomiale sous l'hypothèse de Riemann généralisée ; une hypothèse de théorie des nombres encore non démontrée) et par M. O. RABIN qui l'a présenté sous la forme probabiliste du prochain exercice [57].

Problème 6.6 Test de primalité de Miller-Rabin

- Soit n un entier premier impair. Notons $n = m2^h + 1$ avec m impair et soit $a \in \mathbb{Z}$ un entier premier à n . Considérons la suite (b_0, b_1, \dots, b_h) d'entiers compris entre 0 et $n - 1$ définie par :

$$b_0 \equiv a^m \pmod{n}, \quad b_1 \equiv b_0^2 \pmod{n}, \quad \dots, \quad b_h \equiv b_{h-1}^2 \pmod{n}$$
 - (a) Montrer que $b_h = 1$.
 - (b) Si $b_0 \neq 1$ montrer qu'il existe un indice $i \in \{0, \dots, h - 1\}$ tel que $b_i = -1$.
- Montrer que si n est un entier composé impair supérieur ou égal à 3 : $n = m2^h + 1$ avec m impair alors le nombre d'entiers $a \in (\mathbb{Z}/n\mathbb{Z})^*$, pour lesquels la suite (b_0, b_1, \dots, b_h) vérifie les deux conditions précédentes, est inférieur à $(n - 1)/2$.
- En déduire pour tout entier T un algorithme qui, prenant en entrée un entier n , retourne Composé ou Premier en $O(T \log^3 n)$ opérations binaires, de sorte que
 - si l'algorithme retourne Composé, alors n est toujours un nombre composé ;
 - si l'algorithme retourne Premier, alors la probabilité que n soit composé est inférieure à 2^{-T} .

Solution

- (a) Nous avons $b_h \equiv b_0^{2^h} \equiv a^{2^h m} \equiv a^{n-1} \pmod{n}$. Par le petit théorème de Fermat, nous obtenons donc bien $b_h \equiv 1 \pmod{n}$.
 (b) Supposons que $b_0 \neq 1$. Notons i l'indice minimal pour lequel $b_i \equiv 1 \pmod{n}$. D'après la question précédente, nous avons $0 < i \leq h$. L'entier b_{i-1} vérifie $b_{i-1}^2 \equiv b_i \equiv 1 \pmod{n}$ et $b_{i-1} \not\equiv 1 \pmod{n}$. Puisque n est premier, le polynôme $X^2 - 1$ n'a que deux racines modulo n et nous en déduisons que $b_{i-1} \equiv -1 \pmod{n}$.
- Considérons l'ensemble \mathcal{Y}_n qui contient tous les entiers $a \in (\mathbb{Z}/n\mathbb{Z})^*$ pour lesquels la suite (b_0, b_1, \dots, b_h) vérifie les deux conditions précédentes. \mathcal{Y}_n est un sous-groupe de $(\mathbb{Z}/n\mathbb{Z})^*$. Comme dans l'exercice précédent, il suffit de montrer que \mathcal{Y}_n est un sous-groupe propre non trivial de $(\mathbb{Z}/n\mathbb{Z})^*$ pour obtenir que $\#\mathcal{Y}_n \leq \varphi(n)/2$.
 Supposons que $\mathcal{Y}_n = (\mathbb{Z}/n\mathbb{Z})^*$ puisque dans le cas contraire, le résultat est immédiat. En particulier, n est un nombre de Carmichael et il est donc sans facteur carré avec au moins deux facteurs premiers. Nous pouvons écrire $n = n_1 n_2$ où n_1 et n_2 sont deux entiers supérieurs à 1 premiers entre eux.

Considérons j l'entier maximal pour lequel il existe un élément v de $(\mathbb{Z}/n\mathbb{Z})^*$ tel que $v^{2^j m} = -1 \pmod{n}$. Considérons le sous-groupe de $(\mathbb{Z}/n\mathbb{Z})^*$ défini par

$$\mathcal{Y}_n = \{a \in (\mathbb{Z}/n\mathbb{Z})^*, a^{2^j m} \equiv \pm 1 \pmod{n}\}$$

Ce sous-groupe est non trivial par définition de j et notons $v \in \mathcal{Y}_n$ tel que $v^{2^j m} \equiv -1 \pmod{n}$.

Considérons l'élément $w \in (\mathbb{Z}/n\mathbb{Z})^*$ défini par $w \equiv v \pmod{n_1}$ et $w \equiv 1 \pmod{n_2}$. L'élément w est bien défini par le théorème chinois des restes puisque n_1 et n_2 sont premiers entre eux.

Nous avons alors $w^{2^j m} \equiv v^{2^j m} \pmod{n_1}$ et $w^{2^j m} \equiv 1 \pmod{n_2}$. Puisque $v^{2^j m} \equiv -1 \pmod{n}$, nous avons $w^{2^j m} \equiv -1 \pmod{n_1}$. Par conséquent, nous avons $w^{2^j m} \not\equiv \pm 1 \pmod{n}$.

Par ailleurs, nous avons $w^{2^{j+1} m} \equiv (-1)^2 \pmod{n_1}$ et $w^{2^{j+1} m} \equiv 1 \pmod{n_2}$, donc $w^{2^{j+1} m} \equiv 1 \pmod{n}$ et nous obtenons une contradiction.

\mathcal{Y}_n est donc un sous-groupe propre non trivial de $(\mathbb{Z}/n\mathbb{Z})^*$ et nous avons $\#\mathcal{Y}_n < \varphi(n)/2$ et le nombre d'entiers $a \in (\mathbb{Z}/n\mathbb{Z})^*$ pour lesquels la suite (b_0, b_1, \dots, b_h) vérifie les deux conditions précédentes est majoré par $(n-1)/2$.

3. Comme pour le test de primalité de Solovay-Strassen, l'algorithme consiste simplement à calculer la suite (b_0, b_1, \dots, b_h) pour T valeurs de a tirées uniformément aléatoirement dans $(\mathbb{Z}/n\mathbb{Z})^*$. Si n est composé, il existe au moins $\varphi(n)/2$ éléments de $(\mathbb{Z}/n\mathbb{Z})^*$ pour lesquels le test va échouer. La probabilité que tous les tests réussissent si n est composé est donc inférieure à $(1/2)^T$. Le temps d'exécution de l'algorithme obtenu (cf. Algorithme 6.4) est $O(T \log^3 n)$.

Algorithme 6.4 Test de primalité de Miller-Rabin

ENTRÉE: $n = 1 + 2^h m \in \mathbb{N}$, avec m impair

SORTIE: COMPOSÉ ou PREMIER

```

pour  $i$  de 1 à  $T$  faire
     $a \xleftarrow{u.a.} (\mathbb{Z}/n\mathbb{Z})^*$ 
     $b \leftarrow a^m \pmod{n}$                                  $\triangleright$  par exponentiation dichotomique
    pour  $j$  de 1 à  $h-1$  faire
        si  $b \neq n-1$  et  $b^2 = 1$  alors
            retourner COMPOSÉ
        fin si
         $b \leftarrow b^2 \pmod{n}$ 
    fin pour
    si  $b \neq 1$  alors
        retourner COMPOSÉ
    fin si
fin pour
retourner PREMIER

```

Il est possible de montrer que le nombre d'entiers $a \in (\mathbb{Z}/n\mathbb{Z})^*$, pour lesquels la suite (b_0, b_1, \dots, b_h) vérifie les deux conditions de la suite est inférieur à $\varphi(n)/4$.

Les deux algorithmes précédents montrent que l'ensemble \mathbb{P} définit donc un langage décidable en temps polynomial par une machine de Turing probabiliste avec une probabilité d'erreur inférieure à $1/3$ pour toutes les instances (*i.e.* un langage de la classe de complexité BPP). Pendant longtemps, le problème de savoir si l'ensemble \mathbb{P} défini un langage décidable en temps polynomial par une machine de Turing déterministe (*i.e.* un langage de la classe de complexité P) a été ouvert. En 2002, M. AGRAWAL, N. KAYAL et N. SAXENA [2] ont montré que c'est effectivement le cas. Leur algorithme repose sur une identité polynomiale qui généralise le petit théorème de Fermat.

Exercice 6.7 Identité de Agrawal-Kayal-Saxena

- Montrer que p est un nombre premier si et seulement si p divise tous les coefficients binomiaux $\binom{p}{i}$ pour $i \in \{2, \dots, p-1\}$.
- En déduire que si n est un nombre entier et si a est un nombre premier avec n , alors n est un nombre premier si et seulement si

$$(X+a)^n \equiv X^n + a \pmod{n}$$

Solution

- Lorsque p est un nombre premier, p divise le coefficient binomial

$$\binom{p}{i} = \frac{p \cdot (p-1) \cdots (p-i+1)}{i \cdot (i-1) \cdots 1}$$

pour tout $i \in \{2, \dots, p-1\}$, simplement parce qu'il s'agit d'un entier naturel et que le numérateur est divisible par p mais pas le dénominateur.

Réciproquement, si n est un nombre composé, notons p le plus petit facteur premier de n et $k = n/p$. Nous avons

$$\begin{aligned} \binom{n}{p} &= \frac{n(n-1)(n-2) \cdots (n-p+1)}{p!} \\ &= \frac{k(n-1)(n-2) \cdots (n-p+1)}{(p-1)!} \\ &\not\equiv 0 \pmod{n} \end{aligned}$$

En effet, dans le cas contraire, le numérateur doit être divisible par k et dans ce cas le produit $(n-1)(n-2) \cdots (n-p+1)$ doit être divisible par p , ce qui est impossible puisque p divise n et est premier (il est donc premier avec $n-i$ pour tout entier $i \in \{1, \dots, p-1\}$).

- Le coefficient de X^i dans le polynôme

$$(X+a)^n - (X^n + a) \pmod{n}$$

est égal à $\binom{n}{i}a^{p-i}$. Si n est premier, le coefficient binomial $\binom{n}{i}$ est divisible par n et par conséquent tous les coefficients du polynômes sont nuls modulo n .

Réciproquement, supposons que n est composé. D'après la question précédente, n ne divise pas l'un des coefficients binomiaux $\binom{n}{i}$ pour $i \in \{2, \dots, n-1\}$ et comme n est premier avec a , le coefficient de X^i n'est pas égal à 0 modulo n et le polynôme $(X+a)^n - (X^n + a)$ mod n n'est pas nul modulo n .

L'approche naïve qui consiste à tester l'identité polynomiale en calculant tous les coefficients de $(X+a)^n$ est prohibitive et M. AGRAWAL, N. KAYAL et N. SAXENA ont donné un moyen efficace pour vérifier cette égalité en travaillant dans des anneaux quotients en vérifiant des identités du type

$$(X+a)^n = X^n + a \text{ mod } (n, X^r - 1)$$

L'analyse de cet algorithme dépasse cependant le cadre de cet ouvrage.

Pour terminer cette section, nous présentons un test de primalité qui s'exécute en temps polynomial déterministe prouvé mais seulement pour une classe restreinte de nombres entiers.

Le test de Pépin est un test de primalité pour les nombres de Fermat, c'est-à-dire les nombres de la forme $F_n = 2^{2^n} + 1$ pour $n \in \mathbb{N}$.

Exercice 6.8 Nombres de Fermat et test de primalité de Pépin

- Soit k un entier positif. Montrer que si l'entier $2^k + 1$ est un nombre premier alors k est une puissance de 2.

On appelle n -ième nombre de Fermat le nombre $F_n = 2^{2^n} + 1$ pour $n > 0$.

- Montrer que si F_n est premier, alors un entier g est un générateur de $(\mathbb{Z}/F_n\mathbb{Z})^*$ si et seulement si

$$\left(\frac{g}{F_n}\right) = -1$$

En déduire que si F_n est premier alors 3 est un générateur de $(\mathbb{Z}/F_n\mathbb{Z})^*$.

- Montrer que si $3^{(F_n-1)/2} \equiv -1 \pmod{F_n}$ alors F_n est premier.

Solution

1. Il suffit d'écrire k sous la forme $k = a \cdot 2^b$ avec $a \in \mathbb{N}$ impair et $b \in \mathbb{N}$. En remarquant que le polynôme $X^a + 1$ se factorise sous la forme

$$(X + 1) \sum_{i=0}^{a-1} (-1)^i X^i$$

nous obtenons

$$2^k + 1 = (2^{2^b})^a + 1 = (1 + 2^{2^b}) \sum_{i=0}^{a-1} (-1)^i (2^{2^b})^i$$

et $(1+2^{2^b})$ est donc un diviseur de 2^k+1 . Si 2^k+1 est premier nous avons nécessairement $2^k + 1 = 2^{2^b} + 1$ et $a = 1$.

2. Supposons que

$$\left(\frac{g}{F_n} \right) \equiv -1 \pmod{F_n}$$

Par le critère d'Euler, nous avons

$$g^{(F_n-1)/2} \equiv -1 \pmod{F_n}$$

En particulier, nous obtenons $g^{(F_n-1)} \equiv 1 \pmod{F_n}$, et l'ordre multiplicatif de g modulo F_n divise donc $F_n - 1 = 2^{2^n}$ (qui est une puissance de 2). Par ailleurs, l'ordre de g ne divise pas $(F_n - 1)/2$, il est donc nécessairement égal à $F_n - 1$. Réciproquement, si F_n un premier, tout générateur de $(\mathbb{Z}/F_n\mathbb{Z})^*$ n'est pas un carré et a pour symbole de Legendre la valeur -1 .

Nous avons $2^{2^n} \equiv 1 \pmod{3}$ et donc $F_n \equiv 2 \pmod{3}$. Par conséquent

$$\left(\frac{F_n}{3} \right) = -1 \text{ et donc } \left(\frac{3}{F_n} \right) = -1$$

en appliquant la loi de réciprocité quadratique (et le fait que $F_n \equiv 1 \pmod{4}$). Donc si F_n est premier, 3 est un générateur de $(\mathbb{Z}/F_n\mathbb{Z})^*$.

3. Comme dans la question précédente, si

$$3^{(F_n-1)/2} \equiv -1 \pmod{F_n}$$

L'ordre multiplicatif de 3 modulo F_n est égal à $F_n - 1$. Le groupe multiplicatif $(\mathbb{Z}/F_n\mathbb{Z})^*$ est donc d'ordre maximal et F_n est un nombre premier.

Note

Les nombres de Fermat doivent leur nom au mathématicien français P. DE FERMAT qui émit la conjecture que tous ces nombres étaient premiers. Les entiers F_0, F_1, F_2, F_3 et F_4 sont effectivement premiers mais cette conjecture a été démentie par L. EULER en 1732 qui a montré

que

$$F_5 = 2^{2^5} + 1 = 2^{32} + 1 = 4294967297 = 641 \cdot 6700417$$

Les nombres F_n pour $n \in \{5, \dots, 32\}$ sont tous composés.

6.2 MÉTHODES EXPONENTIELLES DE FACTORISATION

La méthode de factorisation par divisions successives est la technique la plus simple pour factoriser des nombres entiers. Étant donné un entier n , elle consiste à essayer de diviser n par chaque nombre premier inférieur ou égal à \sqrt{n} . Si un tel diviseur est trouvé, un facteur de n est trouvé et dans le cas contraire, l'entier n est premier. La complexité de l'algorithme dans le pire des cas est exponentielle (puisque elle demande \sqrt{n} divisions), néanmoins cette méthode est toujours la première à tester car elle permet de trouver les petits facteurs de l'entier n .

Exercice 6.9 (avec programmation). Factorisation par divisions successives

Factoriser l'entier

$$n = 3148240326296492491829836036538028522262397298543021512290073$$

par divisions successives.

Solution

L'application directe des divisions successives montre que $n = pq$ avec

$$p = 88983774230476795134763412284708944571110946255868729$$

$$q = 35379937$$

où p et q sont des nombres premiers (comme on peut le vérifier facilement en appliquant le test de primalité de Miller-Rabin par exemple).

L'algorithme de Fermat est une méthode de factorisation des entiers qui tente de factoriser un entier impair n en l'écrivant comme la différence de deux carrés $n = a^2 - b^2$. En notant $n = a\beta$ avec a le plus grand diviseur de n inférieur à \sqrt{n} , le nombre d'opérations arithmétiques est de l'ordre de $(\sqrt{n} - a)^2/2a$. En particulier si n possède un diviseur proche de \sqrt{n} , la méthode de Fermat est plus rapide que la méthode des divisions successives mais dans le pire des cas, elle peut demander jusqu'à n opérations arithmétiques.

Algorithme 6.5 Méthode de factorisation de Fermat**ENTRÉE:** $n \in \mathbb{N}$ **SORTIE:** PREMIER ou $d \in \mathbb{N}$, $d | n$, $d \neq 1, n$ **pour** a de $\lceil \sqrt{n} \rceil$ à $(n + 9)/6$ **faire** $b \leftarrow \sqrt{a^2 - n}$ **si** $b \in \mathbb{N}$ **alors** **retourner** $a - b$ **fin si****fin pour****retourner** PREMIER**Exercice 6.10 (avec programmation).** Factorisation par la méthode Fermat

Factoriser l'entier

$$n = 4433634977317959977189716351978918572296527677331175210881861$$

par la méthode de Fermat.

SolutionL'application directe de la méthode de Fermat révèle immédiatement que $n = pq$ avec

$$p = 2105619855842445264527580721807$$

$$q = 2105619855842445264527580722923$$

avec $|p - q| = 1116$ et p et q sont des nombres premiers (comme on peut le vérifier facilement en appliquant le test de primalité de Miller-Rabin par exemple).

L'efficacité de la méthode de Fermat pour factoriser un entier n est optimale lorsque l'un des facteurs de n est proche de \sqrt{n} . Le but de l'exercice suivant est d'étudier une variante de cette méthode où l'entier n est multiplié par un facteur choisi pour rendre la méthode de Fermat efficace.

Exercice 6.11 Algorithme de Lehman *

- Montrer, en utilisant le principe des tiroirs, que pour tout nombre réel α et tout entier $V \geq 1$, il existe un couple d'entiers u, v avec $1 \leq v \leq V$ tels que

$$\left| \alpha - \frac{u}{v} \right| < \frac{1}{vV}$$

2. Soit $n = pq$ un entier impair avec $n^{1/3} < p \leq q$. Montrer qu'il existe deux entiers u et v tels que

$$|uq - vp| < n^{1/3}$$

et $uv \leq \lceil n^{1/3} \rceil$.

Indication

On pourra appliquer le résultat précédent avec $V = n^{1/6} \sqrt{q/p}$.

3. En remarquant, avec les mêmes notations, que $4(uv)n = a^2 - b^2$ avec $a = uq + vp$ et $b = |uq - vp|$, expliquer comment modifier la méthode de Fermat pour qu'elle retourne un facteur non trivial de n en $O(n^{1/3})$ opérations arithmétiques.

Solution

1. Considérons les $V + 1$ réels $a_i = i \cdot \alpha - \lfloor i \cdot \alpha \rfloor$ de l'intervalle $[0, 1[$ pour $i \in \{0, \dots, V\}$ et la partition de cet intervalle en V sous-intervalles de même longueur :

$$[0, 1[= \bigcup_{r=0}^{V-1} \left[\frac{r}{V}, \frac{r+1}{V} \right[$$

Par le principe des tiroirs, il existe un intervalle de cette partition qui contient deux réels a_i et a_j distincts. Leur différence est donc inférieure à $1/V$ en valeur absolue. En posant $v = |i - j|$ et $u = |\lfloor i \cdot \alpha \rfloor - \lfloor j \cdot \alpha \rfloor|$, nous avons

$$\frac{1}{V} \geq |a_i - a_j| = |(i - j)\alpha - \lfloor i \cdot \alpha \rfloor - \lfloor j \cdot \alpha \rfloor| = |v\alpha - u|$$

et le résultat (qui est le *théorème d'approximation de Dirichlet*).

2. En suivant l'indication, nous appliquons le théorème d'approximation de Dirichlet à $\alpha = p/q$ avec $V = n^{1/6} \sqrt{q/p}$. Nous obtenons l'existence d'un couple d'entiers (u, v) avec $1 \leq v \leq V$ tels que

$$\left| \frac{u}{v} - \frac{p}{q} \right| < \frac{1}{vV}$$

soit

$$|uq - vp| < \frac{q}{V} = \frac{\sqrt{pq}}{n^{1/6}} = \frac{n^{1/2}}{n^{1/6}} = n^{1/3}$$

Nous avons alors

$$uv = \frac{u}{v} \cdot v^2 < \left(\frac{p}{q} + \frac{1}{vV} \right) v^2 \leq \frac{p}{q} \cdot \frac{q}{p} n^{1/3} + \frac{v}{V} \leq n^{1/3} + 1$$

et puisque uv est un entier, nous avons bien $uv \leq \lceil n^{1/3} \rceil$.

3. Supposons que n n'a pas de facteur plus petit que $n^{1/3}$ (ce que l'on teste par divisions successives en $O(n^{1/3})$ opérations élémentaires) et que n n'est pas le cube d'un nombre premier (ce que l'on teste en temps polynomial par des méthodes d'analyse numérique comme la méthode de Newton). L'entier n est alors soit premier, soit le produit de deux nombres premiers.

Si n n'est pas premier, il existe deux entiers p et q tels que $n = pq$ avec $n^{1/3} < p \leq n^{1/2} \leq q < n^{2/3}$. Il existe des entiers k , a et b vérifiant

$$\begin{aligned} 4kn &= a^2 - b^2 \\ 1 &\leq k \leq \lceil n^{1/3} \rceil \\ 0 &\leq b \leq n^{1/3} \end{aligned} \tag{6.2}$$

avec $k = uv$, $a = uq + vp$ et $b = |uq - vp| < n^{1/3}$ où les entiers u et v sont les entiers dont l'existence est assurée par la question précédente. Nous avons

$$a = (4kn + b^2)^{1/2} \leq (4kn + n^{2/3})^{1/2} = 2\sqrt{kn} \left(1 + \frac{1}{4kn^{1/3}}\right)^{1/2}$$

et donc

$$2\sqrt{kn} \leq a \leq 2\sqrt{kn} \left(1 + \frac{1}{4kn^{1/3}}\right) \tag{6.3}$$

L'algorithme consiste donc à tester pour tout k dans l'intervalle défini par (6.2) et tout entier a de l'intervalle défini par (6.3) si $4kn - a^2$ est un carré b^2 . Si c'est le cas puisque

$$a + b \leq 2\sqrt{kn} \left(1 + \frac{1}{4kn^{1/3}}\right) + n^{1/3} < n$$

pour $n \geq 21$, nous avons $0 \leq a - b \leq a + b < n$ et $\text{pgcd}(a + b, n)$ révèle donc un diviseur non trivial de n . En effet, si l'on suppose que $\text{pgcd}(a + b, n) = 1$, puisque $4kn = (a - b)(a + b)$, n divise $(a - b)$ ce qui contredit l'inégalité précédente. L'entier n est premier si l'algorithme ne trouve pas de solution.

Le nombre de tests effectués par l'algorithme est

$$\sum_{k=1}^{\lceil n^{1/3} \rceil} \left(1 + \frac{n^{1/6}}{k^{1/2}}\right) = O\left(n^{1/6} \lceil n^{1/3} \rceil^{1/2} + \lceil n^{1/3} \rceil\right) = O(n^{1/3}).$$

d'où le résultat. L'algorithme obtenu a été proposé en 1974 par R. LEHMAN [40].

Algorithme 6.6 Algorithme de Lehman

ENTRÉE: $n \in \mathbb{N}$

SORTIE: PREMIER ou $p \in \mathbb{N}$, $p | n$, $p \neq 1, n$

```

pour  $p = 2$  à  $n^{1/3}$  faire
    si  $p | n$  alors
        retourner  $p$ 
    fin si
fin pour
pour  $k = 1$  à  $\lceil n^{1/3} \rceil$  faire
    pour  $a = 2\sqrt{kn}$  à  $2\sqrt{kn}\left(1 + \frac{1}{4kn^{1/3}}\right)$  faire
        si  $b = \sqrt{4kn - a^2}$  est un entier alors
            retourner pgcd( $a + b, n$ )
        fin si
    fin pour
fin pour
retourner PREMIER

```

L'exercice suivant présente une méthode exponentielle de factorisation des entiers due à J. M. POLLARD [52]. Il s'agit d'une méthode spécifique qui est efficace pour les entiers dont au moins l'un des diviseurs p est tel que $p - 1$ est friable.

Exercice 6.12 Méthode $p - 1$ de Pollard

Soit n un entier impair et soit $B > 0$ une borne fixée. Supposons qu'il existe un nombre premier p divisant n tel que toute puissance d'un nombre premier q^α divisant $p - 1$ est inférieure à B .

1. Soit a l'entier $2^{B!}$. Montrer qu'il existe un algorithme calculant $a' \equiv a \pmod{n}$ en $O(B \log B)$ multiplications modulaires modulo n .
2. Montrer que $p - 1$ divise $B!$ puis que $a' \equiv 1 \pmod{p}$. En déduire que le plus grand diviseur commun de $a' - 1$ et n est un facteur non trivial de n sauf si $a' = 1$.
3. En déduire un algorithme de factorisation.

Solution

1. L'algorithme est simplement l'algorithme d'exponentiation dichotomique 5.1 appliqué à la base 2 et l'exposant $B!$. Sa complexité est de $2 \cdot \log_2(B!)$ multiplications modulaires dans $(\mathbb{Z}/n\mathbb{Z})^*$ et par la formule d'approximation de Stirling, nous avons $\log(B!) = B \log(B) - B/\log(e) + O(\log(B)) = O(B \log B)$.
2. Considérons la décomposition en facteurs premiers de $p - 1$:

$$p - 1 = q_1^{e_1} \dots q_k^{e_k}$$

avec q_1, \dots, q_k des nombres premiers deux à deux distincts et $e_i \geq 1$ pour $i \in \{1, \dots, k\}$. Par hypothèse sur n , $q_i^{e_i}$ est inférieur à B pour tout $i \in \{1, \dots, k\}$, donc $q_i^{e_i}$ divise $B!$ pour tout $i \in \{1, \dots, k\}$ et comme les entiers $q_1^{e_1}, \dots, q_k^{e_k}$ sont premiers entre eux, nous obtenons que $p - 1$ divise $B!$.

Par le petit théorème de Fermat, nous avons $a^{p-1} \equiv 1 \pmod{p}$ et donc $a^{B!} \equiv 1 \pmod{p}$ et en réduisant a' modulo p , nous obtenons $a' \equiv 1 \pmod{p}$.

L'entier p divise $a' - 1$ et n et il divise donc le plus grand diviseur commun de $a' - 1$ et n . Calculer ce plus grand diviseur commun révèle donc un facteur non trivial de n sauf si $a' = 1$.

3. L'algorithme de factorisation consiste simplement à calculer itérativement la valeur a' en (faisant grandir la valeur de B) et à tester si le plus grand diviseur commun révèle un facteur non trivial de n .

Algorithme 6.7 Méthode $p - 1$ de Pollard

ENTRÉE: $n \in \mathbb{N}$, B

SORTIE: $r \in \mathbb{N}$, $r \mid n$

```

 $a \leftarrow 2$ 
pour  $i$  de 1 à  $B$  faire
     $a \leftarrow a^i$ 
     $r \leftarrow \text{pgcd}(a - 1, n)$ 
    si  $r \neq 1$  et  $r \neq n$  alors
        retourner  $r$ 
    fin si
fin pour

```

Si la valeur r est égale à 1, cela indique généralement que n n'a pas de diviseur premier p tel que $p - 1$ soit suffisamment friable alors que si la valeur retournée est $r = n$, cela signifie généralement que tous les facteurs le sont. Il existe de nombreuses variantes de cet algorithme comme la méthode $p+1$ de Williams ou la méthode ECM de Lenstra (qui utilise des courbes elliptiques). En raison de cette attaque, pendant longtemps les standards cryptographiques préconisaient de vérifier qu'un module RSA n'était pas constitué de nombres premiers ayant cette propriété. En raison du développement de la méthode ECM, cette condition n'est plus réellement pertinente de nos jours.

Exercice 6.13 (avec programmation). Factorisation par la méthode $p - 1$ de Pollard

Factoriser l'entier

$n = 117827681420271584017432903522327303325344948050665323956545863$
par la méthode $p - 1$ de Pollard.

Solution

L'application directe de la méthode $p - 1$ de Pollard révèle immédiatement que $n = pq$ avec

$$p = 51837792271037335517474380291201$$

$$q = 2273007322615163386667987960263$$

où p et q sont des nombres premiers (comme on peut le vérifier facilement en appliquant le test de primalité de Miller-Rabin par exemple) avec $p - 1 = 2^7 \cdot 3^5 \cdot 5^2 \cdot 7 \cdot 11^4 \cdot 17^2 \cdot 29 \cdot 41 \cdot 47 \cdot 89 \cdot 131 \cdot 151 \cdot 173 \cdot 223 \cdot 593$.

En 1975, J. POLLARD [53] a introduit un nouvel algorithme de factorisation des entiers. En considérant une fonction $F : (\mathbb{Z}/n\mathbb{Z}) \longrightarrow (\mathbb{Z}/n\mathbb{Z})$ et s_0 un élément de $(\mathbb{Z}/n\mathbb{Z})$, l'algorithme construit une suite définie par la récurrence $s_{i+1} = F(s_i)$ pour $i \in \mathbb{N}$. Comme l'algorithme ρ de Pollard pour le logarithme discret (vu dans le chapitre précédent), la méthode utilise peu de mémoire.

Algorithme 6.8 Algorithme ρ de Pollard (pour la factorisation)

ENTRÉE: $n \in \mathbb{N}$

SORTIE: $d \in \mathbb{N}$, $d \mid n$, $d \neq 1$, n ou ÉCHEC

$x \xleftarrow{u.a} (\mathbb{Z}/n\mathbb{Z})^*$

$y \leftarrow x$

$d \leftarrow 1$

tant que $d = 1$ faire

$x \leftarrow F(x)$

$y \leftarrow F(y)$

$y \leftarrow F(y)$

$d \leftarrow \text{pgcd}(|x - y|, n)$

fin tant que

si $d = n$ alors

retourner ÉCHEC

sinon

retourner d

fin si

Exercice 6.14 Algorithme ρ de Pollard

Analyser l'algorithme ρ de Pollard pour la factorisation.

Solution

Le principe est similaire à l'algorithme ρ de Pollard pour le logarithme discret vu dans l'exercice 5.3. La suite $(s_i)_{i \in \mathbb{N}}$ prend ses valeurs dans un ensemble fini $(\mathbb{Z}/n\mathbb{Z})$. Elle est

6.3. Multi-évaluation de polynômes et algorithme de Pollard-Strassen

définie par récurrence et elle est donc ultimement périodique. Soit p un diviseur premier de n , la suite $(s_i \bmod p)_{i \in \mathbb{N}}$ est également une suite ultimement périodique. Il existe donc un indice $j < k$ pour lequel $s_j = s_k \bmod p$ pour $k > j$. La suite $(s_i)_{i \in \mathbb{N}}$ étant définie par la récurrence $s_{i+1} = F(s_i)$ pour tout entier i , nous avons $s_{j+t} = s_{k+t}$ pour tout entier $t \geq 0$ et la suite $(s_i)_{i \in \mathbb{N}}$ peut être représentée comme sur la figure 6.1.

Si nous trouvons deux valeurs j et k telles que $s_j = s_k \bmod p$ et $s_j \neq s_k \bmod n$, alors le plus grand diviseur commun de $(s_j - s_k)$ et n produit une factorisation non triviale de n . Contrairement à la méthode ρ de Pollard pour le problème du logarithme discret, il est impossible de chercher directement une collision de la forme $s_\ell = s_{2\ell} \bmod p$, puisque calculer les valeurs de la suite $(s_i \bmod p)_{i \in \mathbb{N}}$ est impossible sans la connaissance de p . L'algorithme calcule donc en parallèle s_i et s_{2i} pour $i \in \mathbb{N}$ et recherche une collision par le calcul de plus grand diviseur commun.

Si la fonction F est une fonction aléatoire de $(\mathbb{Z}/n\mathbb{Z})^*$, le nombre espéré d'éléments de la suite $(s_i \bmod p)_{i \in \mathbb{N}}$ pour obtenir deux valeurs communes modulo p est de l'ordre de $j \sim \sqrt{\pi p/2}$. La complexité de l'algorithme pour trouver un facteur p de n est en $O(\sqrt{p})$ applications de la fonction F et donc de l'ordre de $O(n^{1/4})$ si le plus petit facteur premier de n est de l'ordre \sqrt{n} .

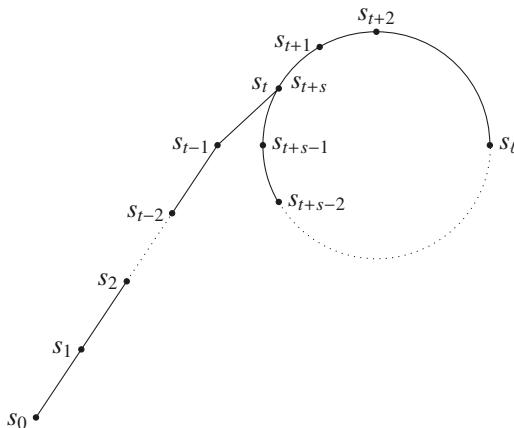


Figure 6.1 – Représentation de la suite $(s_n)_{n \in \mathbb{N}}$ dans l'algorithme ρ de Pollard

6.3 MULTI-ÉVALUATION DE POLYNÔMES ET ALGORITHME DE POLLARD-STRASSEN

Dans cette section, nous allons présenter l'algorithme déterministe de factorisation des entiers ayant la meilleure complexité prouvée. Il a une complexité en temps similaire à celle de l'algorithme ρ de Pollard mais est très coûteux en mémoire. L'algorithme repose sur des techniques d'évaluation d'un polynôme univarié en plusieurs

valeurs. Cet algorithme a d'autres applications en cryptographie (comme nous le verrons au chapitre 7) et nous le détaillons sous la forme de trois exercices distincts : les deux premiers sont liés à l'algorithmique des polynômes et le second se concentre sur le problème de la factorisation des entiers.

Exercice 6.15 Division euclidienne rapide par la méthode de Newton

Soit A un anneau commutatif unitaire (en pratique nous considérerons $A = \mathbb{Z}$ ou $A = (\mathbb{Z}/N\mathbb{Z})$). Soient $S, T \in A[X]$ avec $\deg(S) = n$, $\deg(T) = m$ et T unitaire.

- Montrer que l'algorithme classique de division euclidienne de S par T a une complexité arithmétique en $O(nm)$ (*i.e.* le nombre d'opérations nécessaires dans l'anneau A est en $O(nm)$).
- Pour $P \in A[X]$ et $k = \deg(P)$, nous notons $\text{Rec}(P(X)) = X^k P(1/X)$ le polynôme réciproque de P . Montrer que

$$\text{Rec}(Q) = \text{Rec}(S)\text{Rec}(T)^{-1} \bmod X^{n-m+1}$$

où Q est le quotient de la division euclidienne de S par T .

- Soit $F \in A[X]$ avec $F(0) = 1$ et soit $\ell \geq 1$. Considérons la suite de polynômes $G_i \in A[X]$ définie par $G_0 = 1$ et

$$G_{i+1} = 2G_i - F \cdot G_i^2 \bmod X^{2^{i+1}}$$

pour $i \geq 0$. Montrer que pour tout entier $i \geq 0$, nous avons

$$F \cdot G_i \equiv 1 \bmod X^{2^i}$$

- En déduire un algorithme pour calculer Q et le reste R de la division euclidienne de S par T .
- Montrer que la complexité arithmétique de cet algorithme de division euclidienne appliquée à deux polynômes de degrés $< n$ est en $O(M(n))$ où $M(n)$ est la complexité arithmétique du produit de deux polynômes de degré $< n$ de $A[X]$ (avec $M(n_1 + n_2) \geq M(n_1) + M(n_2)$ pour $n_1, n_2 \in \mathbb{N}$).

Solution

- L'algorithme naïf pour calculer Q et R tels que $S = QT + R$ avec $\deg R < \deg T$ consiste à poser la division. Pour cela les termes dominants sont d'abord isolés :

$$S = aX^n + R_S, T = X^m + R_T \text{ avec } \deg R_S < n \text{ et } \deg R_T < m$$

6.3. Multi-évaluation de polynômes et algorithme de Pollard-Strassen

Si $n < m$, le résultat est immédiat. Sinon la boucle élémentaire de la division de S par T consiste à éliminer le terme de plus haut degré de S en effectuant la soustraction

$$S - aX^\delta T = (aX^n + R_S) - (aX^{\delta+m} + aX^\delta R_T) = R_S - aX^\delta R_T$$

où $\delta = m - n$. Le polynôme obtenu est congru à S modulo T , mais de degré strictement inférieur à n . Il n'y a plus qu'à itérer ce procédé jusqu'à obtenir un polynôme de degré strictement inférieur à n et à garder en mémoire les quotients successifs.

La complexité de cet algorithme est facile à estimer. La boucle élémentaire s'effectue en $O(m)$ opérations dans A ; dans le pire des cas, l'algorithme effectue $(n - m + 1)$ passages dans cette boucle, de sorte que la complexité de la division de S par T est de $O(m(n - m))$ opérations dans A .

2. Nous avons

$$S(X) = Q(X)T(X) + R(X)$$

soit

$$X^n S(1/X) = X^n(Q(1/X)T(1/X) + R(1/X))$$

Nous obtenons donc

$$X^n S(1/X) = [X^{n-m}Q(1/X)][X^m T(1/X)] + X^{n-m+1}[X^{m-1}R(1/X)]$$

et le résultat.

3. Montrons le résultat par récurrence.

- Pour $i = 0$, la formule se vérifie aisément.
- Supposons la formule démontrée un entier $i \in \mathbb{N}$ (*i.e.* nous avons $F \cdot G_i = 1 + P(X)X^{2^i}$). Nous montrons qu'elle est également satisfaite pour $i + 1$ de la façon suivante :

$$\begin{aligned} F \cdot G_{i+1} &\equiv F \cdot (2G_i - F \cdot G_i^2) \bmod X^{2^{i+1}} \\ &\equiv 2 + 2P(X)X^{2^i} - (1 + P(X)X^{2^i})^2 \bmod X^{2^{i+1}} \\ &\equiv 1 + P(X)^2 X^{2^{i+1}} \bmod X^{2^{i+1}} \end{aligned}$$

Par récurrence, nous obtenons le résultat.

4. Le polynôme $\text{Rec}(T)$ vérifie les conditions de l'énoncé (puisque T est unitaire). La formule précédente permet de calculer l'inverse de $\text{Rec}(T)$ modulo X^{n-m+1} en l'appliquant de façon itérative ℓ fois (avec $2^\ell > n - m + 1$). En appliquant la question 2, nous obtenons $\text{Rec}(Q)$ et donc Q puis R , le quotient et le reste de la division euclidienne de S par T , avec deux multiplications supplémentaires.
5. À chaque étape de la boucle, nous effectuons deux multiplications de polynômes de degré au plus 2^i pour $i \in \{0, \dots, \ell\}$. Donc le nombre d'opérations arithmétiques dans A est majoré par :

$$2M(1) + 2M(2) + 2M(4) + \cdots + 2M(2^\ell)$$

où $2^\ell > m-n+1$ et $M(n)$ est la complexité arithmétique du produit de deux polynômes de degré inférieur à n de $A[X]$. Puisque nous supposons que $M(n_1 + n_2) \geq M(n_1) + M(n_2)$ pour $n_1, n_2 \in \mathbb{N}$, nous obtenons que le coût de la division euclidienne appliquée à deux polynômes de degrés $< n$ est en $O(M(n))$.

En particulier, si $A = \mathbb{Z}$ ou $A = (\mathbb{Z}/N\mathbb{Z})$, en utilisant des techniques de multiplication comme la transformée de Fourier rapide, nous obtenons que le coût de la division euclidienne appliquée à deux polynômes de degrés $< n$ est en $O(n \log n)$ opérations arithmétiques dans A .

Exercice 6.16 Multi-évaluation d'un polynôme univarié

Soit A un anneau commutatif unitaire (en pratique nous considérerons $A = \mathbb{Z}$ ou $A = (\mathbb{Z}/N\mathbb{Z})$). Soient $a_1, \dots, a_d \in A$. Nous notons $M(d)$ la complexité arithmétique du produit et de la division euclidienne de deux polynômes de degré $< d$ de $A[X]$ (avec $M(d_1 + d_2) \geq M(d_1) + M(d_2)$ pour $d_1, d_2 \in \mathbb{N}$).

1. Supposons que $d = 2^k$ est une puissance de 2 et considérons l'arbre binaire complet T à n feuilles défini par :

- chacune des d feuilles est associée à un polynôme $X - a_j$ pour $j \in \{1, \dots, d\}$;
- pour chaque nœud interne u ayant les fils v et w associés aux polynômes $M_v(X)$ et $M_w(X)$ respectivement, u est associé au polynôme $M_u(X) = M_v(X) \cdot M_w(X)$.

Donner un algorithme pour construire l'arbre T avec une complexité arithmétique en $O(M(d) \log d)$.

2. Soit $P \in A[X]$ unitaire avec $\deg(P) = d$. Donner un algorithme qui prenant en entrée $P, (a_1, \dots, a_d)$ et l'arbre T , calcule $P(X) \bmod M_u(X)$ pour tout $u \in T$, avec une complexité arithmétique en $O(M(d) \log d)$.
3. Déduire des questions précédentes un algorithme qui calcule $P(a_1), \dots, P(a_d)$ pour tout $d \in \mathbb{N}$ avec une complexité arithmétique en $O(M(d) \log d)$.

Solution

1. La construction de l'arbre se fait simplement des feuilles vers la racine. L'algorithme place aux feuilles les polynômes polynôme $A_j = X - a_j$ pour $j \in \{1, \dots, d\}$. Chaque nœud au dessus de ces feuilles est obtenu en multipliant deux polynômes de degré 1 : $(X - a_{2j+1})(X - a_{2j+2})$ pour $j \in \{0, \dots, (d-2)/2\}$, ce qui requiert le calcul de $d/2$ multiplications de polynômes de degré 1. Le niveau suivant de l'arbre s'obtient en calculant $d/4$ multiplications de polynômes de degré 2 et nous continuons comme

6.3. Multi-évaluation de polynômes et algorithme de Pollard-Strassen

illustré sur la figure 6.2 jusqu'à multiplier deux polynômes de degré $d/2$ pour obtenir la racine.

La complexité de l'algorithme est donc

$$\frac{d}{2}M(1) + \frac{d}{4}M(2) + \dots + 2M\left(\frac{d}{4}\right) + M\left(\frac{d}{2}\right)$$

Il s'agit donc d'un algorithme classique de type « diviser pour régner » et nous obtenons un coût total égal à $O(M(d) \log d)$ opérations dans A .

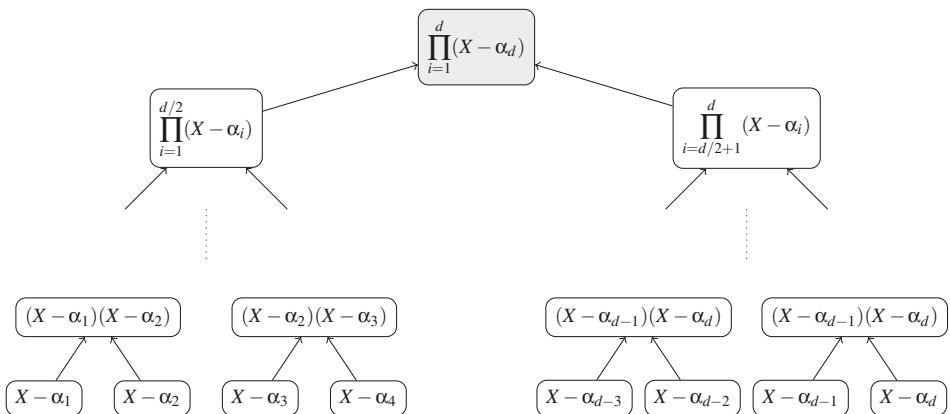


Figure 6.2 – Arbre des produits de polynôme $M_u(X)$ pour $u \in T$

2. Supposons construit l'arbre des produits de polynôme $M_u(X)$ pour $u \in T$ de la question précédente. Pour calculer les valeurs de $P(X) \bmod M_u(X)$ pour tout $u \in T$, l'algorithme consiste à effectuer les divisions euclidiennes de P par les polynômes M_u de l'arbre en partant de la racine et en descendant vers les feuilles (cf. Figure 6.3). La division de P par $(X - a_1) \dots (X - a_d)$ est de complexité arithmétique $M(d)$; en divisant le résultat obtenu par les deux polynômes situés aux deux fils de la racine, nous obtenons $P \bmod (X - a_1) \dots (X - a_{d/2})$ et $P \bmod (X - a_{d/2+1}) \dots (X - a_d)$ en $2M(d/2)$ opérations arithmétiques.

De proche en proche nous obtenons la valeur de P modulo tous les polynômes situés à profondeur h dans l'arbre en $2^h M(d/2^h)$ opérations arithmétiques. Comme dans la question précédente, nous obtenons que le coût de l'algorithme est égal à $O(M(d) \log d)$ opérations dans A .

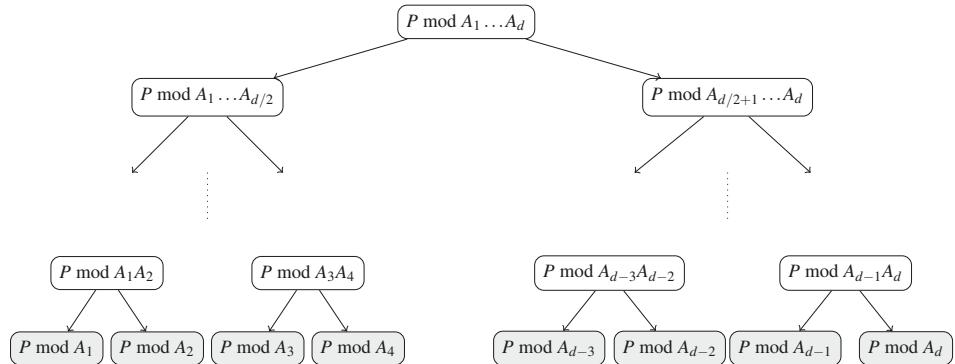


Figure 6.3 – Arbre d'évaluation

3. Les valeurs obtenues aux feuilles de l'arbre précédent sont égales au polynôme $P(X) \bmod X - a_i$ pour $i \in \{1, \dots, d\}$. Il s'agit donc des valeurs $P(a_i) \in A$ et leur calcul simultané requiert $O(M(d) \log d)$ opérations arithmétiques dans A . Dans le cas d'un entier d qui n'est pas une puissance de 2, il suffit d'appliquer l'algorithme précédent avec le plus petit entier k tel que $2^k \geq d$ et, comme $2^k < 2d$, nous obtenons l'algorithme général.

La méthode Pollard-Strassen pour factoriser un entier N utilise de façon presque immédiate l'algorithme précédent dans l'anneau $A = (\mathbb{Z}/N\mathbb{Z})$.

Exercice 6.17 Algorithme de Pollard-Strassen

Soit N un entier positif composé et soit $d = \lfloor N^{1/4} \rfloor$.

- Montrer que $\text{pgcd}((d^2)!, N) \neq 1$.
- En considérant le polynôme $P(X) = (dX + 1)(dX + 2) \dots (dX + d) \in (\mathbb{Z}/N\mathbb{Z})[X]$, montrer comment calculer $(d^2)!$ mod N en $O(M(d) \log d)$ opérations arithmétiques dans l'anneau $(\mathbb{Z}/N\mathbb{Z})$.
- En déduire un moyen de trouver un diviseur propre de N en $O(N^{1/4+o(1)})$ opérations élémentaires.

Solution

- Puisque N est composé, il possède au moins un diviseur non trivial inférieur à \sqrt{N} et donc inférieur à d^2 . Ce diviseur divise également $(d^2)!$ d'où le résultat.

6.3. Multi-évaluation de polynômes et algorithme de Pollard-Strassen

2. Le produit $(d^2)!$ mod N peut s'obtenir comme le produit des d valeurs suivantes modulo N :

$$a_1 = 1 \cdot 2 \cdots (d-1)d = d!$$

$$a_2 = (d+1) \cdot (d+2) \cdots (2d-1) \cdot 2d = (2d)!/d!$$

\vdots

$$a_d = ((d-1)d+1)((d-1)d+2) \cdots (d^2-1) \cdot d^2 = (d^2)!/(d^2-d)!$$

Chaque valeur a_i s'obtient comme $a_i = P(i-1) \bmod N$ pour $i \in \{1, \dots, d\}$. Il suffit donc d'appliquer l'algorithme de multi-évaluation de l'exercice précédent. Le polynôme P se construit par l'arbre des produits de polynômes de l'exercice précédent et son évaluation aux valeurs $0, 1, \dots, d-1$ se fait en appliquant l'algorithme de multi-évaluation. Le calcul de $(d^2)!$ mod N s'obtient alors comme le produit $a_1 \cdot a_2 \cdots a_d$ mod N . La complexité de l'algorithme est donc de l'ordre de $O(M(d) \log d)$ opérations dans $\mathbb{Z}/N\mathbb{Z}$.

3. Nous savons que $\text{pgcd}((d^2)!, N) \neq 1$.

- Si $\text{pgcd}((d^2)!, N) \neq N$, nous avons trouvé un diviseur propre de N .
- Si $\text{pgcd}((d^2)!, N) = N$, nous pouvons construire l'arbre de produits formés des valeurs a_i pour $i \in \{1, \dots, d\}$ (éventuellement complétées de valeurs à 1 pour construire un arbre binaire complet). Nous plaçons les a_i aux feuilles de l'arbre et en chaque noeud de l'arbre le produit de ses deux fils modulo N (de sorte que la racine contiennent le produit $(d^2)! = a_1 \cdot a_2 \cdots a_d \bmod N$). Nous pouvons alors calculer le pgcd de N avec les étiquettes des nœuds de l'arbre en descendant dans une branche où le pgcd vaut N . Si nous obtenons un pgcd différent de N , nous avons trouvé un diviseur propre de N .
- Si nous arrivons à une feuille de l'arbre telle que $\text{pgcd}(a_i, N) = N$ pour un certain $i \in \{1, \dots, d\}$, nous avons

$$a_i = (id+1)(id+2) \cdots (id+d-1) \cdot (id+d) \bmod N$$

Nous pouvons alors calculer de façon systématique les d plus grands communs diviseurs $\text{pgcd}(N, id+j)$ pour $j \in \{1, \dots, d\}$ et l'un deux doit révéler un diviseur propre de N .

En utilisant de l'arithmétique rapide pour les opérations sur les polynômes et les opérations sur les entiers (i.e. multiplication et pgcd), l'algorithme requiert $O(N^{1/4+o(1)})$ opérations élémentaires.

6.4 RACINE CARRÉE MODULAIRE ET FACTORISATION

Dans la section 6.1, nous avons utilisé les propriétés du symbole de Legendre pour tester la primalité d'un entier. Dans cette section, nous allons étudier le problème calculatoire associé (*i.e.* l'extraction de racines carrées modulaires) pour proposer notamment un algorithme de factorisation en temps sous-exponentiel.

Le premier exercice montre qu'il existe un algorithme polynomial probabiliste pour calculer une racine carrée modulo un nombre premier p .

Exercice 6.18 Extraction de racine carrée modulo p

Soit p un nombre premier impair.

1. Nous supposons que $p \equiv 3 \pmod{4}$. Donner un algorithme de complexité $O(\log^3 p)$ opérations binaires qui, étant donné $\alpha \in \{1, \dots, p-1\}$ tel que $\left(\frac{\alpha}{p}\right) = 1$, retourne $\beta \in \{1, \dots, p-1\}$ tel que $\beta^2 \equiv \alpha \pmod{p}$.

Nous supposons désormais que $p \equiv 1 \pmod{4}$. Posons $p = 2^h m + 1$ avec m impair.

2. Donner un algorithme probabiliste qui étant donné p retourne un élément γ de $\{1, \dots, p-1\}$ tel que $\left(\frac{\gamma}{p}\right) = -1$ en temps espéré $O(\log^2 p)$ opérations binaires. Montrer que pour un tel γ , l'élément $\delta = \gamma^m$ engendre l'unique sous-groupe d'ordre 2^h de $(\mathbb{Z}/p\mathbb{Z})^*$.
3. Soit $\alpha \in \{1, \dots, p-1\}$ tel que $(\alpha|p) = 1$. Montrer que α^m appartient au sous-groupe engendré par δ et en utilisant un algorithme de logarithme discret, en déduire un algorithme pour calculer une racine carrée de α^m modulo p .
4. Conclure en donnant un algorithme permettant de calculer les racines carrées de α en temps $O((\log p)^3)$.

Solution

1. Par hypothèse, nous avons $p \equiv 3 \pmod{4}$. Par le critère d'Euler, nous savons que tout élément α de $(\mathbb{Z}/p\mathbb{Z})^*$ tel que $\left(\frac{\alpha}{p}\right) = 1$ vérifie

$$\alpha^{(p-1)/2} \equiv 1 \pmod{p} \quad \text{soit} \quad \alpha^{(p+1)/2} = \alpha^{(p-1)/2} \cdot \alpha \equiv \alpha \pmod{p}$$

Comme $(p+1)$ est divisible par 4 par hypothèse, nous obtenons alors

$$\left(\alpha^{(p+1)/4}\right)^2 = \alpha^{(p+1)/2} = \alpha \pmod{p}$$

et $\beta = \alpha^{(p+1)/4}$ est une racine carrée de α que l'on peut calculer en $O(\log p)$ multiplications dans $(\mathbb{Z}/p\mathbb{Z})$ en appliquant l'algorithme d'exponentiation dichotomique 5.1 (et donc $O(\log^3 p)$ opérations binaires).

2. L'algorithme probabiliste consiste simplement à tirer γ uniformément aléatoirement dans $(\mathbb{Z}/p\mathbb{Z})^*$ et à vérifier que $\left(\frac{\gamma}{p}\right) = -1$. Le calcul du symbole de Legendre en utilisant la loi de réciprocité quadratique est de complexité quadratique et le nombre de γ à tester avant de trouver un non-résidu quadratique est égal à 2 (puisque $(\mathbb{Z}/p\mathbb{Z})^*$ contient $(p-1)/2$ non-résidus quadratiques).

Un tel γ vérifie $\gamma^{(p-1)/2} \equiv -1 \pmod{p}$ par le critère d'Euler, donc

$$(\gamma^m)^{2^{h-1}} \equiv \gamma^{2^{h-1}m} \equiv -1 \pmod{p}$$

et γ^m est donc un générateur de l'unique sous-groupe d'ordre 2^h de $(\mathbb{Z}/p\mathbb{Z})^*$.

3. Nous avons $(\alpha^m)^{2^h} \equiv \alpha^{(p-1)} \equiv 1 \pmod{p}$, donc α^m appartient à l'unique sous-groupe d'ordre 2^h de $(\mathbb{Z}/p\mathbb{Z})^*$. Il existe donc un élément $x \in \{0, \dots, 2^h - 1\}$ tel que $\alpha^m = \delta^x \pmod{p}$. Nous avons

$$1 = \left(\frac{\alpha}{p}\right) = \left(\frac{\alpha^m}{p}\right) = \left(\frac{\delta^x}{p}\right) = \left(\frac{\delta}{p}\right)^x = (-1)^x$$

donc ce x est nécessairement pair.

Lorsque cette valeur x est connue, nous avons $\alpha^{(m+1)} \equiv \delta^x \alpha$ et

$$(\alpha^{(m+1)/2} \delta^{-x/2})^2 = (\delta^x \alpha) \delta^{-x} = \alpha$$

Donc $\beta = \alpha^{(m+1)/2} \delta^{-x/2}$ est une racine carrée de α .

4. Une fois la valeur de δ construite, il suffit de calculer le logarithme discret x ce qui peut être fait soit en appliquant l'algorithme de Pohlig-Hellman 5.5 en $O(h(\log(2) + \log(p)))$ opérations de groupe dans $(\mathbb{Z}/p\mathbb{Z})^*$. Nous obtenons ainsi l'algorithme 6.9 de complexité annoncée (qui est dû à A. TONELLI [64] et D. SHANKS [60]).

Algorithme 6.9 Algorithme de Tonelli-Shanks

ENTRÉE: $p = 2^h m + 1 \in \mathbb{P}$, $\alpha \in (\mathbb{Z}/p\mathbb{Z})^*$ avec $\left(\frac{\alpha}{p}\right) = 1$, $\gamma \in (\mathbb{Z}/p\mathbb{Z})^*$ avec $\left(\frac{\gamma}{p}\right) = -1$.

SORTIE: $\beta \in (\mathbb{Z}/p\mathbb{Z})^*$ tel que $\beta^2 \equiv \alpha \pmod{p}$

```

 $\delta \leftarrow \gamma^m$ 
 $\beta \leftarrow \alpha^m$ 
 $t \leftarrow 0$ 
pour  $i$  de 0 à  $h-1$  faire
  si  $(\beta \delta^t)^{2^{h-1-i}} \equiv -1 \pmod{p}$  alors
     $t \leftarrow t + 2^i$ 
  fin si
fin pour
retourner  $\alpha^{(m+1)/2} \delta^{t/2} \pmod{p}$ 
```

L'exercice suivant montre qu'il existe également un algorithme polynomial probabiliste pour calculer une racine carrée modulo une puissance d'un nombre premier.

Exercice 6.19 Extraction de racine carrée modulo p^ℓ

Soit p un nombre premier impair et $\ell \geq 2$ un entier.

- Montrer qu'un entier x premier avec p vérifie $x^2 \equiv 1 \pmod{p^\ell}$ si et seulement si $x \equiv \pm 1 \pmod{p^\ell}$. En déduire un critère d'Euler généralisé.
- Montrer qu'un entier x premier avec p est un carré modulo p si et seulement si x est un carré modulo p^ℓ .
- Donner un algorithme pour calculer une racine carrée de x modulo p^ℓ .

Indication. On pourra commencer par calculer une racine carrée de x modulo p puis calculer récursivement une racine carrée de x modulo p^{i+1} à partir d'une racine carrée de x modulo p^i pour tout entier $i \in \{1, \dots, \ell - 1\}$.

Solution

- Si $x \equiv \pm 1 \pmod{p^\ell}$, nous avons immédiatement $x^2 \equiv 1 \pmod{p^\ell}$. Réciproquement, si $x^2 \equiv 1 \pmod{p^\ell}$, nous avons que p^ℓ divise $x^2 - 1 = (x - 1)(x + 1)$. En particulier, p divise $x - 1$ ou p divise $x + 1$. Le nombre premier p ne peut pas diviser ces deux valeurs (sinon il diviserait leur différence $(x + 1) - (x - 1) = 2$). L'entier p^ℓ divise donc $(x - 1)$ ou $(x + 1)$, d'où le résultat.
Puisque 1 et -1 sont les seules racines carrées de 1 modulo p^ℓ , nous obtenons donc immédiatement un critère d'Euler généralisé : $x \in (\mathbb{Z}/p^\ell \mathbb{Z})^*$ est un carré modulo p^ℓ si et seulement si $x^{\varphi(p^\ell)/2} \equiv 1 \pmod{p^\ell}$.
- Si x est un carré modulo p^ℓ , alors il existe un entier a tel que $x \equiv a^2 \pmod{p^\ell}$. En réduisant cette égalité, modulo p , nous avons $x \equiv a^2 \pmod{p}$ et x est un carré modulo p .
Si x n'est pas un carré modulo p^ℓ , d'après le critère d'Euler généralisé, nous avons

$$x^{\varphi(p^\ell)/2} = x^{p^{\ell-1}(p-1)/2} \equiv -1 \pmod{p^\ell}$$

En réduisant cette égalité modulo p , nous obtenons $x^{p^{\ell-1}(p-1)/2} \equiv -1 \pmod{p}$ et en appliquant le petit théorème de Fermat $\ell - 1$ fois :

$$x \equiv x^p \equiv x^{p^2} \equiv \cdots \equiv x^{p^{\ell-1}} \pmod{p}$$

nous obtenons

$$\left(\frac{x}{p}\right) \equiv x^{(p-1)/2} \equiv x^{p^{\ell-1}(p-1)/2} \equiv -1 \pmod{p}$$

et x n'est pas un carré modulo p .

3. Supposons connue une racine carrée y_i de x modulo p^i avec $i \in \{1, \dots, \ell - 1\}$. Nous avons $y_i^2 \equiv x \pmod{p^i}$ et nous pouvons donc chercher une valeur $y_{i+1} \in \mathbb{Z}$ telle que $y_{i+1}^2 \equiv x \pmod{p^{i+1}}$. En réduisant modulo p^i , nous obtenons $y_{i+1}^2 \equiv y_i^2 \pmod{p^i}$ et d'après la question 1, nous avons $y_{i+1} \equiv \pm y_i \pmod{p^i}$. Nous cherchons donc $y_{i+1} = y_i + k_{i+1}p^i$ avec k_{i+1} un entier. Nous avons

$$x \equiv y_{i+1}^2 \equiv (y_i + k_{i+1}p^i)^2 \equiv y_i^2 + 2k_{i+1}y_i p^i + k_{i+1}^2 p^{2i} \equiv y_i^2 + 2k_{i+1}y_i p^i \pmod{p^{i+1}}$$

Nous cherchons donc un entier k_{i+1} tel que $(2y_i p^i)k_{i+1} \equiv x - y_i^2 \pmod{p^{i+1}}$. Le nombre premier p ne divise pas $2y_i$, donc le plus grand diviseur commun de $(2y_i p^i)$ et p^{i+1} est égal à p^i et l'équation admet bien une solution k_i (obtenue aisément en utilisant l'algorithme d'Euclide étendu).

L'algorithme pour calculer une racine carrée de x modulo p^ℓ consiste donc à calculer la racine carrée de x modulo p à l'aide de l'algorithme de Tonelli-Shanks puis à calculer successivement les valeurs y_{i+1} et k_{i+1} en utilisant la remarque précédente.

L'exercice suivant montre que le problème de la factorisation d'un entier N et l'extraction de racines carrées modulo N sont de difficulté équivalente.

Exercice 6.20 Extraction de racine carrée modulo N

Soit N un entier dont la décomposition en facteurs premiers est $N = q_1^{f_1} \dots q_d^{f_d}$ où $q_i \in \mathbb{P}$ sont des nombres premiers deux à deux distincts et $f_i \geq 1$ pour $i \in \{1, \dots, d\}$.

- Montrer qu'un entier x est un carré modulo N dès que tous les symboles de Legendre $\left(\frac{x}{q_j}\right)$ sont égaux à 1 pour $j \in \{1, \dots, d\}$.
- Montrer qu'un tel carré modulo N a exactement 2^d racines carrées.
- Montrer que s'il existe un algorithme \mathcal{A} capable d'extraire des racines carrées dans $(\mathbb{Z}/N\mathbb{Z})$ en temps τ , alors il existe un algorithme \mathcal{B} qui retourne un diviseur propre de N en temps espéré $O(\tau \cdot (1 - 2^{1-d}))$.

Solution

- D'après l'exercice précédent, x est un carré modulo $q_i^{f_i}$ si et seulement si x est un carré modulo q_i , c'est-à-dire si et seulement le symbole de Legendre $\left(\frac{x}{q_j}\right) = 1$. Si tous les symboles de Legendre $\left(\frac{x}{q_j}\right)$ sont égaux à 1 pour $j \in \{1, \dots, d\}$, il existe donc $y_i \in (\mathbb{Z}/q_i^{f_i}\mathbb{Z})$ pour $i \in \{1, \dots, d\}$ tels que $x \equiv y_i^2 \pmod{q_i^{f_i}}$. Par le théorème chinois des restes, il existe un entier y tel que $y \equiv y_i \pmod{q_i^{f_i}}$ pour tout $i \in \{1, \dots, d\}$. Cet élément vérifie $y^2 \equiv x \pmod{q_i^{f_i}}$ pour tout $i \in \{1, \dots, d\}$ et donc $y^2 \equiv x \pmod{N}$.

2. En suivant la question précédente, il est possible de construire une racine carrée de x à partir des y_i pour $i \in \{1, \dots, d\}$. Pour tout suite $\alpha = (\alpha_1, \dots, \alpha_d) \in \{-1, 1\}^d$, notons $y_\alpha \in (\mathbb{Z}/N\mathbb{Z})$ tel que $y_\alpha \equiv \alpha_i y_i \pmod{q_i^{f_i}}$ pour tout $i \in \{1, \dots, d\}$. Ces 2^d éléments sont deux à deux distincts (puisque distincts modulo $q_i^{f_i}$ pour au moins une valeur de $i \in \{1, \dots, d\}$) et sont tous des racines carrées de x modulo N .
3. L'algorithme \mathcal{B} consiste simplement à tirer uniformément aléatoirement un élément $y \in (\mathbb{Z}/N\mathbb{Z})$ et à calculer $x = y^2 \pmod{N}$ et à rechercher une racine carrée de x modulo N en appliquant l'algorithme \mathcal{A} . L'algorithme \mathcal{A} va retourner une valeur $z \in (\mathbb{Z}/N\mathbb{Z})$ tel que $z^2 \equiv x \pmod{N}$.

Puisque $z^2 \equiv y^2 \pmod{N}$, N divise $(z - y)(z + y)$. S'il existe un diviseur de N qui divise $(z \pm y)$ mais pas $(z \mp y)$, alors $\text{pgcd}(N, z \pm y)$ est un diviseur propre de N . Nous avons $z \equiv \pm y \pmod{q_i^{f_i}}$ pour tout $i \in \{1, \dots, d\}$ si et seulement si $z \equiv \pm y \pmod{N}$. La probabilité que $z \equiv \pm y \pmod{N}$ est égal à 2^{1-d} puisque x possède 2^d racines carrées modulo N .

En calculant les deux plus grands diviseurs communs $\text{pgcd}(N, z - y)$ et $\text{pgcd}(N, z + y)$, l'algorithme obtient donc un diviseur propre de N avec probabilité $1 - 2^{1-d} \geq 1/2$. En répétant cette méthode jusqu'à obtenir un diviseur, nous obtenons un algorithme \mathcal{B} qui retourne un diviseur propre de N en temps espéré $O(\tau)$.

De nombreuses méthodes modernes de factorisation d'un entier N reposent sur la recherche d'une congruence de carrés. Une méthode naïve de recherche d'une telle congruence est de tirer aléatoirement des valeurs $x \in (\mathbb{Z}/N\mathbb{Z})$ et d'espérer obtenir une collision de la forme :

$$x^2 \equiv y^2 \pmod{N} \quad x \not\equiv \pm y \pmod{N}$$

Cette méthode naïve est inefficace mais il est possible d'accélérer (par un calcul d'indice) la recherche de cette congruence de carrés en construisant des carrés modulaires *friables*. La méthode utilise une base de facteurs $\mathcal{B} = \{2, 3, 5, \dots, p_k\}$ constitués des k premiers nombres premiers comme dans l'algorithme de calcul d'indice de Kraitchik pour le problème du logarithme discret. L'idée générale (due à J. D. DIXON [21]) est la suivante :

- dans une première étape, l'algorithme recherche des relations de la forme

$$x_i^2 \equiv p_1^{e_{i,1}} \cdots p_m^{e_{i,m}} \pmod{N} \tag{6.4}$$

avec $x_i \in (\mathbb{Z}/N\mathbb{Z})$

- dans une seconde étape, l'algorithme cherche à combiner ces relations (en appliquant des techniques d'algèbre linéaire modulo 2) pour obtenir une relation de la forme $x^2 \equiv y^2 \pmod{N}$ et ainsi trouver un diviseur non trivial de N avec une probabilité supérieure à $1/2$ d'après l'exercice précédent.

L'analyse de la complexité de la méthode de Dixon est similaire à celle de la méthode de Kraitchik, la seule différence est de calculer le nombre d'éléments x_i vérifiant la relation (6.4). Un élément p_m -friable inférieur à \sqrt{N} vérifie trivialement (6.4), mais si nous ne considérons que ces entiers, la complexité de la méthode est très mauvaise. L'exercice suivant donne une estimation plus fine de ce cardinal.

Problème 6.21 Carrés modulaires friables

Soit N un entier dont la décomposition en facteur premiers est $N = q_1^{f_1} \dots q_d^{f_d}$ (où les $q_i \in \mathbb{P}$ sont des nombres premiers deux à deux distincts et $f_i \geq 1$ pour $i \in \{1, \dots, d\}$). Notons $(p_n)_{n \geq 1}$ la suite des nombres premiers (avec $p_1 = 2, p_2 = 3, \dots$). Soit $m \geq 1$ un entier et posons s le plus grand entier pair inférieur ou égal à $\ln N / \ln p_m$ (*i.e.* $s = 2\lfloor \ln N / 2 \ln p_m \rfloor$). Le but de l'exercice est de minorer le cardinal de l'ensemble

$$\left\{ x \in (\mathbb{Z}/N\mathbb{Z})^* \mid x^2 \bmod N \text{ est } p_m\text{-friable} \right\}$$

Pour un élément x de cet ensemble, il existe une suite d'entiers (e_1, \dots, e_m) telle que $x^2 \equiv p_1^{e_1} \dots p_m^{e_m} \bmod N$.

1. Soit $e = (e_1, \dots, e_m)$ une suite finie d'entiers telle que $e_1 + \dots + e_m \leq s$. Montrer que le nombre de façons d'écrire cette suite comme somme terme à terme de deux suites finies $f = (f_1, \dots, f_m)$ et $g = (g_1, \dots, g_m)$ avec $f_1 + \dots + f_m \leq s/2$ et $g_1 + \dots + g_m \leq s/2$ est majoré par $\binom{s}{s/2}$.

Indication

Comme dans l'exercice 5.6 (page 155), on pourra représenter la suite e en coloriant les entiers de 1 à $m+s$ en deux couleurs : les e_1 premiers en blanc, suivi d'un noir, suivi de e_2 blancs et ainsi de suite puis colorier $s/2$ entiers blancs en gris pour représenter la suite f .

2. À toute suite finie d'entiers $f = (f_1, \dots, f_m)$ telles que $f_1 + \dots + f_m \leq s/2$, on associe l'entier $U(f) = p_1^{f_1} \dots p_m^{f_m} \bmod N$ et la suite des symboles de Legendre

$$\lambda(f) = \left(\left(\frac{U(f)}{q_1} \right), \dots, \left(\frac{U(f)}{q_d} \right) \right)$$

Montrer que si f et g sont deux suites finies d'entiers telles que $\lambda(f) = \lambda(g)$ alors $U(f) \cdot U(g)$ est un carré modulo N .

3. Pour toute suite $a \in \{-1, +1\}^d$, nous notons n_a le nombre de suite finie d'entiers $h = (h_1, \dots, h_m)$ telles que $h_1 + \dots + h_m \leq s/2$ et $\lambda(h) = a$. Montrer que le nombre d'entiers $x \in [1, N-1]$ tel que $x^2 \bmod N$ soit m -friable est minoré par

$$\frac{2^d}{\binom{s}{s/2}} \sum_{a \in \{-1, +1\}^d} (n_a)^2$$

4. En appliquant l'inégalité de Cauchy-Schwarz, en déduire que la probabilité qu'un entier de $[1, N - 1]$ tiré uniformément aléatoirement produise un carré p_m -friable modulo N est minorée par $m^s/Ns!$.

Solution

1. Considérons le cas $s = 6$ et $m = 3$ et la partition $1 + 3 + 1 = 5 \leq 6$. En suivant l'indication, nous obtenons la représentation suivante :

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \circ & \bullet & \circ & \circ & \circ & \bullet & \circ & \bullet & \circ \end{array}$$

En coloriant $s/2 = 3$ entiers blancs en gris, nous obtenons effectivement toutes les décompositions possibles

$$f \in \{(0, 1, 1), (0, 2, 0), (0, 2, 1), (0, 3, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1), (1, 2, 0)\}$$

et $g = e - f$:

	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9	
(0, 1, 1)	○	●	○	○	○	●	○	●	○		○	●	○	○	○	●	○	●	○	(1, 0, 1)
(0, 1, 1)	○	●	○	○	○	●	○	●	○		○	●	○	○	○	●	○	●	○	(1, 1, 0)
(0, 2, 0)	○	●	○	○	○	○	●	○	●		○	●	○	○	○	●	○	●	○	(1, 1, 1)
(0, 2, 1)	○	●	○	○	○	○	●	○	●		○	●	○	○	○	●	○	●	○	(1, 1, 0)
(0, 1, 1)	○	●	○	○	○	○	●	○	●		○	●	○	○	○	●	○	●	○	(1, 1, 1)
(0, 2, 0)	○	●	○	○	○	○	●	○	●		○	●	○	○	○	●	○	●	○	(1, 2, 0)
(0, 2, 1)	○	●	○	○	○	○	●	○	●		○	●	○	○	○	●	○	●	○	(1, 1, 0)
(0, 2, 0)	○	●	○	○	○	○	●	○	●		○	●	○	○	○	●	○	●	○	(1, 1, 1)
(0, 2, 1)	○	●	○	○	○	○	●	○	●		○	●	○	○	○	●	○	●	○	(1, 2, 0)
(0, 3, 0)	○	●	○	○	○	○	●	○	●		○	●	○	○	○	●	○	●	○	(1, 2, 0)

Plus généralement, considérons un ensemble $A = \{a_1, \dots, a_s\} \subset \{1, \dots, s+m\}$ à s éléments avec $a_1 < a_2 < \dots < a_s$ (correspondant aux entiers coloriés en blanc).

Considérons l'ensemble $E_{s/2}$ constitués des sous-ensembles de A à $s/2$ éléments. En notant $a_0 = 0$, nous avons $a_{i+1} - a_i \in \{1, 2\}$ pour tout $i \in \{0, \dots, s-1\}$.

Nous appelons *bloc* de A les m sous-suites de A formées d'entiers consécutifs. Notons θ l'application qui associe à un élément B de $E_{s/2}$ (correspondant aux entiers coloriés en gris) la suite d'entiers de \mathbb{N}^m formée du nombres d'entiers de B contenus dans chaque bloc. En notant $f = (f_1, \dots, f_m) \in \mathbb{N}^m$ l'image par θ d'un élément de $E_{s/2}$, nous obtenons immédiatement $f_1 + \dots + f_m \leq s/2$. De plus, en notant $e \in \mathbb{N}^m$ le nombre d'éléments dans chaque bloc et $g = e - f = (g_1, \dots, g_m)$, nous avons $g_1 + \dots + g_m \leq s/2$.

L'application θ est clairement surjective sur l'ensemble des suites f vérifiant ces conditions et le nombre de façons de décomposer e est donc bien majoré par $\#E_{s/2} = \binom{s}{s/2}$.

2. Soient f et g sont deux suites finies d'entiers telles que $\lambda(f) = \lambda(g)$. Par multiplicativité du symbole de Legendre, nous avons pour tout nombre premier q_i ,

$$\left(\frac{U(f) \cdot U(g)}{q_i} \right) = \left(\frac{U(f)}{q_i} \right) \left(\frac{U(g)}{q_i} \right) = \left(\frac{U(f)}{q_i} \right)^2 = 1$$

L'entier $U(f) \cdot U(g)$ est donc un carré modulo tous les diviseurs premiers de N . D'après l'exercice précédent, $U(f) \cdot U(g)$ est donc un carré modulo N .

3. Considérons deux entiers m -friables de la forme

$$U(f) = p_1^{f_1} \cdots p_m^{f_m} \text{ et } U(g) = p_1^{g_1} \cdots p_m^{g_m}$$

avec $f_1 + \cdots + f_m \leq s/2$ et $g_1 + \cdots + g_m \leq s/2$. D'après la question précédente, si $\lambda(f) = \lambda(g)$, l'entier $U(f)U(g)$ est un carré modulo N . D'après l'exercice précédent, chacun de ces entiers possède 2^d racines carrées distinctes modulo N .

De plus,

$$U(f) \cdot U(g) = p_1^{f_1+g_1} \cdots p_m^{f_m+g_m} = p_1^{e_1} \cdots p_m^{e_m}$$

avec $e_1 + \cdots + e_m \leq s$, donc $U(f) \cdot U(g) < N$.

Le nombre d'entiers de $\{1, \dots, N-1\}$ dont le carré est p_m -friable est donc minoré par le nombre de couples de suites finies $f = (f_1, \dots, f_m)$ et $g = (g_1, \dots, g_m)$ avec $f_1 + \cdots + f_m \leq s/2$ et $g_1 + \cdots + g_m \leq s/2$ telles que $\lambda(f) = \lambda(g)$ multiplié par 2^d et divisé par le nombre de représentations possibles pour la décomposition.

Puisque n_a désigne n_a le nombre de suite finie d'entiers $h = (h_1, \dots, h_m)$ telles que $h_1 + \cdots + h_m \leq s/2$ et $\lambda(h) = a$ pour tout $a \in \{-1, 1\}^d$, le nombre de couples de telles suites avec $\lambda(f) = \lambda(g) = a$ est égal à n_a^2 . Nous obtenons que le nombre d'entiers $x \in [0, N]$ tel que $x^2 \pmod{N}$ est m -friable est minoré par

$$\frac{2^d}{\binom{s}{s/2}} \sum_{a \in \{-1, +1\}^d} n_a^2$$

4. En appliquant l'inégalité de Cauchy-Schwarz, nous obtenons

$$2^d \sum_{a \in \{-1, +1\}^d} n_a^2 \geq \left(\sum_{a \in \{-1, +1\}^d} n_a \right)^2$$

Nous avons

$$\begin{aligned} \sum_{a \in \{-1, +1\}^d} n_a &= \sum_{a \in \{-1, +1\}^d} \#\{h \in \mathbb{N}^m \mid h_1 + \cdots + h_m \leq s/2 \wedge \lambda(h) = a\} \\ &= \#\{h = (h_1, \dots, h_m) \in \mathbb{N}^m \mid h_1 + \cdots + h_m \leq s/2\} \\ &= \binom{m + s/2}{s/2} \end{aligned}$$

où la dernière égalité a été démontrée à la première question de l'exercice 5.6. Nous obtenons que le nombre d'entiers de $[1, N-1]$ dont le carré est p_m -friable modulo N est minoré par

$$\binom{s}{s/2}^{-1} \binom{m + s/2}{s/2}^2 = \frac{((s/2)!)^2}{s!} \binom{m + s/2}{s/2}^2 \geq \frac{((s/2)!)^2}{s!} \left(\frac{m^{s/2}}{(s/2)!} \right)^2 = \frac{m^s}{s!}$$

La probabilité qu'un entier aléatoire de $[1, N - 1]$ ait un carré p_m -friable modulo N est donc minoré par $m^s/Ns!$.

Avec une analyse proche de celle vue à l'exercice 5.7, nous déduisons de l'estimation précédente que la méthode de Dixon donne un algorithme sous-exponentiel de factorisation de complexité $L_N(1/2, \sqrt{9/2})$. Les deux méthodes pour résoudre le problème du logarithme discret dans un anneau $(\mathbb{Z}/N\mathbb{Z})^*$ et factoriser l'entier N sont donc très proches. L'exercice suivant montre que dans certains cas, le problème de la factorisation d'un entier N se réduit à un problème de logarithme discret dans $(\mathbb{Z}/N\mathbb{Z})^*$.

Algorithme 6.10 Méthode de Dixon

ENTRÉE: $N \in \mathbb{N}$ et une base de facteurs $\mathcal{B} = \{2, 3, 5, \dots, p_k\}$

SORTIE: $d \in \{2, \dots, N - 1\}$ tel que $d \mid N$ ou ÉCHEC

$\mathcal{L} \leftarrow \emptyset$

$i \leftarrow 0$

$\mathcal{L} \leftarrow \emptyset$

répéter

$i \leftarrow i + 1$

répéter

$x_i \xleftarrow{\text{u.a.}} (\mathbb{Z}/N\mathbb{Z});$

$y_i \leftarrow x_i^2 \pmod{N}$

jusqu'à y_i est p_m -friable

$\triangleright y_i = p_1^{e_{i,1}} \dots p_m^{e_{i,m}}$ par divisions successives

$\vec{e}_i \leftarrow (e_{i,1}, \dots, e_{i,m})$

$\mathcal{L} \leftarrow \mathcal{L} \cup \{(x_i, \vec{e}_i)\}$

jusqu'à $i = m + 1$

Trouver $\vec{v} \in \mathbb{Z}^{k+1}$ tel que $v_1 \vec{e}_1 + v_2 \vec{e}_2 + \dots + v_{k+1} \vec{e}_{k+1} \in (2 \cdot \mathbb{Z})^k$.

\triangleright par élimination gaussienne

$x \leftarrow \prod_{i=1}^{m+1} x_i^{v_i} \pmod{N}; y \leftarrow \prod_{i=1}^{m+1} \prod_{j=1}^m p_j^{e_{i,j} v_i} \pmod{q}$

si $\text{pgcd}(x - y, N) \notin \{1, N\}$ **alors**

retourner $\text{pgcd}(x - y, N)$

sinon

si $\text{pgcd}(x + y, N) \notin \{1, N\}$ **alors**

retourner $\text{pgcd}(x + y, N)$

fin si

sinon

retourner ÉCHEC

fin si

Exercice 6.22 Factorisation et logarithme discret

Soient p et q deux nombres premiers distincts et soit $N = pq$. Soit α un élément inversible de $(\mathbb{Z}/N\mathbb{Z})^*$. Notons α_p et α_q les images de α dans $(\mathbb{Z}/p\mathbb{Z})^*$ et $(\mathbb{Z}/q\mathbb{Z})^*$ (respectivement).

- Montrer que l'ordre de α dans $(\mathbb{Z}/N\mathbb{Z})^*$ est égal au plus petit multiple commun de l'ordre de α_p dans $(\mathbb{Z}/p\mathbb{Z})^*$ et de l'ordre de α_q dans $(\mathbb{Z}/q\mathbb{Z})^*$.
- Notons $d = \text{pgcd}(p - 1, q - 1)$. Montrer qu'il existe un élément d'ordre $\varphi(N)/d$ dans $(\mathbb{Z}/N\mathbb{Z})^*$ où $\varphi(N) = \#(\mathbb{Z}/N\mathbb{Z})^*$ est la fonction indicatrice d'Euler de N .

Dans la suite nous supposons que $p > 3$ et $q > 3$ et que $\text{pgcd}(p - 1, q - 1) = 2$.

- Soit γ un élément d'ordre $\varphi(N)/d$ de $(\mathbb{Z}/N\mathbb{Z})^*$ et soit a le logarithme discret de $\gamma^N \pmod{N}$ en base γ . Montrer que $N - a = \varphi(N)$.
- Écrire un algorithme polynomial qui prend en entrée N et a et retourne les facteurs premiers p et q de N .

Solution

- Notons λ_p , λ_q et λ_N les ordres de α modulo p , q et N (respectivement). Puisque

$$\alpha^{\lambda_N} \equiv 1 \pmod{N}$$

nous avons $\alpha^{\lambda_N} \equiv 1 \pmod{p}$ et $\alpha^{\lambda_N} \equiv 1 \pmod{q}$ et λ_p et λ_q divisent donc λ_N . Nous obtenons que $\lambda = \text{ppcm}(\lambda_p, \lambda_q)$ divise λ_N . Réciproquement, puisque

$$\alpha^\lambda \equiv 1 \pmod{p} \quad \text{et} \quad \alpha^\lambda \equiv 1 \pmod{q}$$

nous avons $\alpha^\lambda \equiv 1 \pmod{N}$ par le théorème chinois des restes et λ_N divise λ , d'où le résultat.

- Soient γ_p et γ_q des générateurs de $(\mathbb{Z}/p\mathbb{Z})^*$ et $(\mathbb{Z}/q\mathbb{Z})^*$. Puisque p et q sont distincts, le théorème chinois des restes nous assure l'existence d'un élément $\gamma \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $\gamma \equiv \gamma_p \pmod{p}$ et $\gamma \equiv \gamma_q \pmod{q}$. D'après la question précédente, l'ordre de cet élément est égal à $\text{ppcm}(p - 1, q - 1) = (p - 1)(q - 1)/\text{pgcd}(p - 1, q - 1) = \varphi(N)/d$.
- Le logarithme discret de γ^N en base γ est le plus petit entier positif a tel que $\gamma^a \equiv \gamma^N \pmod{N}$. Par hypothèse, γ est d'ordre $\varphi(N)/d = \varphi(N)/2$. Nous avons $\gamma^{N-a} \equiv 1 \pmod{N}$, donc a est le plus petit entier positif tel que $\varphi(N)/2$ divise $N - a$. Par hypothèse sur p et q , $3\varphi(N) > 2N$ et donc nécessairement, nous avons $N - a = 2\varphi(N)/2 = \varphi(N)$.
- Nous avons $N = pq$ et $\varphi(N) = N - a$ avec

$$\varphi(N) = (p - 1)(q - 1) = pq - p - q + 1 = (N + 1) - (p + q)$$

Nous connaissons donc la somme et le produit de p et q et nous pouvons les retrouver en résolvant l'équation quadratique

$$X^2 - (\varphi(N) - (N + 1))X + N = X^2 - (p + q)X + pq = (X - p)(X - q)$$

Chapitre 6 • Factorisation des entiers et primalité

Les diviseurs premiers p et q sont donc les entiers

$$\frac{(\varphi(N) - (N + 1)) \pm \sqrt{(\varphi(N) - (N + 1))^2 - 4N}}{2}$$

et la racine carrée peut être obtenue en temps polynomial par une méthode d'analyse numérique (comme la méthode de Newton par exemple).

CHIFFREMENT À CLÉ PUBLIQUE

7

Le *chiffrement asymétrique* ou *chiffrement à clé publique* est une méthode de chiffrement qui repose sur l'utilisation d'une clé publique (qui peut être diffusée) et d'une clé privée (gardée secrète). La première permet de chiffrer un message et la seconde de le déchiffrer. Un expéditeur peut ainsi utiliser la clé publique d'un destinataire pour chiffrer un message qu'il sera le seul capable de déchiffrer, garantissant ainsi la confidentialité du message transmis.

Dans l'article fondateur de la cryptographie asymétrique, W. DIFFIE et M. HELLMAN ont suggéré que l'utilisation d'une *fonction à sens unique à trappe* devait permettre la création de schémas de chiffrement à clé publique. Il s'agit d'une fonction facile à calculer mais pour laquelle il est impossible de calculer l'inverse en temps polynomial à moins de connaître une information supplémentaire appelée *trappe*. R. L. RIVEST, A. SHAMIR et L. ADLEMAN ont proposé en 1978 le premier exemple de fonction à sens unique à trappe (supposée) qui porte désormais le nom de *fonction RSA*.

Nous examinerons tout d'abord les propriétés de la fonction RSA. Nous analyserons ensuite plusieurs variantes du système de *chiffrement RSA* qui reposent sur cette fonction à sens unique à trappe (supposée). Nous étudierons ensuite le protocole de *mise en accord de clé* de Diffie-Hellman. Nous examinerons enfin les propriétés de sécurité du système de *chiffrement d'ElGamal* qui repose en partie sur la difficulté du problème du logarithme discret dans un groupe.

7.1 FONCTION RSA

La fonction à sens unique (conjecturale) la plus souvent citée est la multiplication des entiers et la fonction RSA repose en partie sur le problème de la factorisation des entiers. Elle consiste à choisir deux grands nombres premiers notés p et q et à calculer leur produit $N = p \cdot q$. Un entier qui est le produit de deux nombres premiers impairs est appelé un *module RSA*.

La fonction RSA utilise ensuite un entier e premier avec $\varphi(N)$, la fonction indicatrice d'Euler de N (*i.e.* $\varphi(N) = |(\mathbb{Z}/N\mathbb{Z})^*| = (p-1)(q-1)$). La fonction RSA associée à N et e est définie par

$$f_{N,e} : \begin{cases} (\mathbb{Z}/N\mathbb{Z})^* \longrightarrow (\mathbb{Z}/N\mathbb{Z})^* \\ x \longmapsto x^e \bmod N \end{cases}$$

En connaissant la valeur de $\varphi(N)$, il est possible de calculer l'inverse d de e modulo $\varphi(N)$ et la connaissance de d permet d'inverser facilement la fonction RSA. En effet, pour tout $x \in (\mathbb{Z}/N\mathbb{Z})^*$, nous avons

$$(f_{N,e}(x))^d = (x^e)^d \bmod N = x^{ed} \bmod N = x$$

d'après le petit théorème de Fermat.

L'entier d est donc la trappe utilisée pour inverser $f_{N,e}$ et si l'on dispose d'un moyen de factoriser l'entier N , alors il est possible de calculer $\varphi(N)$ et donc la trappe d . Un problème ouvert fondamental de la cryptographie à clé publique est de savoir s'il est possible d'inverser la fonction RSA sans connaître la factorisation de N .

L'exercice suivant montre que la connaissance de la trappe d est équivalente à la connaissance de la factorisation de N .

Exercice 7.1 Fonction RSA et factorisation

Soient N un module RSA, $2 < e < N$ un entier premier avec $\varphi(N)$ et $2 < d < \varphi(N)$ l'inverse de e modulo $\varphi(N)$. Montrer qu'il existe un algorithme polynomial probabiliste qui, étant donnés N , e et d , retourne la factorisation de N .

Solution

Avec la connaissance d'un couple d'entiers (e, d) vérifiant $ed \equiv 1 \pmod{\varphi(N)}$, nous avons, pour tout $x \in (\mathbb{Z}/N\mathbb{Z})^*$, $x^{ed-1} \equiv 1 \pmod{N}$.

Puisque $ed - 1$ est un nombre pair, $y = x^{(ed-1)/2}$ est une racine carrée de 1 modulo N pour tout $x \in (\mathbb{Z}/N\mathbb{Z})^*$. Si cette valeur y est différente de 1 et -1 alors en calculant $\text{pgcd}(y-1, N)$ ou $\text{pgcd}(y+1, N)$, nous obtenons un diviseur non trivial de N (*cf.* Exercice (6.20)). Dans le cas d'un module RSA qui est le produit de deux nombres premiers, un diviseur propre donne la factorisation complète. Si la valeur de y est 1 ou -1 et si $(ed - 1)/2$ est encore pair, il est possible de répéter la méthode avec $y' = x^{(ed-1)/4}$.

Plus généralement, notons k la plus grande puissance de 2 divisant $ed - 1$. Pour un élément $x \in (\mathbb{Z}/N\mathbb{Z})^*$, l'algorithme calcule, $y = x^{(ed-1)/2^k} \pmod{N}$. Si cette valeur est égale à ± 1 , l'algorithme recommence avec un autre choix de x mais dans le cas contraire, il existe une puissance $y^{2^i} \not\equiv \pm 1 \pmod{N}$ telle que $y^{2^{i+1}} \equiv \pm 1 \pmod{N}$ ce qui révèle la factorisation de N . Nous obtenons ainsi l'algorithme (7.1)

Algorithme 7.1 Factorisation de N avec la connaissance de e et d

ENTRÉE: N un module RSA, $e, d \leq N$ deux entiers avec $e \cdot d \equiv 1 \pmod{\varphi(N)}$

SORTIE: (p, q) tel que $N = pq$ avec $p, q \geq 2$.

```

 $k \leftarrow 1$ 
tant que  $2$  divise  $(ed - 1)/2^k$  faire
     $k \leftarrow k + 1$ 
fin tant que
tant que VRAI faire
     $x \xleftarrow{u.a.} (\mathbb{Z}/N\mathbb{Z})^*$ 
     $\ell \leftarrow 0$ 
     $y \leftarrow x^{(ed-1)/2^k}$ 
    tant que  $y \neq \pm 1$  et  $\ell < k$  faire
        si  $y^2 \equiv 1 \pmod{N}$  alors
             $p \leftarrow \gcd(y - 1, N)$ 
             $q \leftarrow N/p$ 
            retourner  $(p, q)$ 
        sinon
             $y \leftarrow y^2$ 
             $\ell \leftarrow \ell + 1$ 
        fin si
    fin tant que
fin tant que

```

$\triangleright 2^k \mid (ed - 1)$ et $2^{k+1} \nmid (ed - 1)$

La probabilité qu'un élément x tiré uniformément aléatoirement dans $(\mathbb{Z}/N\mathbb{Z})^*$, vérifie $x^{(ed-1)/2} \equiv \pm 1 \pmod{N}$ est supérieure à $1/2$ (*cf.* Problème (6.6) sur le test de primalité de Miller-Rabin) et le nombre espéré de répétitions de l'algorithme est donc inférieur à 2.

Note

Ce résultat (montré par M. O. RABIN dans [57]) ne montre cependant pas qu'inverser la fonction RSA est équivalent à la factorisation du module utilisé. Il est en effet peut-être possible d'inverser cette fonction sans connaître la valeur de la clé secrète d . Le problème de l'équivalence de la factorisation des modules RSA et de l'inversion de la fonction RSA est toujours ouvert. En 2004, A. MAY [45, 15] a présenté un algorithme polynomial déterministe montrant que la connaissance de la trappe d est équivalente à la connaissance de la factorisation de N .

Étant donné un module RSA N , un entier e premier avec $\varphi(N)$, et un élément $y \in (\mathbb{Z}/N\mathbb{Z})^*$, le problème algorithmique qui consiste à retrouver $x \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $x^e = y \pmod{N}$ est appelé *problème RSA*. Les deux exercices suivants montrent que ce problème algorithmique possède de bonnes propriétés pour la cryptographie.

Exercice 7.2 Auto-réducibilité du problème RSA

Soient N un module RSA et e un nombre entier premier avec $\varphi(N)$. Considérons un algorithme \mathcal{A} qui prend en entrée un élément de $(\mathbb{Z}/N\mathbb{Z})^*$ et retourne un élément de

$(\mathbb{Z}/N\mathbb{Z})^*$, en temps τ (dans le pire des cas) où τ représente au moins le coût d'une exponentiation dans $(\mathbb{Z}/N\mathbb{Z})^*$.

Supposons qu'il existe un sous-ensemble E de $(\mathbb{Z}/N\mathbb{Z})^*$ avec $\#E \geq \epsilon N$ et $\epsilon \in]0, 1]$ pour lequel lorsque \mathcal{A} est exécuté sur un élément $x \in E$, l'élément y retourné par \mathcal{A} vérifie $y^e = x \pmod{N}$.

Montrer qu'il existe un algorithme \mathcal{B} qui résout le problème RSA dans $(\mathbb{Z}/N\mathbb{Z})^*$ en un temps espéré $O(\tau/\epsilon)$.

Solution

Il suffit de reprendre l'idée de l'exercice (5.8) en exécutant l'algorithme \mathcal{A} sur plusieurs instances (uniformément distribuées) du problème RSA jusqu'à trouver une instance qui appartient à l'ensemble E . L'algorithme \mathcal{B} doit être capable de résoudre le problème RSA initial à partir des instances créées. Comme dans l'exercice (5.8), il suffit d'utiliser les propriétés algébriques de la fonction RSA et notamment ses propriétés de morphisme. Considérons donc l'algorithme \mathcal{B} défini à partir \mathcal{A} dans l'algorithme (5.7).

Algorithme 7.2 Algorithme \mathcal{B}

ENTRÉE: $y \in (\mathbb{Z}/N\mathbb{Z})^*$

SORTIE: $x \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $x^e = y \pmod{N}$.

```

tant que VRAI faire
    u.a.
     $r \leftarrow (\mathbb{Z}/N\mathbb{Z})^*$ 
     $z \leftarrow y \cdot r^e$ 
     $w \leftarrow \mathcal{A}(z)$ 
    si  $(w/r)^e = y \pmod{N}$  alors
        retourner  $(w/r) \pmod{N}$ 
    fin si
fin tant que
```

Le temps d'exécution de l'algorithme \mathcal{B} est essentiellement égal au temps d'exécution de l'algorithme \mathcal{A} multiplié par le nombre d'itérations de la boucle **tant que ... faire**. Cette boucle se termine dès que \mathcal{A} retourne la bonne racine e -ième de $z \in (\mathbb{Z}/N\mathbb{Z})^*$ ce qui par hypothèse se produit dès que $z \in E$. L'élément r est tiré uniformément aléatoirement dans $(\mathbb{Z}/N\mathbb{Z})^*$, donc la probabilité que $y \cdot r^e \in E$ est indépendante de y et égale à ϵ . Le nombre espéré d'itérations de la boucle **tant que ... faire** est donc égal à $1/\epsilon$ d'où le résultat.

L'exercice suivant montre que déterminer le bit le plus significatif (ou le moins significatif) d'un élément $x \in (\mathbb{Z}/N\mathbb{Z})^*$ à partir de $y \equiv x^e \pmod{N}$ est aussi difficile que de retrouver la valeur de x en intégralité (*i.e.* que résoudre le problème RSA).

Problème 7.3 Sécurité des bits de la fonction RSA

Soient N un module RSA, e un nombre entier premier avec $\varphi(N)$ et $f_{N,e}$ la fonction RSA associée.

- Montrer qu'il existe un algorithme qui étant donné $f_{N,e}(x)$ pour $x \in (\mathbb{Z}/N\mathbb{Z})^*$ retourne le symbole de Jacobi $\left(\frac{x}{N}\right)$ en temps polynomial.

Considérons la fonction $\delta : (\mathbb{Z}/N\mathbb{Z}) \rightarrow \{0, 1\}$ définie par

$$\delta(x^e) = \begin{cases} 0 & \text{si } 0 \leq x \bmod N \leq N/2, \\ 1 & \text{si } N/2 \leq x \bmod N \leq N - 1. \end{cases}$$

- Montrer l'équivalence

$$\delta((2x)^e \bmod N) = 0 \iff x \in \left[0, \frac{N}{4}\right] \cup \left[\frac{N}{2}, \frac{3N}{4}\right]$$

et généraliser en caractérisant les $x \in (\mathbb{Z}/N\mathbb{Z})^*$ tels que $\delta((2^t x)^e \bmod N) = 0$.

- Montrer comment transformer tout algorithme polynomial calculant $\delta(z)$ pour tout $z \in (\mathbb{Z}/N\mathbb{Z})^*$ en un algorithme polynomial qui inverse la fonction RSA.
- Considérons la fonction $\gamma : (\mathbb{Z}/N\mathbb{Z}) \rightarrow \{0, 1\}$ définie par

$$\gamma(x^e \bmod N) = \begin{cases} 0 & \text{si } x \text{ est pair,} \\ 1 & \text{si } x \text{ est impair.} \end{cases}$$

Montrer comment transformer tout algorithme polynomial calculant $\gamma(z)$ pour tout $z \in (\mathbb{Z}/N\mathbb{Z})^*$ en un algorithme polynomial qui inverse la fonction RSA.

Solution

- Nous avons

$$\left(\frac{x}{N}\right) = \left(\frac{x}{N}\right)^e = \left(\frac{x^e \bmod N}{N}\right) = \left(\frac{f_{N,e}(x)}{N}\right)$$

et ce dernier symbole de Jacobi peut être calculé en temps polynomial (en utilisant la loi de réciprocité quadratique).

- Supposons que $x \in \left[0, \frac{N}{4}\right]$. Nous avons $2x \in \left[0, \frac{N}{2}\right]$ et $\delta((2x)^e) = 0$.

Si $x \in \left[\frac{N}{2}, \frac{3N}{4}\right]$. Nous avons $2x \in \left[N, \frac{3N}{2}\right]$, d'où $2x \bmod N = 2x - N \in \left[0, \frac{N}{2}\right]$ et $\delta((2x)^e) = 0$.

De même si $x \in \left[\frac{N}{4}, \frac{N}{2}\right] \cup \left[\frac{3N}{4}, N\right]$, nous avons $\delta((2x)^e) = 1$, d'où l'équivalence.

En généralisant, nous obtenons immédiatement

$$\delta((2^t x)^e) = 0 \iff x \in \bigcup_{i=0}^{2^t-1} \left[\frac{2i}{2^{t+1}}N, \frac{2i+1}{2^{t+1}}N \right]$$

En effet, $\delta((2^t x)^e) = 0$ si et seulement si $2^t x \bmod N$ est dans l'intervalle $[0, N/2[$, c'est-à-dire si le $t + 1$ -ième bit de x/N en base 2 est égal à 0. Il s'agit bien de l'union des intervalles décrite ci-dessus.

3. Tout entier $x < N$ s'écrit en base deux avec un nombre de bits au plus égal à $n = \lceil \log N \rceil$. De plus, connaissant $y = f_{N,e}(x)$, il est facile de calculer $f_{N,e}(2x)$ par une multiplication modulaire

$$(2x)^e = 2^e \cdot x^e \equiv f_{N,e}(x) \cdot 2^e \bmod N$$

qui se fait en temps polynomial en $\log N$. D'après la question précédente, en appliquant successivement l'algorithme polynomial calculant $\delta(z)$ pour tout $z \in (\mathbb{Z}/N\mathbb{Z})^*$ à $f_{N,e}(2^i x)$ pour $i \in \{0, \dots, \lceil \log N \rceil\}$ nous obtenons le $(i + 1)$ -ième bit de x/N de proche en proche en temps polynomial.

Algorithme 7.3 Inversion de la fonction RSA à partir de \mathcal{A} calculant δ

ENTRÉE: $y \in (\mathbb{Z}/N\mathbb{Z})^*$

SORTIE: $x \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $x^e = y \bmod N$.

```

 $d \leftarrow 0$ 
 $f \leftarrow N$  ▷  $x \in [d, f]$  dans tout l'algorithme
pour  $i$  de 0 à  $\lceil \log N \rceil$  faire
     $b \leftarrow \mathcal{A}(N, e, (a^e y) \bmod N)$ 
    si  $b = 0$  alors
         $d \leftarrow (d + f)/2$ 
    sinon
         $f \leftarrow (d + f)/2$ 
    fin si
fin pour
retourner  $\lfloor d \rfloor$ 

```

4. Il suffit de remarquer que $\gamma(2^e x^e \bmod N) = \delta(x^e)$ pour tout $x \in (\mathbb{Z}/N\mathbb{Z})^*$. En effet, si x est inférieur à $N/2$, $2x$ est inférieure ou égale à N et $2x \bmod N = 2x$ est pair, donc $\gamma((2x)^e \bmod N) = 0$. Si x est supérieur à $N/2$, $2x \bmod N = 2x - N$ et comme N est impair, $\gamma((2x)^e \bmod N) = 1$. En utilisant cette remarque pour calculer δ , il suffit d'appliquer l'algorithme de la question précédente pour inverser la fonction RSA.

Le réseau d'excellence européen *Ecrypt* en cryptographie recommande d'utiliser des modules RSA de 3248 bits pour obtenir une sécurité équivalente à l'AES avec des clés de 128 bits. Le module RSA le plus grand qui a été factorisé dans la littérature fait 768 bits. La factorisation réalisée a mis en œuvre une très importante puissance de calcul répartie entre différents pays et a nécessité des avancées technologiques et théoriques [36].

7.2 CHIFFREMENT RSA

La notion de fonction à sens unique à trappe donne naturellement naissance à un système de chiffrement à clé publique. L'espace des textes clairs est le domaine de la fonction et l'espace des textes chiffrés est son image. Pour chiffrer un message, on lui applique simplement la fonction à sens unique et l'utilisateur qui connaît la trappe associée peut inverser la fonction et retrouver le message clair initial. Initialement, le protocole de chiffrement a été décrit sous cette forme appelée *système de chiffrement RSA naïf*.

Protocole de chiffrement RSA naïf

Génération des clés : l'utilisateur tire aléatoirement deux nombres premiers p et q et calcule $N = pq$. Il calcule la fonction indicatrice d'Euler de N , $\varphi(N) = (p - 1)(q - 1)$. Il choisit un exposant public e premier à $\varphi(N)$ et calcule d l'inverse de e modulo $\varphi(N)$. La clé publique est le couple (N, e) et la clé secrète est l'entier d .

Chiffrement : étant donné un message $m \in (\mathbb{Z}/N\mathbb{Z})^*$, l'algorithme de chiffrement calcule le chiffré $c \equiv m^e \pmod{N}$.

Déchiffrement : étant donné un chiffré $c \in (\mathbb{Z}/N\mathbb{Z})^*$ et la clé secrète d , l'algorithme de déchiffrement retourne $c^d \pmod{N}$.

La notion naturelle de sécurité que doit garantir un système de chiffrement à clé publique est l'assurance que la vue d'un texte chiffré ne doit pas révéler d'information sur le texte clair. Les objectifs de l'adversaire, par ordre de difficulté décroissante, peuvent donc être :

- le *bris total* : l'adversaire retrouve la clé privée de déchiffrement ;
- l'*inversion du chiffrement* : l'adversaire est capable de retrouver l'intégralité d'un message clair associé à un message chiffré donné ;
- la *sécurité sémantique* : l'adversaire est capable d'obtenir un bit d'information sur un message clair associé à un message chiffré donné.

Cette notion est la variante calculatoire de la sécurité parfaite : étant donnés une clé publique, deux messages M_0 et M_1 choisis par l'adversaire et un chiffré C d'un message M_b pour $b \in \{0, 1\}$, un adversaire ne peut pas obtenir la valeur du bit b avec une probabilité significativement meilleure que $1/2$.

Pour un système de chiffrement à clé publique, nous supposons toujours que l'attaquant dispose de la clé publique et peut par conséquent chiffrer les messages de son choix. Les *attaques à clairs choisis* correspondent donc dans ce contexte au moyen d'attaque le plus faible. Dans une *attaque à chiffrés choisis non adaptative*, l'attaquant peut obtenir le déchiffrement de tout message, mais seulement avant d'avoir reçu le chiffré c^* (sur lequel il souhaite obtenir de l'information). Dans une *attaque*

à chiffrés choisis (adaptative), l'attaquant peut obtenir tout au long de l'attaque le déchiffrement de tout message (excepté le chiffré c^*).

Exercice 7.4 Sécurité du protocole de chiffrement RSA naïf

1. Montrer que le protocole de chiffrement RSA naïf n'est pas sémantiquement sûr sous une attaque à clairs choisis.
2. Montrer que le protocole de chiffrement RSA naïf est inversible sous une attaque à un chiffré choisi.

Solution

1. Nous avons vu dans l'exercice précédent que la fonction RSA ne masque pas le symbole de Jacobi du message chiffré. Un adversaire est donc capable d'obtenir un bit d'information sur un message clair associé à un message chiffré donné et le protocole de chiffrement RSA naïf n'est pas sémantiquement sûr.

Une autre raison pour laquelle le protocole de chiffrement RSA naïf n'est pas sémantiquement sûr sous une attaque à clairs choisis est qu'il est déterministe. Étant donnés deux textes clairs m_0 et m_1 et le chiffré c de l'un d'entre eux, il est possible d'appliquer la fonction RSA à m_0 et à m_1 et de regarder quel chiffré est égal c .

2. Pour obtenir le déchiffrement d'un chiffré c^* , un attaquant peut simplement choisir un élément $r \in (\mathbb{Z}/N\mathbb{Z})^*$ et demander le déchiffrement m de $c^* \cdot r^e \pmod{N}$. Dans ce cas, nous avons $m^e = c^* \cdot r^e \pmod{N}$ et donc $(m/r)^e = c^* \pmod{N}$ et $m/r \pmod{N}$ est le texte clair associé à c^* .

Exercice 7.5 RSA avec module commun

Expliquer pourquoi il est dangereux d'attribuer dans le système de chiffrement RSA naïf des couples clé publique/clé secrète (e_1, d_1) , (e_2, d_2) associé à un même module RSA N à deux utilisateurs différents.

Solution

La première faille de sécurité vient naturellement du fait qu'un utilisateur qui connaît un couple clé publique/clé secrète pour un module RSA donné peut utiliser ce couple pour factoriser l'entier N (*cf.* Exercice 7.1). En particulier, il peut calculer les clés secrètes des autres utilisateurs et ainsi obtenir le déchiffrement de messages qui ne lui sont pas destinés.

Même si l'on suppose que les utilisateurs qui partagent le même module RSA se font mutuellement confiance, un attaquant en dehors du groupe peut obtenir le déchiffrement d'un message qui serait envoyé à deux utilisateurs différents. En effet, supposons qu'un

message m est chiffré pour un deux utilisateurs dont les exposants de chiffrement e_1 et e_2 sont premiers entre eux. Le théorème de Bézout assure l'existence de deux entiers u et v tels que $ue_1 + ve_2 = 1$ et l'algorithme d'Euclide étendu permet de calculer u et v en temps polynomial. Si un attaquant intercepte c_1 et c_2 deux chiffrés du même message m avec les exposants de chiffrement e_1 et e_2 (respectivement), il peut retrouver m en effectuant un simple produit de deux exponentiations :

$$c_1^u c_2^v \equiv (m^{e_1})^u (m^{e_2})^v = m^{e_1 u + e_2 v} = m \bmod N$$

Le système RSA naïf utilisé dans ce contexte n'assure donc pas la confidentialité des messages transmis.

Exercice 7.6 (avec programmation).

Diffusion de données chiffrées avec RSA

1. Considérons le système de chiffrement RSA où tous les utilisateurs choisissent l'exposant de chiffrement $e = 3$ et supposons qu'un même message m doive être envoyé à trois destinataires différents (utilisant des modules RSA différents). Montrer comment un attaquant interceptant ces trois chiffrés peut retrouver le message m sans connaître les clés secrètes des destinataires.
2. **Application.** Un même message m a été chiffré avec le système de chiffrement RSA naïf à trois utilisateurs dont les clés publiques sont $(N_1, 3)$, $(N_2, 3)$ et $(N_3, 3)$. Le chiffrement a produit les trois chiffrés c_1 , c_2 et c_3 (respectivement).

$$\begin{aligned} N_1 &= 2828397017089907131052840387106128713282514421195726109593859 \\ c_1 &= 161340658484276930595607630148167439628632052300968205657282 \\ N_2 &= 3093736383172883855913466918447482558463408826373170329533707 \\ c_2 &= 2920025432866783050696766042954529191133978814738805935291595 \\ N_3 &= 4495119919511106064205284407123143309601197579854381074387973 \\ c_3 &= 742851878532958654303493521961761568283962501737283926134034 \end{aligned}$$

Déchiffrer le message m .

Solution

1. Notons N_1 , N_2 et N_3 les modules RSA des trois destinataires et c_1 , c_2 et c_3 les chiffrés du message m pour les clés publiques $(N_1, 3)$, $(N_2, 3)$ et $(N_3, 3)$. Nous avons

$$\begin{aligned} c_1 &\equiv m^3 \bmod N_1 \\ c_2 &\equiv m^3 \bmod N_2 \\ c_3 &\equiv m^3 \bmod N_3 \end{aligned}$$

Nous pouvons supposer sans perte de généralité que les modules N_1, N_2 et N_3 sont premiers entre eux (sinon un calcul de plus grand diviseur commun revèle la factorisation de l'un d'entre eux et permet de retrouver le message m). En appliquant le théorème chinois des restes, nous pouvons donc construire un entier c tel que

$$c \equiv m^3 \pmod{N_1 N_2 N_3}$$

Puisque m est nécessairement inférieur à N_1, N_2 et N_3 , m^3 est inférieur au produit $N_1 N_2 N_3$ et la valeur c obtenue est donc égale à l'entier m^3 . En appliquant une méthode d'analyse numérique comme la méthode de Newton il est possible d'extraire la racine cubique (entièrre) de c et le raisonnement précédent montre qu'elle est égale au message chiffré m .

2. Avec les notations de l'exercice précédent, en appliquant le théorème chinois des restes, nous obtenons

$$\begin{aligned} c = m^3 &= 18816763723536577725467160405896417262574772298 \\ &\quad 49409426207693797722198701224860897069000 \end{aligned}$$

et il suffit d'extraire la racine cubique de c pour obtenir le texte clair

$$m = 123456789012345678901234567890$$

Note

Cette faille dans le protocole de chiffrement RSA naïf a été remarquée pour la première fois par J. HÅSTAD [31].

Le nombre de multiplications effectuées par l'algorithme d'exponentiation dichotomique 5.1 dépend de la taille de l'exposant utilisé. Il a donc été suggéré pour rendre l'inversion de la fonction RSA plus efficace d'utiliser une clé secrète d relativement petite. Pour se prémunir d'attaque par recherche exhaustive de la clé secrète, il est bien sûr nécessaire que le nombre de tels exposants soit suffisamment grand. En 1989, M. J. WIENER [69] a montré qu'il est possible de retrouver la valeur de d à partir du couple (N, e) dès que $d < (N^{1/4})/3$. La technique de Wiener repose sur l'utilisation de fractions continues.

Soit α un nombre réel. En posant $a_0 = \lfloor \alpha \rfloor$, nous obtenons $\alpha = a_0 + \alpha_0$ avec $\alpha_0 \in [0, 1[$. Si α_0 est non nul, nous pouvons écrire $\alpha_0^{-1} = a_1 + \alpha_1$ avec $a_1 = \lfloor \alpha_0^{-1} \rfloor$, de sorte que $\alpha = a_0 + 1/(a_1 + \alpha_1)$. En définissant par récurrence $a_k = \lfloor \alpha_{k-1}^{-1} \rfloor$ et $\alpha_k = \alpha_{k-1}^{-1} - a_k$ dès lors que $\alpha_{k-1} \neq 0$, nous obtenons l'expression

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\cdots + \cfrac{1}{a_k + \alpha_k}}}} = [a_0, a_1, a_2, a_3, \dots, a_k + \alpha_k]$$

La suite (a_k) est déterminée uniquement par α et il s'agit d'une suite finie si et seulement si $\alpha = a/b$ est un nombre rationnel (auquel cas, la suite (a_k) est simplement obtenue en appliquant l'algorithme d'Euclide à a et b). Soit $[a_0, a_1, \dots, a_k, \dots]$ le développement en fraction continue de α ; définissons la suite des nombres rationnels (p_k/q_k) définis par $(p_0, q_0) = (a_0, 1)$, $(p_1, q_1) = (a_0a_1 + 1, a_1)$ et

$$\begin{cases} p_k = a_k p_{k-1} + p_{k-2} \\ q_k = a_k q_{k-1} + q_{k-2} \end{cases}$$

Le nombre rationnel p_k/q_k est appelée la k -ième réduite du développement en fraction continue $[a_0, a_1, \dots, a_k, \dots]$. Pour tout entier k , nous avons $p_k/q_k = [a_0, a_1, \dots, a_k]$ et $|\alpha - p_k/q_k| \leq 1/(q_k q_{k+1})$.

L'attaque de Wiener sur le chiffrement RSA utilisant un petit exposant secret d repose sur le fait que les réduites du développement en fraction continue d'un nombre réel α non nul sont de meilleures approximations rationnelles de α et si un nombre rationnel est proche de α alors il s'agit d'une réduite de son développement en fraction continue :

Théorème 7.1

Soit $\alpha \in \mathbb{R}$.

1. La k -ième réduite p_k/q_k du développement en fraction continue de α est la meilleure approximation de α par une fraction de dénominateur inférieur à q_k :

$$\forall (p, q) \in (\mathbb{Z} \times \mathbb{N} \setminus \{0\}), \left| \alpha - \frac{p}{q} \right| < \left| \alpha - \frac{p_k}{q_k} \right| \Rightarrow q \geq q_k$$

2. **Théorème de Lagrange.** Si il existe deux entiers p et q tels que

$$\left| \alpha - \frac{p}{q} \right| \leq \frac{1}{2q^2}$$

alors p/q est une des réduites du développement en fraction continue de α .

Exercice 7.7 Attaque de Wiener

Soient $N = pq$ un module RSA avec $p < q < 2p$, $2 < e < \varphi(N)$ un nombre entier premier avec $\varphi(N)$ et $2 < d < \varphi(N)$ l'inverse de e modulo $\varphi(N)$.

1. Par définition, il existe un entier $k \in \mathbb{N}$ tel que $ed = 1 + k\varphi(N)$. Montrer que $k < d$ et que

$$\left| \frac{e}{N} - \frac{k}{d} \right| = \frac{3k}{d\sqrt{N}}$$

2. En déduire un algorithme polynomial qui retrouve la factorisation d'un module RSA N étant donné un couple (N, e) pour lequel il existe un entier $d \leq \frac{1}{\sqrt{6}}N^{1/4}$ tel que $ed \equiv 1 \pmod{\varphi(N)}$.

Solution

1. Nous avons $ed - k\varphi(N) = 1$ et nous obtenons

$$\begin{aligned} \left| \frac{e}{N} - \frac{k}{d} \right| &= \left| \frac{ed - kN}{Nd} \right| \\ &= \left| \frac{ed - k\varphi(N) - kN + k\varphi(N)}{Nd} \right| \\ &= \left| \frac{1 - k(N - \varphi(N))}{Nd} \right| \end{aligned}$$

Puisque $N - \varphi(N) = pq - (p - 1)(q - 1) = p + q - 1 < 3\sqrt{N}$, nous en déduisons :

$$\left| \frac{e}{N} - \frac{k}{d} \right| \leq \left| \frac{3k\sqrt{N}}{Nd} \right| = \frac{3k\sqrt{N}}{\sqrt{N}\sqrt{Nd}} = \frac{3k}{d\sqrt{N}}$$

Par ailleurs, nous avons $k\varphi(N) = ed - 1 < ed$, donc $k\varphi(N) < ed$. Puisque par hypothèse $e < \varphi(N)$, nous obtenons $k\varphi(N) < ed < \varphi(N)d$, et finalement $k\varphi(N) < \varphi(N)d$ soit $k < d$.

2. Puisque $k < d$ and $d \leq \frac{1}{\sqrt{6}}N^{\frac{1}{4}}$, nous avons

$$\frac{1}{2d^2} \geq \frac{6}{2\sqrt{N}} = \frac{3}{\sqrt{N}} > \frac{3k}{d\sqrt{N}}$$

et d'après la question précédente

$$\left| \frac{e}{N} - \frac{k}{d} \right| \leq \frac{1}{2d^2}$$

En appliquant le théorème de Lagrange, nous en déduisons que k/d est une réduite du développement en fraction continue de e/N . Ce développement peut-être calculé en temps polynomial en appliquant l'algorithme d'Euclide et pour chaque réduite, il est possible de tester si le dénominateur d est la bonne clé secrète en appliquant la technique de l'exercice 7.1.

Exercice 7.8 (avec programmation). Attaque de Wiener

Un utilisateur a créé une clé publique RSA (N, e) donnée ci-dessous où, pour accélérer les opérations de déchiffrement, il a choisi un exposant privé d relativement petit (où d est l'inverse de e modulo $\varphi(N)$). Implanter l'attaque de Wiener de l'exercice précédent pour trouver la valeur de d à partir de (N, e) et en déduire la factorisation de N .

$N = 2630048851947048265274043876774585976831617720728227254753421$

$e = 60177566799353897687038964037333604046539474788802464201235$

Solution

D'après l'exercice précédent, si d est suffisamment petit, le nombre rationnel k/d est une réduite du développement en fraction continue de e/N . Le développement en fraction continue de e/N est donné par la suite 0, 43, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 94347152401899997935393, 1, 7, 1, 3, 4, 10, 4, 1, 2, 1, 26, 2, 13, 3, 1, 21, 1, 1, 1, 27, 2, 1, 8, 3, 2, 10, 1, 1, 50, 19, 1, 1, 3, 3, 3, 1, 5, 1, 7, 2, 127, 7, 2, 3, 3, 4, 5, 2, 2, 1, 1, 7, 1, 52, 1, 3, 3. Autrement dit, nous avons

$$\frac{e}{N} = 0 + \cfrac{1}{43 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{\dots}}}}}}} = [0, 43, 1, 2, 2, 1, 1, 2, \dots]$$

En calculant les réduites de ce développement, nous obtenons successivement les nombres rationnels suivants :

$$0, \frac{1}{43}, \frac{1}{44}, \frac{3}{131}, \frac{7}{306}, \frac{10}{437}, \frac{17}{743}, \frac{44}{1923}, \frac{61}{2666}, \frac{105}{4589}, \frac{166}{7255}, \frac{437}{19099}, \dots$$

En calculant $c = 2^e \bmod N$, nous obtenons

$$c = 1230544696135210835528980976054431308478098415971165170244701 \bmod N,$$

et nous recherchons un dénominateur d parmi les dénominateurs de ces réduites pour lequel $c^d \equiv 2 \bmod N$. Une recherche informatique montre rapidement que $d = 19099$ convient et donc que vraisemblablement, $ed = 1 \bmod \varphi(N)$. Si c'est effectivement le cas, nous avons $x^{ed-1} \equiv 1 \bmod N$ pour tout $x \in (\mathbb{Z}/N\mathbb{Z})^*$ et comme $ed - 1$ est pair, nous pouvons obtenir une racine carrée non triviale de 1 (cf. Exercice 7.1).

L'entier $ed - 1$ est divisible par 2^7 , nous avons

$$2^{(ed-1)/2} \equiv 2^{(ed-1)/2^2} \equiv 2^{(ed-1)/2^3} \equiv 1 \bmod N$$

et $r = 2^{(ed-1)/2^4} \not\equiv 1 \bmod N$ avec

$$r = 2177196145483083074385872078047518412756468999348364188092007 \bmod N.$$

Nous obtenons ainsi une racine carrée non triviale de 1 et en calculant $\text{pgcd}(r-1, N)$ nous obtenons le nombre premier $p = 1271481759930465089654679368173$ puis $q = N/p = 2068491216178264811711559871777$.

Note

Les résultats d'approximation diophantienne et plus généralement de *géométrie des nombres* ont des applications spectaculaires en cryptographie à clé publique. D. BONEH et G. DURFEE [10] ont présenté en utilisant des techniques de géométrie des nombres un algorithme (heuristique) qui retrouve la clé secrète d si elle vérifie $d < N^{0.292}$. L'algorithme LLL est notamment un outil précieux pour attaquer des cryptosystèmes asymétriques mais qui dépasse le cadre de cet ouvrage. Nous renvoyons le lecteur curieux au livre très complet [49].

Exercice 7.9 (avec programmation). RSA et clairs liés

- Montrer que si l'on dispose des chiffrés RSA c et c' d'un clair aléatoire m et d'un clair lié $m + r$, où $0 < r < N$ est connu, pour une clé publique $(N, e = 3)$, alors on peut retrouver m en temps polynomial.
- Montrer comment généraliser cette approche pour tout entier e suffisamment petit.
- Application.** Utiliser la méthode de la question précédente pour retrouver le message m vérifiant $c = m^{17} \bmod N$ et $c' = (m + 1)^{17} \bmod N$ avec

$$N = 4750268523286534182543999246472514570042418299923101154793593$$

$$c = 1935621880512522306378371392939548737091684771868008026431626$$

$$c' = 1011424881854699101846188248967755233987658392601847378035075.$$

Solution

- Nous avons

$$c = m^3 \bmod N$$

$$c' = (m + r)^3 = m^3 + 3m^2r + 3mr^2 + r^3 \bmod N$$

donc

$$c' - c + 2r^3 = r(3m^2 + 3mr + 3r^2) \bmod N$$

$$c' + 2c - r^3 = m(3m^2 + 3mr + 3r^2) \bmod N$$

et finalement

$$m = \frac{r(c' + 2c - r^3)}{c' - c + 2r^3}$$

Donc il est possible de retrouver m en temps polynomial si $c' - c + 2r^3$ est inversible modulo N (ce qui est le cas avec une forte probabilité pour $0 < r < N$).

- La méthode précédente peut être généralisée en remarquant que m est racine des polynômes $g_1(X) = X^e - c$ et $g_2(X) = (X + r)^e - c'$ dans $(\mathbb{Z}/N\mathbb{Z})^*$. Le facteur linéaire $X - m$ divise donc les deux polynômes $g_1(X)$ et $g_2(X)$ et par conséquent leur plus grand diviseur commun $h(X)$. En appliquant l'algorithme d'Euclide dans $(\mathbb{Z}/N\mathbb{Z})[X]$, nous pouvons calculer $h(X)$ en temps polynomial ($(\mathbb{Z}/N\mathbb{Z})[X]$ n'est pas euclidien mais si une erreur se produit au cours du calcul du plus grand diviseur commun, c'est que

l'algorithme a rencontré un diviseur de zéro dans $(\mathbb{Z}/N\mathbb{Z})$ et il est alors possible de factoriser N). Le calcul de $h(X)$ nécessite un temps quadratique en e et $\log(N)$ et, de manière empirique, nous obtenons la plupart du temps que $h(X)$ est de degré 1 et donc égal à $X - m$.

3. En calculant le PGCD des polynômes $X^{17} - c$ et $(X+1)^{17} - c'$ modulo N , nous obtenons le polynôme $X - m$ avec

$$m = 371557122223853644755653627213080652093538084224189068804219$$

Les mécanismes de chiffrement symétrique étant moins coûteux en temps de calcul, ceux-ci sont généralement préférés aux mécanismes de chiffrement asymétrique. La cryptographie asymétrique est uniquement utilisée pour partager une clé symétrique, qui est ensuite utilisée pour tout le reste de l'échange. Les clés symétriques sont en pratique plus courtes que les modules RSA et l'exercice suivant montre qu'il convient d'être prudent lorsque l'on chiffre une « petite » clé avec la fonction RSA.

Exercice 7.10 RSA et petits textes clairs

Soient $N = pq$ un module RSA et $2 < e < \varphi(N)$ un nombre entier premier avec $\varphi(N)$.

1. Montrez que déchiffrer un message m chiffré par la fonction RSA est immédiat si $m^e < N$.
2. Supposons que $m = ab$ avec $a, b < T$. Proposer un algorithme par compromis temps-mémoire qui étant donné le chiffré c de m retrouve le message clair m .
3. Posons

$$\theta(T) = \#\{m \in \mathbb{N}, m = a \cdot b, a, b < T\}$$

En considérant les produits de la forme $r \cdot b$ avec $T/2 < r < T$ et $b < T$ où r est un nombre premier, Montrer que $\theta(T) \geq T^2/(2 \ln T)$ pour T suffisamment grand. On pourra utiliser le théorème des nombres premiers sous la forme explicite suivante :

$$\frac{x}{\ln x + 2} < \pi(x) < \frac{x}{\ln x - 4} \text{ pour } x \geq 55$$

4. Supposons qu'une clé AES aléatoire k de 128 bits est chiffrée avec le protocole de chiffrement RSA naïf. Montrer que k peut être retrouvée en 2^{65} évaluations de la fonction RSA avec probabilité supérieure à 12%.

Solution

1. Si $m^e < N$, le chiffré de m est $m^e \bmod N = m^e$. Il s'agit donc d'une puissance e -ième entière du message m . En appliquant une méthode d'analyse numérique, il est possible de retrouver m sans connaître la clé secrète associée à e .

2. Il suffit d'adapter l'algorithme « pas de bébé, pas de géant » de Shanks à ce contexte. En effet, si $m = ab$ avec $a, b < T$, nous pouvons écrire

$$m/a = b \quad \text{et donc} \quad c/a^e = (m/a)^e = b^e$$

Il suffit dans un premier temps de calculer les T valeurs prises par le terme de droite de cette égalité (*i.e.* les éléments b^e pour $b \in \{1, \dots, T-1\}$) et de stocker les valeurs obtenues (b^e, b) dans une table de hachage indexée par les éléments de $(\mathbb{Z}/N\mathbb{Z})^*$. Dans un second temps, l'algorithme va rechercher parmi les valeurs c/a^e pour $a \in \{1, \dots, T-1\}$ si l'une d'elles apparaît dans la table de hachage. Le nombre d'exponentiations à effectuer dans $(\mathbb{Z}/N\mathbb{Z})^*$ est de l'ordre de $2T$ alors qu'une recherche exhaustive aurait demander T^2 exponentiations dans $(\mathbb{Z}/N\mathbb{Z})^*$.

3. Suivant l'indication, considérons les produits d'un nombre premier $T > r > T/2$ par un nombre inférieur à $b < T$. Par le théorème des nombres premiers effectif, nous obtenons $T \cdot (T/2 \ln T)$ (pour T suffisamment grand) tels nombres parmi lesquels seuls les produits de la forme $r_1 \cdot r_2$ avec $T > r_1 > T/2$ et $T > r_2 > T/2$ sont comptés deux fois. Nous obtenons donc

$$\theta(T) \geq T(T/2 \ln T) - (T/2 \log T)^2/2 \simeq T^2/2 \ln T$$

4. Il suffit d'appliquer les deux questions précédentes avec $T = 2^{64}$. L'ensemble $\Theta(T) = \Theta(2^{64}) = \{m \in \mathbb{N}, m = a \cdot b, a, b < T\}$ est inclus dans l'intervalle d'entier $\{0, \dots, 2^{128}\}$ et il possède $\theta(T) = \theta(2^{64}) \geq 2^{128}/(128 \ln 2)$ éléments.

L'attaque par compromis temps-mémoire de la question 2 permettra de retrouver la clé AES k si elle appartient à $\Theta(2^{64})$ en 2^{65} évaluations de la fonction RSA. La probabilité de succès de l'attaque est donc supérieure à $(128 \ln 2)^{-1} \geq 0.1202$, d'où le résultat.

Note

Le problème d'estimer $\theta(T)$ a été posé par P. ERDÖS en 1955. En 2008, K. FORD a montré le résultat difficile affirmant que $\theta(T)$ est de l'ordre de $T^2/(\ln(T)^c(\ln \ln T)^{3/2})$ où $c = 1 - (1 + \ln \ln 2)/\ln 2 \simeq 0.086071 \dots$ [25].

Problème 7.11 Implantation du chiffrement RSA et théorème chinois des restes

- Expliquer comment accélérer le déchiffrement RSA pour un module $N = pq$ en commençant par faire le calcul modulo p et q . Montrer que l'algorithme obtenu est quatre fois plus rapide que l'algorithme naïf.
- Pour accélérer encore le calcul, une idée naturelle est d'utiliser une clé secrète d telle $d_p \equiv d \pmod{p-1}$ et $d_q \equiv d \pmod{q-1}$ sont relativement petits (et $d_p \neq d_q$). Proposer un algorithme de complexité $O(\min(\sqrt{d_p}, \sqrt{d_q}))$ de type

« pas de bébé, pas de géant » qui retourne la factorisation de N (en utilisant l'algorithme de multi-évaluation de polynômes vu au chapitre précédent).

3. Supposons que par accident (ou par intervention d'un adversaire), l'utilisateur effectuant un déchiffrement RSA fasse une seule erreur d'instruction lors de l'algorithme de déchiffrement de la première question. Montrer comment obtenir la factorisation de N .

Solution

1. Pour calculer $m = c^d \pmod{N}$, nous pouvons calculer séparément le message m modulo p et modulo q , en calculant $m_p = c^{d_p} \pmod{p}$ et $m_q = c^{d_q} \pmod{q}$, avec $d_p = d \pmod{p-1}$ et $d_q = d \pmod{q-1}$.

En utilisant le théorème chinois des restes, nous obtenons l'unique message $m \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $m \equiv m_p \pmod{p}$ et $m \equiv m_q \pmod{q}$. En précalculant $p(p^{-1} \pmod{q})$ et $q(q^{-1} \pmod{p})$, nous obtenons la valeur m en calculant seulement deux multiplications supplémentaires :

$$m = m_p q (q^{-1} \pmod{p}) + m_q p (p^{-1} \pmod{p})$$

Une exponentiation modulaire ayant une complexité cubique en la taille du module (sans utilisation de méthode de multiplication rapide), chacune des deux exponentiations est 8 fois plus rapide que l'exponentiation naïve $m = c^d \pmod{N}$. Il en résulte bien un gain en temps de calcul d'un facteur 4.

2. Pour un message aléatoire $m \in (\mathbb{Z}/N\mathbb{Z})^*$, nous avons $m^{ed_p-1} \equiv 1 \pmod{p}$ et avec forte probabilité $m^{ed_p-1} \not\equiv 1 \pmod{q}$. En particulier $\text{pgcd}(m^{ed_p-1}-1, N) = p$ avec une forte probabilité.

Supposons que $1 < d_p < K$ et posons $T = \lceil \sqrt{K} \rceil + 1$. En effectuant la division euclidienne de d_p par T , il existe des entiers d_0 et d_1 tels que $d_p = d_1 T + d_0$ avec $0 \leq d_0 < T$ et $0 \leq d_1 < T$. Nous avons donc

$$\text{pgcd}(m^{ed_p-1}-1, N) = \text{pgcd}(m^{ed_0-1}(m^{eT})^{d_1}-1, N) = p$$

Il est possible de calculer $(m^{eT})^{d_1}$ pour $d_1 \in \{0, \dots, T-1\}$ et m^{ed_0-1} pour $d_0 \in \{0, \dots, T-1\}$ en $O(T)$ multiplications dans $(\mathbb{Z}/N\mathbb{Z})^*$. La difficulté est de détecter cette égalité sans calculer tous les pgcd. Suivant l'indication, nous allons utiliser l'algorithme de multi-évaluation de polynômes du chapitre précédent. Posons

$$G(X) = \prod_{j=0}^{T-1} (m^{ej-1} X - 1) \pmod{N}$$

Ce polynôme est de degré T et peut être construit en temps $O(M(T) \log T)$ opérations dans $(\mathbb{Z}/N\mathbb{Z})^*$. Posons $\alpha = m^{eT} \pmod{N}$. Nous voulons calculer $G(\alpha^{d_1})$ pour chaque valeur candidate $0 \leq d_1 < T$. Par l'algorithme de multi-évaluation, nous pouvons le faire en $O(M(T) \log T)$ opérations dans $(\mathbb{Z}/N\mathbb{Z})^*$. Une fois ces valeurs calculées, il suffit de calculer les pgcd pour obtenir la factorisation de N .

3. Supposons que seule la valeur de $m^d \bmod p$ a été correctement calculée, et non celle de $m^d \bmod q$ en utilisant l'algorithme de déchiffrement de la question 1.
La déchiffrement incorrect de \tilde{m} vérifiera donc $\tilde{m} \equiv m \bmod p$ et $\tilde{m} \not\equiv m \bmod q$ et le plus grand diviseur commun de révèle la factorisation de N .

7.3 MISE EN ACCORD DE CLÉ DE DIFFIE-HELLMAN

La *mise en accord de clé* de Diffie-Hellman est une méthode inventée par W. DIFFIE et M. HELLMAN par laquelle deux personnes dialoguant sur un canal de communication non sécurisé peuvent créer une clé sans qu'une tierce partie puisse retrouver cette clé (même en ayant intercepté tous leurs échanges sur le canal).

Après avoir choisi un groupe $\mathbb{G} = \langle g \rangle$ d'ordre q où le problème du logarithme discret est jugé difficile, les deux utilisateurs (appelés traditionnellement Alice et Bob) effectuent l'interaction suivante (illustrée sur la figure 7.1) :

- Alice tire uniformément aléatoirement un entier a dans $(\mathbb{Z}/q\mathbb{Z})$, calcule $K_a = g^a \in \mathbb{G}$ et envoie la valeur obtenue à Bob ;
- Bob tire uniformément aléatoirement un entier b dans $(\mathbb{Z}/q\mathbb{Z})$, calcule $K_b = g^b \in \mathbb{G}$ et envoie la valeur obtenue à Alice ;
- à réception de la valeur K_b , Alice calcule K_b^a ;
- à réception de la valeur K_a , Bob calcule K_a^b ;

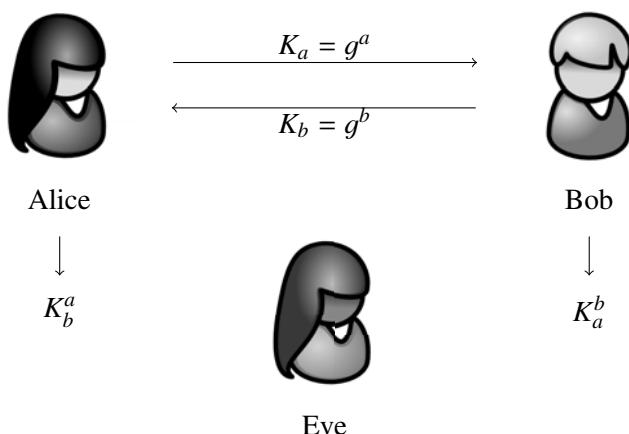


Figure 7.1 – Protocole de mise en accord de clé de Diffie-Hellman

À la fin de cette interaction, si les utilisateurs ont suivi le protocole, ils partagent la clé $K = g^{ab} = K_a^b = K_b^a$. Un attaquant passif qui aurait intercepté toutes les communications entre Alice et Bob doit reconstruire la valeur de K à partir des éléments g , $K_a = g^a$ et $K_b = g^b$. L'exercice suivant montre que le protocole de mise en accord de clé de Diffie-Hellman n'est cependant pas sûr contre des attaquants actifs.

Exercice 7.12 Attaque par le milieu

Décrire une attaque dans le protocole de mise en accord de clé Diffie-Hellman dans laquelle un attaquant *actif* (*i.e.* qui peut modifier les données pendant le protocole Diffie-Hellman) peut ensuite intercepter, déchiffrer et modifier toutes les communications qu'Alice ou Bob chiffreraient avec sa clé.

Solution

Dans le protocole décrit dans la figure 7.1, il suffit de supposer que l'attaquant actif réalise les opérations suivantes (illustrées sur la figure 7.2) :

- Lorsque Alice envoie la valeur K_a à Bob, Eve bloque la communication puis tire uniformément aléatoirement un entier \tilde{a} dans $(\mathbb{Z}/q\mathbb{Z})$, calcule $\widetilde{K}_a = g^{\tilde{a}} \in \mathbb{G}$ et envoie la valeur obtenue à Bob ;
- Lorsque Bob envoie la valeur K_b à Alice, Eve bloque la communication puis tire uniformément aléatoirement un entier \tilde{b} dans $(\mathbb{Z}/q\mathbb{Z})$, calcule $\widetilde{K}_b = g^{\tilde{b}} \in \mathbb{G}$ et envoie la valeur obtenue à Alice ;
- à réception de la valeur \widetilde{K}_b , Alice calcule \widetilde{K}_b^a ;
- à réception de la valeur \widetilde{K}_a , Bob calcule \widetilde{K}_a^b ;

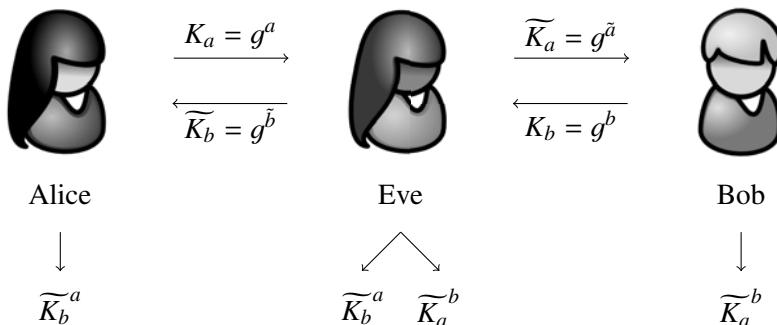


Figure 7.2 – Attaque par le milieu contre protocole de Diffie-Hellman

Contrairement au protocole réalisé entre utilisateurs honnêtes, l'égalité $\widetilde{K}_b^a = \widetilde{K}_a^b$ n'est plus vérifiée. Cependant si Alice envoie par la suite un message chiffré de façon symétrique avec la clé \widetilde{K}_b^a , Eve peut déchiffrer grâce à sa connaissance de \tilde{b} (puisque

$\widetilde{K}_b^a = K_a^{\tilde{b}}$), prendre connaissance du message et éventuellement le rechiffrer avec la clé $\widetilde{K}_a^b = K_b^{\tilde{a}}$ (que Bob croit partager avec Alice) pour que la communication ait l'apparence d'une communication normale.

Le problème algorithmique sur lequel repose la sécurité passive du protocole de mise en accord de clé de Diffie-Hellman (*i.e.* dans un groupe \mathbb{G} , étant donnés g, g^a, g^b calculer g^{ab}) est appelé *problème calculatoire de Diffie-Hellman*. Ce problème est *a priori* plus facile à résoudre que le problème du logarithme discret dans \mathbb{G} . À ce jour, il n'existe cependant pas d'algorithme pour résoudre ce problème plus rapidement qu'en calculant un logarithme discret. Dans certains cas, il est en fait possible de montrer que la difficulté des deux problèmes est équivalente.

Problème 7.13 Logarithme discret et Diffie-Hellman *

Soit M un entier. Soit $\mathbb{G} = \langle g \rangle$ un groupe fini cyclique d'ordre premier p tel que $p - 1$ soit M -friable et soit $h = g^x \in \mathbb{G}$.

1. Donner un algorithme probabiliste qui retourne $g \in (\mathbb{Z}/p\mathbb{Z})^*$ un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$.
2. Montrer que si $x \neq 0 \pmod p$, il existe $y \in (\mathbb{Z}/(p-1)\mathbb{Z})$ tel que $x = g^y \pmod p$. En déduire que pour retrouver x , il suffit de calculer y modulo tous les diviseurs de $p - 1$ qui sont puissances d'un nombre premier.
3. Soit q un diviseur premier de $p - 1$. Donner un algorithme qui prenant en entrée h et g et disposant d'un oracle qui résout le problème calculatoire de Diffie-Hellman dans \mathbb{G} retourne $y \pmod q$ en faisant au plus $O(\log p)$ requêtes à l'oracle et $O(\sqrt{q})$ exponentiations dans $(\mathbb{Z}/(p-1)\mathbb{Z})$ et \mathbb{G} .
4. Adapter l'algorithme précédent pour qu'il retourne $y \pmod {q^e}$ lorsque q^e est une puissance d'un nombre premier divisant $p - 1$.
5. Conclure.

Solution

1. Notons $p - 1 = q_1^{e_1} \dots q_\ell^{e_\ell}$ la décomposition en facteurs premiers de $p - 1$ (où les q_i pour $i \in \{1, \dots, \ell\}$ sont deux à deux distincts et $e_i \geq 1$ pour $i \in \{1, \dots, \ell\}$). Puisque $p - 1$ est M -friable, nous avons $q_i \leq M$ pour tout $i \in \{1, \dots, \ell\}$ et nous pouvons les supposer connus (l'algorithme ρ de Pollard ou l'algorithme de Pollard-Strassen permet de calculer les q_i pour $i \in \{1, \dots, \ell\}$ en $O(M^{1/4+o(1)})$ opérations élémentaires).

Pour construire un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$ de façon probabiliste, un algorithme peut tirer g dans $(\mathbb{Z}/p\mathbb{Z})^*$ pour tout $i \in \{1, \dots, \ell\}$ et vérifier que $g^{(p-1)/q_i}$ est différent de 1

dans $(\mathbb{Z}/p\mathbb{Z})^*$ pour tous les $i \in \{1, \dots, \ell\}$. Pour construire un tel élément g plus efficacement, nous pouvons tirer uniformément aléatoirement, pour chaque $i \in \{1, \dots, \ell\}$, g_i dans $(\mathbb{Z}/p\mathbb{Z})^*$ et vérifier que $g_i^{(p-1)/q_i}$ est différent de 1. La valeur

$$g = \prod_{i=1}^{\ell} g_i^{(p-1)/q_i^{e_i}}$$

convient alors. En effet, pour tout $i \in \{1, \dots, \ell\}$, $g_i^{(p-1)/q_i^{e_i}}$ est d'ordre $q_i^{e_i}$. Le produit g a pour ordre le pgcd des ordres des $g_i^{(p-1)/q_i^{e_i}}$ pour $i \in \{1, \dots, \ell\}$, soit $(p-1) = q_1^{e_1} \dots q_{\ell}^{e_{\ell}}$. Pour tout $i \in \{1, \dots, \ell\}$, un choix aléatoire de g_i dans $(\mathbb{Z}/p\mathbb{Z})^*$ vérifie $g_i^{(p-1)/q_i} \neq 1$ avec probabilité $1/q_i$. Pour tout $i \in \{1, \dots, \ell\}$, il faut donc essayer en moyenne $1/(1 - 1/q_i) \leq 2$ éléments avant d'en trouver un qui convient. L'algorithme est donc polynomial en $\ell = O(\log p)$ une fois la factorisation de $p-1$ connue.

2. Si x est inversible dans $(\mathbb{Z}/p\mathbb{Z})$, puisque g est un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$, il existe un entier $y \in (\mathbb{Z}/(p-1)\mathbb{Z})$ tel que $x = g^y \pmod{p}$. Si l'on connaît la valeur de y modulo chaque diviseur de $p-1$, il est possible de reconstruire la valeur de y modulo $p-1$ par le théorème chinois des restes (comme dans l'algorithme de Pohlig-Hellman (5.5)), puis la valeur de x .
3. Pour retrouver la valeur de y modulo q un diviseur de $p-1$, il suffit de calculer un logarithme discret dans le sous-groupe de $(\mathbb{Z}/p\mathbb{Z})^*$ d'ordre q engendré par $g^{(p-1)/q}$. La valeur de y mod q est simplement le logarithme discret de $(g^y)^{(p-1)/q}$ en base $g^{(p-1)/q}$ (comme dans l'algorithme de Pohlig-Hellman (5.5)). La difficulté vient du fait que la valeur de g^y n'est pas connue explicitement puisqu'il s'agit de x le logarithme discret recherché.

Nous ne disposons que de la valeur $g^x = g^{g^y}$ et il faut donc appliquer un algorithme de résolution de logarithme discret générique (par exemple l'algorithme de Shanks) implicitement « dans l'exposant ». Il est donc nécessaire de construire l'élément $(g^y)^{(p-1)/q}$ « dans l'exposant ». Il s'agit de l'élément

$$g^{(g^y)^{(p-1)/q}} = g^{x^{(p-1)/q}} = g^{x \times \dots \times x}.$$

La multiplication dans l'exposant est une opération *a priori* difficile mais c'est justement celle qui est réalisée par l'oracle qui résout le problème calculatoire de Diffie-Hellman. Nous obtenons donc l'algorithme suivant (7.4) qui applique l'algorithme de Shanks dans l'exposant une fois cette valeur calculée :

- La première étape de l'algorithme de Shanks (*i.e.* les « pas de bébé ») demande le calcul de g^i dans l'exposant, c'est-à-dire, le calcul de

$$g^{g^i \bmod p-1} \bmod p$$

pour $i \in \{0, \dots, T\}$ avec $T = \lceil \sqrt{q} \rceil + 1$. Elle demande donc T multiplications dans $(\mathbb{Z}/p\mathbb{Z})^*$ et T exponentiations dans \mathbb{G} .

- La deuxième étape est le calcul de l'élément $\mathfrak{t} = g^{q-T}$ dans l'exposant. Elle requiert une exponentiation dans $(\mathbb{Z}/p\mathbb{Z})^*$ et une exponentiation dans \mathbb{G} .

- La dernière étape (*i.e.* les « pas de géant ») demande le calcul de $g^{y(p-1)/q} \cdot t^i$ dans l'exposant, c'est-à-dire, le calcul de

$$g^{x(p-1)/q \cdot i} = g^{g^{(p-1)/q} \cdot t^i \bmod p-1 \bmod p}$$

pour $i \in \{0, \dots, \sqrt{q}\}$. Elle demande donc \sqrt{q} multiplications dans $(\mathbb{Z}/p\mathbb{Z})^*$ et \sqrt{q} exponentiations dans \mathbb{G} .

Le calcul de $g^{x(p-1)/q}$ se fait $O(\log(p))$ invocations de l'oracle qui résout le problème calculatoire de Diffie-Hellman en appliquant un algorithme d'exponentiation dichotomique dans l'exposant. Le coût total de l'algorithme pour obtenir la valeur de $y \bmod q$ est donc $O(\sqrt{q})$ exponentiations dans $(\mathbb{Z}/p\mathbb{Z})^*$, $O(\sqrt{q})$ exponentiations dans \mathbb{G} et $O(\log(p))$ exécutions de l'oracle qui résout le problème calculatoire de Diffie-Hellman.

Algorithme 7.4 Algorithme de Shanks dans l'exposant

ENTRÉE: $g, h \in \mathbb{G}, p = |\mathbb{G}|, q$ diviseur de $p - 1$.

SORTIE: $y \bmod q$ où $h = g^x$ et $x = g^y$ avec $y \in (\mathbb{Z}/(p-1)\mathbb{Z})$

$$n = \sum_{i=0}^{\ell-1} n_i 2^i \leftarrow (p-1)/q \quad \triangleright n_i \in \{0, 1\} \text{ pour } i \in \{0, \dots, \ell-1\}$$

$$\alpha \leftarrow 1_{\mathbb{G}}$$

pour i de $\ell - 1$ à 0 **faire**

$$\alpha \leftarrow \text{DIFFIE-HELLMAN}(\alpha, \alpha)$$

si $n_i = 1$ **alors**

$$\alpha \leftarrow \text{DIFFIE-HELLMAN}(\alpha, h)$$

fin si

fin pour

$$T \leftarrow \sqrt{q}$$

$$\Upsilon \leftarrow \emptyset$$

pour i de 0 à T **faire**

$$x_i \leftarrow g^{q^i} \quad \triangleright \text{une multiplication dans } (\mathbb{Z}/p\mathbb{Z})^* \text{ et une exponentiation dans } \mathbb{G}$$

$$\Upsilon \leftarrow \Upsilon \cup (x_i, i) \quad \triangleright \text{table de hachage}$$

fin pour

$$k \leftarrow g^{q^{q-T}} \quad \triangleright \text{une exponentiation dans } (\mathbb{Z}/p\mathbb{Z})^* \text{ et une exponentiation dans } \mathbb{G}$$

pour j de i à T **faire**

$$z \leftarrow \alpha^{q^{k-j}} \quad \triangleright \text{une multiplication dans } (\mathbb{Z}/p\mathbb{Z})^* \text{ et une exponentiation dans } \mathbb{G}$$

si $z = x_i$ avec $(x_i, i) \in \Upsilon$ **alors**

retourner $i + Tj \bmod q$

fin si

fin pour

4. L'adaptation de l'algorithme précédent pour le calcul de $y \bmod q^e$ lorsque q^e est une puissance d'un nombre premier divisant $p - 1$ est similaire à l'algorithme de Pohlig-Hellman (5.5). L'algorithme calcule successivement la valeur de $y \bmod q^i$ pour $i \in \{1, \dots, e\}$. La question précédente montre comment traiter le cas $i = 1$ et pour $i \geq 1$, le problème revient à calculer un logarithme discret en base $g^{x(p-1)/q^i}$, un élément qui peut

se calculer en $O(\log(p))$ exécutions de l'oracle qui résout le problème calculatoire de Diffie-Hellman.

- Si l'ordre du groupe \mathbb{G} est M -friable, nous obtenons un algorithme qui résout le problème du logarithme discret dans \mathbb{G} en utilisant un oracle qui résout le problème calculatoire de Diffie-Hellman dans \mathbb{G} un nombre de fois polynomial en $\log(p)$ et M . En particulier, pour des petites valeurs de M , nous obtenons que les deux problèmes sont équivalents.

7.4 CHIFFREMENT D'ELGAMAL ET VARIANTES

L'algorithme de chiffrement d'ElGamal est un algorithme de cryptographie asymétrique basé sur le problème du logarithme discret. Il a été créé par T. ELGAMAL [26].

Protocole de chiffrement d'ElGamal naïf

Génération des clés : l'utilisateur choisit un groupe \mathbb{G} d'ordre q dans lequel le problème du logarithme discret est jugé difficile et g un générateur de \mathbb{G} . Il tire uniformément aléatoirement $x \in (\mathbb{Z}/q\mathbb{Z})^*$ et calcule $y = g^x \in \mathbb{G}$. La clé publique est (q, g, y) et la clé secrète associée est x .

Chiffrement : étant donné un message clair $m \in \mathbb{G}$, l'algorithme de chiffrement tire uniformément aléatoirement $r \in (\mathbb{Z}/q\mathbb{Z})^*$ et calcule $c_1 = g^r \in \mathbb{G}$ et $c_2 = m \cdot y^r \in \mathbb{G}$. Le chiffré de m est le couple (c_1, c_2) .

Déchiffrement : étant donné un chiffré $(c_1, c_2) \in \mathbb{G}^2$, l'algorithme de déchiffrement retourne (c_2/c_1^x) .

Exercice 7.14 Sécurité du chiffrement d'ElGamal naïf

Considérons le protocole de chiffrement ElGamal naïf dans un groupe \mathbb{G} où le problème calculatoire de Diffie-Hellman est supposé difficile.

- Montrer que le protocole de chiffrement ElGamal n'est pas à sens unique sous une attaque à un chiffré choisi.
- Pour renforcer la sécurité du protocole de chiffrement ElGamal, nous proposons d'ajouter un élément supplémentaire c_3 (où \mathcal{H} est une fonction de hachage) pour ajouter de la redondance au chiffré. Évaluer, pour chacun des cas suivants, la sécurité du protocole de chiffrement résultant (où l'algorithme de vérification vérifie la redondance c_3 avant de retourner le texte clair).
 - $c_3 = \mathcal{H}(g^r)$
 - $c_3 = \mathcal{H}(y^r)$

- (c) $c_3 = \mathcal{H}(m)$
- (d) $c_3 = \mathcal{H}(m, g^r)$
- (e) $c_3 = \mathcal{H}(m, y^r)$

Solution

1. Étant donnés une clé publique (q, g, y) et un chiffré $(c_1, c_2) \in \mathbb{G}^2$, un attaquant peut utiliser la propriété de multiplicativité du chiffrement d'ElGamal naïf pour produire un chiffré (c'_1, c'_2) associé au même texte clair que (c_1, c_2) en calculant

$$c'_1 = c_1 \cdot g^s \text{ et } c'_2 = c_2 \cdot y^s$$

pour un $s \in (\mathbb{Z}/q\mathbb{Z})^*$ arbitraire. En effet, si $c_1 = g^r \in \mathbb{G}$ et $c_2 = m \cdot y^r \in \mathbb{G}$, nous avons $c'_1 = g^{r+s} \in \mathbb{G}$ et $c'_2 = m \cdot y^{r+s} \in \mathbb{G}$. Le chiffré (c'_1, c'_2) est différent du chiffré (c_1, c_2) et l'attaquant peut demander le déchiffrement de (c'_1, c'_2) et obtenir le message m .

2. (a) Dans ce premier cas, la valeur c_3 peut s'obtenir publiquement à partir de c_1 et l'attaque de la question précédente s'applique sans modification. Ce système de chiffrement n'est pas à sens unique sous une attaque à un chiffré choisi.
- (b) Ajouter la valeur $c_3 = \mathcal{H}(y^r)$ au chiffré ElGamal empêche l'attaque de la question 1 mais il est possible de l'adapter. En utilisant les propriétés de multiplicativité du chiffrement d'ElGamal naïf, un attaquant peut produire un chiffré (c_1, c'_2, c_3) associé à un texte clair différent mais avec $c'_2 = c_2 \cdot g \neq c_2$. Le logarithme discret r est défini par c_1 , donc le déchiffrement de (c_1, c'_2, c_3) sera effectué par l'algorithme de déchiffrement et retourne le texte clair $m' = g \cdot m$ si (c_1, c_2, c_3) est le chiffré de m . L'attaquant obtient donc facilement m et ce système de chiffrement n'est pas à sens unique sous une attaque à un chiffré choisi.
- (c) Ce troisième type de redondance ne renforce pas la sécurité du protocole du chiffrement d'ElGamal naïf contre les attaques à chiffré choisi mais détruit également la sécurité sémantique du schéma. En effet, étant donnés deux textes clairs m_0 et m_1 et le chiffré $c = (c_1, c_2, c_3)$ de l'un d'entre eux, il est possible d'appliquer la fonction de hachage \mathcal{H} à m_0 et à m_1 et de regarder quel chiffré est égal c_3 .
- (d) Comme pour le cas précédent, ce type de redondance détruit la sécurité sémantique du schéma. En effet, étant donnés deux textes clairs m_0 et m_1 et le chiffré $c = (c_1, c_2, c_3)$ de l'un d'entre eux, il est possible d'appliquer la fonction de hachage \mathcal{H} à (m_0, c_1) et à (m_1, c_1) et de regarder quelle empreinte est égale c_3 .
- (e) *A priori* il n'existe pas d'attaque à chiffré choisi contre le chiffrement d'ElGamal utilisant cette redondance (et il est possible de le montrer dans un modèle idéal où la fonction de hachage utilisée se comporte comme une fonction parfaitement aléatoire).

La sécurité vient de fait que si un attaquant modifie l'un des éléments du chiffré, il lui est difficile de calculer la redondance correspondante. Il semble donc qu'un attaquant ne peut construire un chiffré valide que s'il connaît le message clair correspondant (et l'oracle de déchiffrement lui est donc inutile).

Dans les exercices suivants, nous allons considérer le système de chiffrement d'ElGamal naïf dans le groupe $\mathbb{G} = (\mathbb{Z}/p\mathbb{Z})^*$ pour un nombre premier p .

Exercice 7.15 Sécurité des bits du logarithme discret

Soit p un nombre premier avec $p \equiv 3 \pmod{4}$ et soit g un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$. Posons

$$f : (\mathbb{Z}/(p-1)\mathbb{Z}) \longrightarrow (\mathbb{Z}/p\mathbb{Z})^*, \quad x \longmapsto f(x) = g^x \pmod{p}$$

$$\ell : (\mathbb{Z}/(p-1)\mathbb{Z}) \longrightarrow \{0, 1\}, \quad x \longmapsto \ell(x) = x \pmod{2}$$

$$m : (\mathbb{Z}/(p-1)\mathbb{Z}) \longrightarrow \{0, 1\}, \quad x \longmapsto m(x) = \begin{cases} 0 & \text{si } 0 \leq x < (p-1)/2 \\ 1 & \text{si } (p-1)/2 \leq x < p-1 \end{cases}$$

- Montrer qu'il existe un algorithme polynomial qui prenant en entrée $(p, g, f(x))$ retourne la valeur $\ell(x)$. En déduire que le chiffrement d'ElGamal naïf n'est pas sémantiquement sûr dans $(\mathbb{Z}/p\mathbb{Z})^*$.
- Montrer que s'il existe un algorithme polynomial qui prenant en entrée $(p, g, f(x))$ retourne la valeur $m(x)$ alors il existe un algorithme polynomial qui prenant en entrée $(p, g, f(x))$ retourne la valeur x .

Solution

- Nous avons g^x est un carré dans $(\mathbb{Z}/p\mathbb{Z})^*$ si et seulement si x est pair et

$$\ell(x) = 0 \iff \left(\frac{f(x)}{p}\right) = 1 \quad \text{soit} \quad \ell(x) = \left(1 - \left(\frac{f(x)}{p}\right)\right)/2$$

et ce symbole de Legendre peut être calculé en temps quadratique (en utilisant la loi de réciprocité quadratique).

Étant donné un chiffré ElGamal $(c_1, c_2) \in (\mathbb{Z}/p\mathbb{Z})^*$ pour une clé publique $(p-1, g, y)$ ElGamal, un attaquant peut calculer la parité du logarithme discret x de y en base g et la parité du logarithme discret de r de c_1 en base g . Avec ces notations $c_2 = m \cdot y^r = m \cdot g^{xr} \pmod{p}$, et l'attaquant peut calculer la parité du logarithme discret de c_2 en base g et obtenir la parité du logarithme discret de m en base g . Un adversaire est donc capable d'obtenir un bit d'information sur un message clair associé à un message chiffré donné et le protocole de chiffrement d'ElGamal naïf dans $(\mathbb{Z}/p\mathbb{Z})^*$ n'est pas sémantiquement sûr.

- Posons $y = f(x) = g^x \pmod{p}$. À partir de y nous pouvons calculer la parité de x en utilisant le résultat de la question précédente.

Le principe de l'algorithme est de calculer la valeur de x bit par bit (du bit le moins significatif vers le bit le plus significatif) par extractions de racines carrées successives. En effet si

$$x = \sum_{i=0}^{\ell} x_i 2^i$$

nous avons

$$y \cdot g^{-x_0} = g^{2 \cdot \sum_{i=1}^{\ell} x_i 2^{i-1}}$$

Le carré $y \cdot g^{-x_0}$ a deux racines carrées que l'on peut calculer en temps polynomial déterministe puisque $p \equiv 3 \pmod{4}$ (cf. Exercice 6.18). Si nous obtenons la racine $g^{\sum_{i=1}^{\ell} x_i 2^{i-1}}$, nous pouvons itérer la méthode et obtenir de proche en proche la valeur de x_i pour $i \in \{1, \dots, \ell\}$.

L'élément $z = yg^{-x_0}$ a deux racines carrées $g^{\sum_{i=1}^{\ell} x_i 2^{i-1}} \pmod{p}$ (la racine carrée « principale ») et $(-1) \cdot g^{\sum_{i=1}^{\ell} x_i 2^{i-1}} \pmod{p}$. Nous avons

$$\begin{aligned} \log_g(g^{\sum_{i=1}^{\ell} x_i 2^{i-1}}) &= \sum_{i=1}^{\ell} x_i 2^{i-1} \in \left\{0, \dots, \frac{p-3}{2}\right\} \\ \text{et} \quad \log_g((-1) \cdot g^{\sum_{i=1}^{\ell} x_i 2^{i-1}}) &= \log_g(g^{(p-1)/2} \cdot g^{\sum_{i=1}^{\ell} x_i 2^{i-1}}) \\ &= \frac{p-1}{2} + \sum_{i=1}^{\ell} x_i 2^{i-1} \in \left\{\frac{p-1}{2}, \dots, p-1\right\} \end{aligned}$$

Nous pouvons donc calculer $z^{(p+1)/4} \pmod{p}$ qui est l'une des racines carrés de z et s'il existe un algorithme \mathcal{A} qui calcule $m(x)$ à partir de $f(x)$, nous pouvons exécuter \mathcal{A} sur $z^{(p+1)/4} \pmod{4}$ et

- si \mathcal{A} retourne 0, alors $z^{(p+1)/4} = g^{\sum_{i=1}^{\ell} x_i 2^{i-1}}$ est la racine carrée « principale » de z ;
- si \mathcal{A} retourne 1, alors $(-1)z^{(p+1)/4}$ est la racine carrée « principale » de z .

Nous obtenons alors l'algorithme 7.5 qui s'exécute en temps polynomial si \mathcal{A} s'exécute en temps polynomial.

Algorithme 7.5 Inversion du logarithme discret à partir de \mathcal{A} calculant δ

ENTRÉE: $y \in (\mathbb{Z}/p\mathbb{Z})^*$

SORTIE: $x \in (\mathbb{Z}/q\mathbb{Z})$ tel que $y = g^x$.

```

 $x_0 \leftarrow \left(1 - \left(\frac{y}{p}\right)\right)/2 \quad \triangleright x_0 \equiv x \pmod{2}$ 
 $x \leftarrow x_0$ 
 $z \leftarrow y \cdot g^{-x_0} \quad \triangleright z \text{ est un carré modulo } p$ 
 $i \leftarrow 1$ 
tant que  $z \neq 1$  faire
     $z \leftarrow z^{(p+1)/4} \pmod{p} \quad \triangleright$  racine carrée de  $z$ 
     $b \leftarrow \mathcal{A}(z) \quad \triangleright b$  est le bit de poids fort de  $\log_g(z)$ 
    si  $b = 1$  alors
         $z \leftarrow p - z \quad \triangleright$  racine carrée « principale » de  $z$ 
    fin si
     $x_i \leftarrow \left(1 - \left(\frac{z}{p}\right)\right)/2$ 
     $x \leftarrow x + x_i 2^i$ 
     $z \leftarrow z \cdot g^{-x_i 2^i}$ 
     $i \leftarrow i + 1$ 
fin tant que
retourner  $x$ 

```

Pour obtenir la sécurité sémantique du système de chiffrement d'ElGamal naïf il est donc nécessaire de se placer dans le sous-groupe des carrés de $(\mathbb{Z}/p\mathbb{Z})^*$ (pour que le symbole de Legendre ne révèle pas d'information sur le message m chiffré).

Pour améliorer l'efficacité du système de chiffrement d'ElGamal naïf dans $(\mathbb{Z}/p\mathbb{Z})^*$, il a été suggéré utiliser d'un sous-groupe \mathbb{G} d'ordre q de $(\mathbb{Z}/p\mathbb{Z})^*$ (où q divise $(p-1)/2$). Pour assurer que le problème du logarithme discret est difficile dans \mathbb{G} , il faut que p soit suffisamment grand pour empêcher les attaques par calcul d'indice et que q soit suffisamment grand pour empêcher les attaques génériques comme l'algorithme de Shanks ou l'algorithme ρ de Pollard. Le réseau d'excellence européen *Ecrypt* en cryptographie recommande par exemple pour une sécurité de 128 bits de prendre un nombre premier p de 3248 bits et un diviseur q de $p-1$ de 256 bits.

L'exercice suivant montre que si le chiffrement d'ElGamal naïf est utilisé dans ce contexte, il est nécessaire de vérifier que le chiffré est effectivement formé d'éléments de \mathbb{G} .

Exercice 7.16 Attaque sur le chiffrement ElGamal par petit sous-groupe

Soit p un nombre premier impair et soit q un nombre premier divisant $(p-1)/2$.

Considérons le système de chiffrement d'ElGamal implanté dans le sous-groupe \mathbb{G} d'ordre q de $(\mathbb{Z}/p\mathbb{Z})^*$ et g un générateur de \mathbb{G} .

- Montrer que si un utilisateur accepte de déchiffrer tout message pour sa clé publique $y = g^x$ sans vérifier que le chiffré appartient à \mathbb{G}^2 , un attaquant peut apprendre la parité de x .
- Montrer que s'il existe des nombres premiers $\ell_1, \dots, \ell_t \leq T$ deux à deux distincts tels que $\ell_1 \dots \ell_t \mid (p-1)/q$ et $\ell_1 \dots \ell_t > q$, alors l'attaquant peut retrouver la valeur de x en $O(t\sqrt{T})$ opérations dans $(\mathbb{Z}/p\mathbb{Z})^*$.

Solution

- Par hypothèse, si l'utilisateur est interrogé sur un couple $(c_1, c_2) \in (\mathbb{Z}/p\mathbb{Z})^*$, il retourne toujours $c_2/c_1^x \bmod p$. En demandant le déchiffrement de $(p-1, c_2)$ pour une valeur c_2 arbitraire, l'utilisateur retourne $c_2/(p-1)^x \equiv c_2(-1)^x \bmod p$ et l'attaquant peut en déduire la parité de x .
- La méthode est proche de l'algorithme de Pohlig-Hellman (*cf.* Exercice 5.4). L'attaquant peut produire des générateurs g_i des sous-groupes d'ordre ℓ_i de $(\mathbb{Z}/p\mathbb{Z})^*$ pour $i \in \{1, \dots, t\}$. Si il demande le déchiffrement de (g_i, c_2) pour une valeur c_2 arbitraire pour $i \in \{1, \dots, t\}$, l'attaquant obtient l'élément $g_i^x \bmod p$ et en calculant le logarithme discret de cet élément dans le sous-groupe d'ordre ℓ_i (en $O(\sqrt{\ell_i}) = O(\sqrt{T})$ multiplications dans $(\mathbb{Z}/p\mathbb{Z})^*$), il obtient la valeur de x modulo ℓ_i .

En appliquant le théorème chinois des restes, l'attaquant obtient $x \bmod \ell_1 \dots \ell_t$, et puisque $\ell_1 \dots \ell_t > q$, il obtient la valeur de x . Si l'utilisateur ne vérifie pas que le chiffré est effectivement formé d'éléments de \mathbb{G} , le système de chiffrement d'ElGamal implanté dans le sous-groupe \mathbb{G} n'est pas résistant au bris total sous une attaque à chiffrés choisis.

SIGNATURES NUMÉRIQUES

Puisque l'authentification de documents est à la base des systèmes d'échanges commerciaux (contrats, factures, documents notariés), W. DIFFIE et M. HELLMAN ont mis en exergue, dès 1976, la nécessité de développer un procédé d'authentification numérique offrant les mêmes garanties que les signatures manuscrites, à savoir l'authenticité et l'intégrité du message signé (*i.e.* l'assurance que des données n'ont pas été modifiées par un moyen inconnu ou non autorisé et qu'elles proviennent bien de la source revendiquée) ainsi que la non-répudiation (*i.e.* l'assurance qu'une entité ne peut pas démentir des actions ou des engagements passés). Cette dernière propriété est généralement assurée par une procédure de vérification publique, tandis que les deux premières imposent qu'une seule *entité* doit être capable de produire des couples message/signature acceptés par ce processus. En particulier, il doit être impossible de déduire le moyen de signer à partir du procédé de vérification et par conséquent ce type de protocole doit être *asymétrique*.

Nous analyserons tout d'abord plusieurs protocoles de signature numériques fondés sur la *fonction RSA*. Nous étudierons ensuite les propriétés de sécurité du protocole de *signatures d'ElGamal* et de certaines variantes dont la sécurité repose sur la difficulté du problème du logarithme discret dans un groupe. Enfin, nous étudierons le schéma de *signature de Lamport* fondé sur l'existence d'une fonction à sens unique et nous discuterons l'efficacité et la sécurité de variantes de ce protocole.

8.1 SIGNATURES BASÉES SUR LA PRIMITIVE RSA

La première réalisation des signatures numériques a été proposée par R. RIVEST, A. SHAMIR et L. ADLEMAN en 1978. Initialement, elle a été décrite sous la forme suivante appelée *protocole de signature RSA naïf* :

Protocole de signature RSA naïf

Génération des clés : le signataire tire aléatoirement deux nombres premiers p et q et calcule le module RSA $N = pq$. Il calcule la fonction indicatrice d'Euler de N , $\varphi(N) = (p - 1)(q - 1)$. Il choisit un exposant public $e > 1$ premier à $\varphi(N)$ et calcule d l'inverse de e modulo $\varphi(N)$. La clé publique est le couple (N, e) et la clé secrète est l'entier d .

Signature : étant donné un message $m \in (\mathbb{Z}/N\mathbb{Z})^*$, le signataire calcule la signature $\sigma \equiv m^d \pmod{N}$.

Vérification : étant donné un message $m \in (\mathbb{Z}/N\mathbb{Z})^*$ et une signature supposée $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$, l'algorithme de vérification accepte σ si et seulement si $\sigma^e \equiv m \pmod{N}$.

La notion de sécurité intuitive que doit garantir un protocole de signature numérique est l'impossibilité pour un adversaire de créer une signature pour une clé publique sans la connaissance de sa clé privée associée. Les objectifs de l'adversaire, par ordre de difficulté décroissante, peuvent donc être :

- le *bris total* : l'adversaire retrouve la clé privée de signature ;
- la *contrefaçon universelle* : l'adversaire est capable de signer tous les messages ;
- la *contrefaçon existentielle* : l'adversaire est capable de créer un couple message/signature.

Pour un schéma de signature numérique, nous distinguons généralement trois types d'attaques :

- les *attaques sans message* : l'adversaire dispose uniquement de la clé publique du signataire dont il veut contrefaire une signature ;
- les *attaques à messages connus* : l'adversaire dispose de la clé publique du signataire, ainsi que de couples message/signature créés par le signataire ;
- les *attaques à messages choisis* : l'adversaire dispose de la clé publique du signataire, ainsi que de couples message/signature créés par le signataire, où les messages sont choisis par l'attaquant en fonction de la clé publique et des signatures obtenues précédemment.

Exercice 8.1 Sécurité du protocole de signature RSA naïf

1. Montrer que le protocole de signature RSA naïf n'est pas résistant à la contrefaçon existentielle sous une attaque sans message.
2. Montrer que le protocole de signature RSA naïf n'est pas résistant à la contrefaçon universelle sous une attaque à un message choisi.

Solution

1. Puisque $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$ est une signature valide de $m \in (\mathbb{Z}/N\mathbb{Z})^*$ si et seulement si $\sigma^e \equiv m \pmod{N}$, pour produire une contrefaçon existentielle, il suffit à un attaquant de tirer σ aléatoirement dans $(\mathbb{Z}/N\mathbb{Z})^*$ et de calculer $m \equiv \sigma^e \pmod{N}$. Le couple (m, σ) est alors accepté par l'algorithme de vérification.

2. Pour obtenir une signature sur un message m , un attaquant peut simplement choisir un élément $r \in (\mathbb{Z}/N\mathbb{Z})^*$ et demander la signature σ de $m \cdot r^e \bmod N$. Dans ce cas, nous avons $\sigma^e = m \cdot r^e \bmod N$ et donc $(\sigma/r)^e = m \bmod N$ et $\sigma/r \bmod N$ est une signature de m .

Le protocole RSA naïf est donc sujet à des attaques dévastatrices. En 1986, W. DE JONGE et D. CHAUM [19] ont proposé des variantes de ce protocole pour essayer de le rendre résistant aux contrefaçons. L'exercice suivant analyse la sécurité de deux d'entre elles.

Exercice 8.2 Sécurité des protocoles de signature de De Jonge et Chaum

Nous considérons deux variantes du schéma de signature RSA où le module $N = pq$ est le produit de deux nombres premiers p et q dits *forts* (*i.e.* $p = 2p' + 1$ et $q = 2q' + 1$ où p' et q' sont premiers).

1. Supposons que la signature d'un message impair $m \in \mathbb{Z}$ est l'entier $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$, s'il existe, tel que $\sigma^m = m \bmod N$. Montrer que ce schéma n'est pas résistant à une contrefaçon universelle sous une attaque à deux messages choisis.
2. Supposons que la signature d'un message $m \in \mathbb{Z}$ est l'entier $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$, s'il existe, tel que $\sigma^{2m+1} = m \bmod N$. Montrer que ce schéma n'est pas résistant à une contrefaçon universelle sous une attaque à deux messages choisis.

Indication

On pourra chercher deux messages m_1 et m_2 (dépendants de m) et deux entiers a et b tels que $\sigma = \sigma_1^a / \sigma_2^b$ soit une signature valide de m (où σ_1 et σ_2 sont les signatures de m_1 et m_2).

Solution

1. Supposons que l'adversaire obtienne les signatures σ_1 et σ_2 des messages $m_1 = m\alpha$ et $m_2 = m^2\alpha$ pour $\alpha \in \mathbb{Z}$ impair. Nous avons

$$\sigma_1^\alpha = (m\alpha)^{m^{-1}} \bmod N \text{ et } \sigma_2^{m\alpha} = (m^2\alpha)^{m^{-1}} \bmod N$$

et donc $\sigma = \sigma_2^{m\alpha} / \sigma_1^\alpha \bmod N$ est une signature valide de m . En effet, nous avons

$$\sigma^m = \frac{\sigma_2^{m^2\alpha}}{\sigma_1^{m\alpha}} = \frac{m^2\alpha}{m\alpha} = m$$

2. L'attaquant cherche à obtenir une signature σ d'un message $m \in \mathbb{Z}$ telle que

$$\sigma^{2m+1} \equiv m \bmod N$$

Nous pouvons supposer sans perte de généralité que $2m + 1$ est premier à N (dans le cas contraire en calculant le PGCD de $2m + 1$ et N , l'attaquant peut factoriser le module RSA N et par conséquent obtenir la clé secrète du signataire).

L'attaquant doit construire deux messages m_1 et m_2 et, à partir de ces messages et de leurs signatures σ_1 et σ_2 , calculer la signature σ . Nous avons

$$\sigma_1^{2m_1+1} \equiv m_1 \pmod{N} \text{ et } \sigma_2^{2m_2+1} \equiv m_2 \pmod{N}$$

En cherchant une combinaison multiplicative de σ_1 et σ_2 pour construire σ sous la forme $\sigma = \sigma_1^a/\sigma_2^b$ pour des valeurs a et b à déterminer, nous avons

$$m = \sigma^{2m+1} = \sigma_1^{a(2m+1)}/\sigma_2^{b(2m+1)}$$

Il suffit donc de trouver deux messages m_1 et m_2 et deux entiers a et b tels que

$$m_1/m_2 \equiv m \pmod{N}$$

$$a(2m+1) = 2m_1 + 1$$

$$b(2m+1) = 2m_2 + 1$$

Pour toute valeur impaire de b , nous avons

$$b(2m+1) = 2bm + b = 2(bm + (b-1)/2) + 1$$

Nous pouvons donc poser $m_2 = (b(2m+1)-1)/2$ mod N pour un b impair. La condition $m_1/m_2 \equiv m \pmod{N}$ donne $m_1 = mm_2 + kN$ pour une valeur $k \in \mathbb{N}$. Puisque nous voulons également l'égalité $a(2m+1) = 2m_1 + 1$ pour un entier a à déterminer, nous obtenons modulo $2m+1$

$$2m_1 + 1 \equiv 2mm_2 + 2kN + 1 \equiv 0 \pmod{2m+1}$$

soit

$$k \equiv (-2mm_2 - 1) \cdot (2N)^{-1} \pmod{2m+1}$$

En posant k cette valeur et a l'entier tel que $2mm_2 + 2kn + 1 = a(2m+1)$, nous obtenons comme attendu $\sigma = \sigma_1^a/\sigma_2^b$, la relation

$$\sigma^{2m+1} = \frac{\sigma_1^{a(2m+1)}}{\sigma_1^{b(2m+1)}} = \frac{\sigma_1^{2mm_2+2kn+1}}{\sigma_2^{2m_2+1}} = \frac{\sigma_1^{2m_1+1}}{\sigma_1^{2m_1+1}} = \frac{m_2}{m_1} = m \pmod{N}$$

Plusieurs solutions ont été proposées pour construire des signatures basées sur la primitive RSA qui résistent à toutes les formes de contrefaçon. Elles utilisent généralement une fonction d'encodage $\mathcal{F} : \mathcal{M} \longrightarrow (\mathbb{Z}/N\mathbb{Z})^*$ qui casse les propriétés de morphisme de la fonction RSA (où \mathcal{M} désigne l'espace des messages à signer).

Protocole de signature \mathcal{F} -RSA

Génération des clés : le signataire tire aléatoirement deux nombres premiers p et q et calcule $N = pq$. Il calcule la fonction indicatrice d'Euler de N , $\varphi(N) = (p - 1)(q - 1)$. Il choisit un exposant public $e > 1$ premier à $\varphi(N)$ et calcule d l'inverse de e modulo $\varphi(N)$. La clé publique est le couple (N, e) et la clé secrète est l'entier d .

Signature : étant donné un message $m \in \mathcal{M}$, le signataire calcule la signature $\sigma \equiv \mathcal{F}(m)^d \pmod{N}$.

Vérification : étant donné un message $m \in \mathcal{M}$ et une signature supposée $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$, l'algorithme de vérification accepte σ si et seulement si $\sigma^e \equiv \mathcal{F}(m) \pmod{N}$.

Les propriétés de la fonction d'encodage \mathcal{F} doivent être étudiées avec soin pour assurer la sécurité du protocole de signature \mathcal{F} -RSA. L'exercice suivant montre notamment qu'elle doit être résistante à la pré-image, à la seconde pré-image et aux collisions.

Exercice 8.3 Sécurité de \mathcal{F} -RSA et propriétés de \mathcal{F}

- Montrer que si la fonction \mathcal{F} est une fonction de hachage qui n'est pas résistante à la pré-image alors le protocole \mathcal{F} -RSA n'est pas résistant à la contrefaçon existentielle sous une attaque sans message.
- Montrer que si la fonction \mathcal{F} est une fonction de hachage qui n'est pas résistante à la seconde pré-image alors le protocole \mathcal{F} -RSA n'est pas résistant à la contrefaçon universelle sous une attaque à un message choisi.
- Montrer que si la fonction \mathcal{F} est une fonction de hachage qui n'est pas résistante aux collisions alors le protocole \mathcal{F} -RSA n'est pas résistant à la contrefaçon existentielle sous une attaque à un message choisi.

Solution

- Puisque $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$ est une signature valide de $m \in \mathcal{M}$ si et seulement si $\sigma^e \equiv \mathcal{F}(m) \pmod{N}$, pour produire une contrefaçon existentielle, il suffit à un attaquant de tirer σ aléatoirement dans $(\mathbb{Z}/N\mathbb{Z})^*$ et de calculer $\mu \equiv \sigma^e \pmod{N}$. Si la fonction \mathcal{F} n'est pas résistante à la pré-image, l'attaquant peut trouver un message m tel que $\mathcal{F}(m) = \mu$ et le couple (m, σ) est alors accepté par l'algorithme de vérification.
- Supposons qu'un attaquant souhaite obtenir une signature sur un message m . Si la fonction \mathcal{F} n'est pas résistante à la seconde pré-image, il peut rechercher un message m' tel que $\mathcal{F}(m) = \mathcal{F}(m') \in (\mathbb{Z}/N\mathbb{Z})^*$. En demandant la signature de m' au signataire, l'attaquant obtient $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$ tel que $\sigma^e \equiv \mathcal{F}(m) = \mathcal{F}(m') \pmod{N}$, donc σ est également la signature de m .

3. Si un attaquant obtient une collision (m_1, m_2) pour la fonction \mathcal{F} (i.e. $\mathcal{F}(m_1) = \mathcal{F}(m_2) \in (\mathbb{Z}/N\mathbb{Z})^*$), puisque la procédure de vérification d'une signature ne dépend que de la valeur de l'empreinte du message, si l'attaquant obtient la signature σ de m_1 , σ est aussi immédiatement une signature de m_2 .

Lorsque la fonction \mathcal{F} est une fonction de hachage dans l'ensemble $(\mathbb{Z}/N\mathbb{Z})^*$ qui se comporte comme une fonction aléatoire, il a été montré que le protocole de signature \mathcal{F} -RSA est résistant aux contrefaçons existentielles sous une attaque à messages choisis.

Le Comité Consultatif International pour le Téléphone et le Télégraphe¹ (CCITT) a proposé en 1988 d'utiliser la fonction de hachage suivante pour le système \mathcal{F} -RSA.

Fonction de hachage du CCITT

Étant donné un module RSA de $\ell + 1$ bits, la fonction de hachage \mathcal{H} de la recommandation CCITT fonctionne de la façon suivante :

- le message m est divisé en t blocs m_1, \dots, m_t de $\ell/2$ bits. Si le dernier bloc fait moins de $\ell/2$, il est complété par des uns pour atteindre $\ell/2$ bits ;
- chaque demi-octet d'un bloc m_i est complété par quatre uns sur la gauche (produisant ainsi des blocs M_i de ℓ bits, cf. Figure 8.1) ;

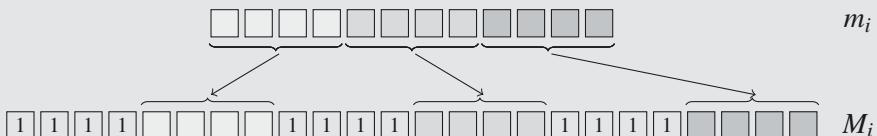


Figure 8.1 - Extension des blocs dans la fonction de hachage du CCITT

- les blocs M_i sont ensuite compressés en partant d'une valeur $H_0 = 0$, et en calculant successivement² $H_i = (H_{i-1} \oplus M_i)^2 \bmod N$ pour $i \in \{1, \dots, t\}$. L'empreinte finale $\mathcal{H}(m)$ est la valeur H_t .

Cette fonction semble résistante aux collisions (au moins sur un bloc de message) mais l'exercice suivant montre cependant qu'il est possible de produire des contrefaçons existentielles sur le schéma \mathcal{F} -RSA si cette fonction est utilisée pour fonction d'encodage \mathcal{F} .

1. Il s'agit d'un groupe de travail de l'organisation intergouvernementale *Union internationale des télécommunications (International Telecommunication Union en anglais)* rattachée aux Nations unies.
2. Il est important de noter que dans cette étape de calcul, l'algorithme mélange une opération sur les bits (le \oplus) et une opération sur les entiers (le carré).

Exercice 8.4 Sécurité de \mathcal{F} -RSA pour la recommandation CCITT

Soit N un module RSA de $\ell + 1$ bits et soit \mathcal{H} la fonction de hachage du CCITT correspondante.

1. Soient deux messages (m_1, \dots, m_{t-1}) et (m'_1, \dots, m'_{t-1}) (avec chaque bloc de longueur $\ell/2$ bits). Montrer que si le haché H'_{t-1} de (m'_1, \dots, m'_{t-1}) commence par quatre bits égaux à 1, il est possible de trouver deux blocs m_t et m'_t de $\ell/2$ bits tels que $\mathcal{H}(m) = 256\mathcal{H}(m')$ mod N avec $m = (m_1, \dots, m_t)$ et $m' = (m'_1, \dots, m'_t)$.
2. En déduire une contrefaçon existentielle sous une attaque à trois messages choisies contre le protocole \mathcal{F} -RSA lorsque la fonction d'encodage \mathcal{F} est la recommandation CCITT \mathcal{H} .

Solution

1. Puisque trouver deux carrés égaux modulo N permet la factorisation de N , il semble difficile de trouver deux valeurs distinctes telles que $H_t = 256H'_t$. Par conséquent, nous allons essayer de construire deux blocs de messages m_t et m'_t tels que

$$H_{t-1} \oplus M_t = 16(H'_{t-1} \oplus M'_t)$$

Une telle valeur s'obtient si, en interprétant les valeurs de H_{t-1} , M_t , H'_{t-1} et M'_t en chaînes de bits, la valeur de $H_{t-1} \oplus M_t$ est égale à celle de $H'_{t-1} \oplus M'_t$ suivie de quatre zéros :

- Pour faire terminer la valeur $H_{t-1} \oplus M_t$ par quatre zéros, il suffit de fixer les quatre derniers bits de M_t aux valeurs des quatre derniers bits de H_{t-1} . Ces quatre derniers bits sont ceux du dernier bloc de message m_t .
- Il faut ensuite faire coïncider les quatre derniers bits de $H'_{t-1} \oplus M'_t$ avec les bits aux positions $\{5, 6, 7, 8\}$ de $H_{t-1} \oplus M_t$. Ces bits sont égaux aux complémentaires des bits de H_{t-1} puisque M_t prend la valeur 1 aux positions $\{5, 6, 7, 8\}$. Il suffit donc de poser pour les quatre derniers bits de m'_t les valeurs $(1 \oplus H_{t-1}[i+4]) \oplus H'_{t-1}[i]$ pour $i \in \{1, 2, 3, 4\}$.
- De même, il faut ensuite faire coïncider les bits aux positions $\{9, 10, 11, 12\}$ de $H_{t-1} \oplus M_t$ avec les bits aux positions $\{5, 6, 7, 8\}$ de $H'_{t-1} \oplus M'_t$. Ces bits sont égaux aux complémentaires des bits de H'_{t-1} puisque M'_t prend la valeur 1 aux positions $\{5, 6, 7, 8\}$. Il suffit donc de poser pour les bits aux positions $\{5, 6, 7, 8\}$ de m_t les valeurs correspondantes à $(1 \oplus H'_{t-1}[i]) \oplus H_{t-1}[i+4]$ pour $i \in \{5, 6, 7, 8\}$.

De proche en proche, il est possible de construire deux blocs de messages m_t et m'_t (cf. Figure 8.2). La condition sur les quatre premiers bits de H'_{t-1} assure que nous avons

$$H_{t-1} \oplus M_t = 16(H'_{t-1} \oplus M'_t) \text{ et donc } \mathcal{H}(m) = 256\mathcal{H}(m') \text{ mod } N$$

avec $m = (m_1, \dots, m_t)$ et $m' = (m'_1, \dots, m'_t)$.

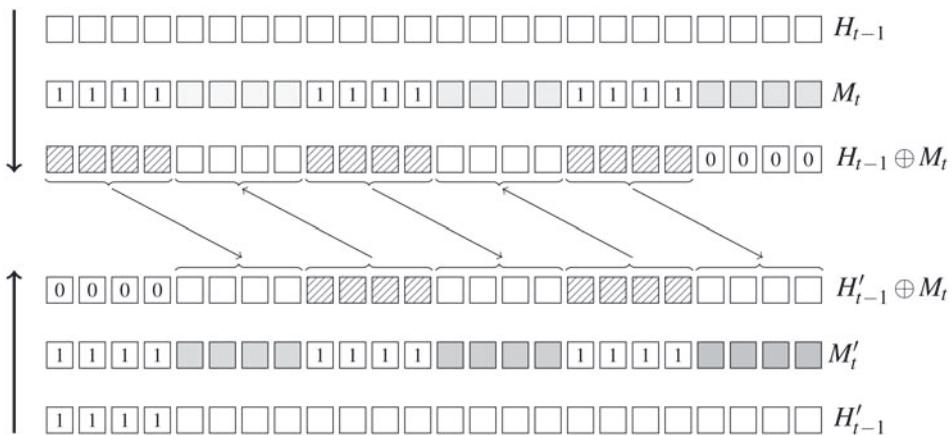


Figure 8.2 – Construction des messages tels que $\mathcal{H}(m) = 256\mathcal{H}(m')$

2. Il suffit pour l'attaquant de construire deux couples de messages (m_1, m'_1) et (m_2, m'_2) tels que $\mathcal{H}(m_1) = 256\mathcal{H}(m'_1)$ et $\mathcal{H}(m_2) = 256\mathcal{H}(m'_2)$. La condition sur les bits de poids fort de H'_{t-1} est vérifiée avec probabilité 1/16 et le choix des premiers blocs de message peut être fait librement sous cette seule condition. Les derniers blocs de chaque message sont construits en utilisant la question précédente.

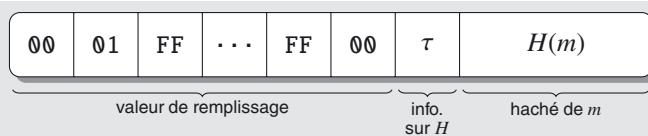
Puisque $\mathcal{H}(m_1)/\mathcal{H}(m_2) = \mathcal{H}(m'_1)/\mathcal{H}(m'_2)$, l'attaquant peut simplement demander la signature des trois messages m_1, m'_1, m_2 pour obtenir la signature de m_2 par multiplicativité de la fonction RSA.

En cryptographie asymétrique, *PKCS #1* (de l'anglais *Public-Key Cryptography Standards*) est une famille de normalisation de systèmes de cryptographie à clé publique proposée par les *RSA Laboratories*. Il fournit notamment les définitions et les recommandations pour implanter la primitive RSA en cryptographie à clé publique. La version PKCS #1 v1.5 de ces recommandations propose un standard pour la signature RSA.

Encodage PKCS #1 v1.5 pour la signature \mathcal{F} -RSA

L'encodage PKCS #1 v1.5 pour la signature opère de la façon suivante :

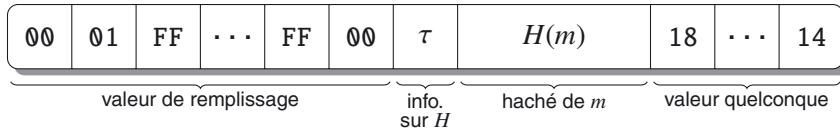
- une empreinte $\mathcal{H}(m)$ du message à signer m est calculée avec une fonction de hachage \mathcal{H} (cette empreinte est en pratique beaucoup plus petite que le module RSA en utilisant par exemple MD5 ou SHA-1) ;
- l'entier $\mathcal{F}(m)$ sur lequel est appliquée la permutation RSA est alors la concaténation de l'octet 00, de l'octet 01, d'une suite d'au moins 8 octets FF, un octet 00 puis une chaîne τ et enfin l'empreinte $\mathcal{H}(m)$.



La chaîne τ contient le descriptif (*i.e.* l'identifiant ASN.1) de la fonction de hachage \mathcal{H} utilisée. La vérification d'une signature σ se fait alors

- en calculant $s = \sigma^e \bmod N$, en retirant la chaîne 00 01 FF...FF 00 ;
- en obtenant le type de fonction de hachage utilisée \mathcal{H} dans l'information τ ;
- en vérifiant que la partie restante coïncide avec $\mathcal{H}(m)$.

Certaines implantations de l'algorithme de vérification ne contrôlaient pas (dans le dernier point) que la partie restante coïncidait avec $\mathcal{H}(m)$. À la place, elles utilisaient la longueur de l'empreinte encodée dans l'identifiant τ pour extraire cette longueur de la partie restante (sans s'assurer que l'empreinte était effectivement à droite dans le message). En particulier, pour ce type d'implantation, si la valeur obtenue s est de la forme suivante :



notée (E), la signature est acceptée par l'algorithme de vérification.

Exercice 8.5 Sécurité de \mathcal{F} -RSA avec encodage PKCS #1 v1.5

Considérons la signature \mathcal{F} -RSA avec encodage PKCS #1 v1.5. Nous allons montrer que si l'implantation est incorrecte (comme mentionné ci-dessus), alors le schéma n'est pas résistant aux contrefaçons (pour un exposant public $e = 3$).

- Montrer que si le signataire utilise comme exposant public $e = 3$ alors il est possible de construire une contrefaçon qui sera acceptée par l'algorithme de vérification erroné.

Indication

L'attaquant pourra construire un cube parfait de la forme (E).

- Calculer la probabilité de succès de cette attaque, pour un module RSA de 1024 bits, lorsque

- (a) la fonction \mathcal{H} est la fonction MD5 qui retourne des empreintes de 128 bits et pour lequel τ est codé sur 144 bits.
- (b) la fonction \mathcal{H} est la fonction SHA-1 qui retourne des empreintes de 160 bits et pour lequel τ est codé sur 120 bits.
3. Supposons que l'attaquant Carole souhaite obtenir une signature valide sur un chèque identifié par un numéro de série et un ordre, par exemple, un message de la forme

$m = \text{Cheque } \# ? ? ? ? \text{ de } 10000 \text{ euros pour Carole}$

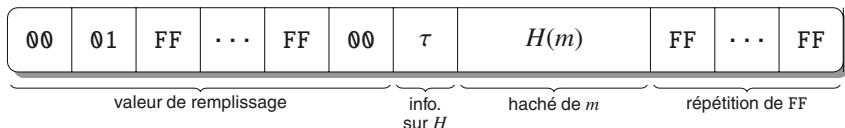
où les points d'interrogation sont remplacés par un entier entre 1 et 10000. Programmer cette attaque en supposant que la fonction de hachage \mathcal{H} utilisée est MD5 dont le descriptif τ (en hexadécimal) est égal à

3020300C06082A864886F70D020505000410

Solution

1. Le principe de l'attaque pour produire une fausse signature sur le message m est le suivant :

- une empreinte $\mathcal{H}(m)$ du message à signer m est calculée avec une fonction de hachage \mathcal{H} ;
- l'entier $\tilde{\mathcal{F}}(m)$ est défini comme la concaténation de l'octet 00, de l'octet 01, d'une suite d'exactement 8 octets FF, de l'octet 00, de la chaîne τ , du haché $\mathcal{H}(m)$ et enfin d'une suite d'octets FF jusqu'à la longueur du module RSA attaqué ;



- si cet entier $\tilde{\mathcal{F}}(m)$ est compris entre deux cubes consécutifs d'entiers : $t^3 < \tilde{\mathcal{F}}(m) < (t + 1)^3$, alors t^3 sera de la forme indiquée. Dans ce cas, l'entier t sera considéré comme une signature valide du message m par l'algorithme de vérification erroné.

2. Pour un module RSA de 1024 bits, la taille complète de la signature est de 128 octets. La fonction de remplissage PKCS#1 v1.5 occupe au minimum

- 45 pour la fonction MD5 ;
- 46 octets pour la fonction SHA-1.

Il faut en effet deux octets pour l'encodage de 00 01, huit octets supplémentaires pour les FF, un octet pour 00, 18 octets et 15 octets respectivement pour l'encodage τ et enfin 16 octets et 20 octets respectivement pour la valeur de l'empreinte. et il reste

donc 83 octets pour la fonction MD5 et 82 octets pour la fonction SHA-1 pour les données liées au message (soit 664 et 656 bits respectivement).

Comme chaque encodage commence avec la chaîne **00 01 FF**, la valeur d'encodage est inférieure à 2^{1009} . La distance entre deux cubes parfaits consécutifs de cette taille est inférieure à 2^{673} . Puisque l'attaquant ne peut contrôler que 664 ou 656 bits de données, la probabilité d'obtenir un encodage entre deux cubes parfaits consécutifs est de l'ordre de 2^{-9} pour la fonction *MD5* et de l'ordre de 2^{-17} pour la fonction *SHA-1*. On s'attend donc à devoir répéter l'attaque pour 2^9 (*resp.* 2^{17}) messages avant d'obtenir un succès dans le premier cas (*resp.* dans le second cas).

3. Les entiers entre 1 et 10 000 pour lesquels l'attaque fonctionne sont 1670, 2041, 2482, 2809, 4554, 4700, 5207, 7518. Par exemple en considérant le message

m = Cheque #1670 de 10000 euros pour Carole

nous obtenons l'empreinte (en hexadécimal)

$\text{MD5}(m) = 4A954C5B0665FBCFA2C4680BD8CF2862$

et donc la valeur $\tilde{\mathcal{F}}(m)$

En calculant la partie entière inférieure de la racine cubique de $\tilde{\mathcal{F}}(m)$, nous obtenons

$$t = \lfloor \tilde{\mathcal{F}}(m)^{1/3} \rfloor = 176368846691323818197502760767911056469103568347044 \backslash \\ 699892187007081199379657489659639311392162614022368$$

et le codage hexadécimal de t^3 est

00001FFFFFFF003020300C06082A864886F70D0205050004104A954C
5B0665FBCFA2C4680BD8CF286252EFBDAB2AFF6C3C6C991DEF21EB0EE9FEBB3
372945147BB73A3AD9326D82D2B0CE3BC9E77C4B3B642A5ECC8DBC20855EA3030
D4E34D170B36710D6A1FF35648C11214A5A0B8A61BA211E63C4A44F7B96B8000

qui serait donc bien accepté par un algorithme de vérification erroné.

D'autres méthodes ont été proposées pour tenter de se prémunir des attaques sur les signatures basées sur la primitive RSA. Au lieu d'utiliser une fonction de hachage, elles proposent d'ajouter de la redondance au message. Une *fonction de redondance* \mathcal{F} est une fonction inversible prenant en entrée un message m et renvoyant en sortie une chaîne de bits. Dans l'exercice suivant nous allons considérer les signatures RSA avec une redondance linéaire de la forme $\mathcal{F}(m) = \omega \cdot m$ pour une valeur de ω fixée.

Un entier $\sigma \in (\mathbb{Z}/N\mathbb{Z})^*$ est une signature valide de m si et seulement si $(\sigma^e \bmod N)/\omega$ est l'entier m . Dans le cas où ω est une puissance de 2, l'encodage de m est de la forme suivante

Message m	00	00	...	00	00
-------------	----	----	-----	----	----

d'où le nom de *redondance à droite*. En 1985, W. DE JONGE et D. CHAUM [18] ont présenté des attaques contre le schéma \mathcal{F} – RSA utilisant cette fonction de redondance.

Exercice 8.6 Contrefaçon existentielle de \mathcal{F} -RSA avec redondance linéaire

Montrer que le protocole de signature \mathcal{F} -RSA n'est pas résistant aux contrefaçons existentielles sous une attaque à trois messages choisis si la fonction \mathcal{F} est la fonction de redondance linéaire.

Solution

Soit N un module RSA et soit ω la valeur utilisée pour la redondance linéaire. Considérons un message m de la forme $m = ab$ avec $a, b \geq 1$ et $\omega \cdot m < N$. Soient deux entiers b_1 et b_2 tels que $\omega \cdot a \cdot b_1 < N$, $\omega \cdot b \cdot b_2 < N$ et $\omega \cdot b_1 \cdot b_2 < N$. En posant $m_1 = a \cdot b_1$, $m_2 = b \cdot b_2$ et $m_3 = b_1 \cdot b_2$, l'attaquant peut demander les signatures σ_1 , σ_2 et σ_3 de m_1 , m_2 et m_3 (respectivement). Nous avons

$$\sigma_i^e \bmod N = \mathcal{F}(m_i) = \omega \cdot m_i$$

pour $i \in \{1, 2, 3\}$.

Comme $m = (m_1 m_2)/m_3$, nous avons

$$\left(\frac{\sigma_1 \sigma_2}{\sigma_3} \right)^e \bmod N = \frac{(\omega \cdot m_1)(\omega \cdot m_2)}{\omega \cdot m_3} = \omega \cdot m$$

et $\sigma = \sigma_1 \sigma_2 / \sigma_3$ est donc une signature valide de m .

L'exercice suivant montre que le système est vulnérable à une contrefaçon universelle sous une attaque à trois messages choisis si la redondance linéaire utilisée ω vérifie $\omega < \sqrt{N}$ où N est le module RSA.

Exercice 8.7 Contrefaçon universelle de \mathcal{F} -RSA avec redondance linéaire *

Soit N un module RSA.

1. Soit $X \in \{0, \dots, N-1\}$ un entier premier avec N et soit $T \in \mathbb{N}$. Montrer en utilisant le principe des tiroirs qu'il existe $\lambda \in \mathbb{Z}$ tel que $0 < |\lambda| \leq N/T$ pour lequel $0 \leq (\lambda X) \bmod N \leq T$.
2. En considérant le développement en fractions continues de X/N , proposer un algorithme qui, prenant en entrée X, N et T , retourne un entier λ vérifiant $0 < |\lambda| \leq N/T$ et $0 \leq (\lambda X) \bmod N \leq T$.
3. En déduire une contrefaçon universelle sous une attaque à deux messages choisis sur le schéma de signature RSA avec redondance linéaire $\omega \leq N^{1/2}$.

Solution

1. Considérons l'ensemble des valeurs $aX - b \bmod N$ lorsque $0 < a \leq \lceil N/T \rceil$ et $0 \leq b < T$. Il y a $T \cdot \lceil N/T \rceil > N$ couples (a, b) et seulement N valeurs possibles pour $aX - b \bmod N$. Par le principe des tiroirs, il existe deux couples distincts (a_1, b_1) et (a_2, b_2) avec $0 < a_i \leq \lceil N/T \rceil$ et $0 \leq b_i < T$ pour $i \in \{1, 2\}$ tels que

$$a_1X - b_1 \equiv a_2X - b_2 \bmod N$$

Nous avons donc $(a_1 - a_2)X \equiv (b_1 - b_2) \bmod N$. Comme X est premier avec N , nous avons $a_1 \neq a_2$ et $b_1 \neq b_2$. En supposant que $b_1 > b_2$ et en posant $\lambda = a_1 - a_2$, nous obtenons

$$0 < \lambda X \bmod N \equiv b_1 - b_2 < T \text{ et } 0 < |\lambda| \leq N/T$$

2. Considérons le développement en fractions continues de X/N et notons $(P_i)_i$ et $(Q_i)_i$ les numérateurs et les dénominateurs des réduites de X/N . En notant j l'unique entier tel que $Q_j < T \leq Q_{j+1}$, nous avons

$$\left| \frac{X}{N} - \frac{P_j}{Q_j} \right| \leq \frac{1}{Q_j Q_{j+1}}$$

et par conséquent $|XQ_j - NP_j| \leq N/Q_{j+1}$. En posant $\lambda = Q_j$, nous obtenons le résultat. Le développement en fraction continue de $\frac{X}{N}$ s'obtient facilement en appliquant l'algorithme d'Euclide à X et N .

3. Étant donné un message $m < N/\omega$, posons $M = \omega \cdot m < N$ et $X = M^{-1} \bmod N$. En appliquant l'algorithme précédent à X et N avec $T = N/\omega$, nous obtenons un entier λ tel que $\lambda X \leq N/\omega$ (ou $-\lambda X \leq N/\omega$) et $\lambda \leq \omega$. Puisque $\omega < \sqrt{N}$, nous avons $\lambda \leq N/\omega$. En posant $m_1 = \lambda \bmod N$ et $m_2 = \lambda X \bmod N$ (ou $m_2 = -\lambda m_1 \bmod N$), si l'attaquant demande la signature de m_1 et m_2 (notée σ_1 et σ_2), il obtient une signature de m en calculant $\sigma = \pm \sigma_1 / \sigma_2 \bmod N$ (le signe étant le même que celui utilisé pour m_2). En effet, nous avons alors

$$\sigma^e = \pm \left(\frac{\sigma_1}{\sigma_2} \right)^e = \frac{\omega \lambda}{(\pm \omega \lambda X)} = X^{-1} = M$$

En 1997, C. BOYD [12] a proposé un protocole de signature numérique dont la clé publique est un couple d'entiers (N, g) où N est un module RSA et g est un élément de $(\mathbb{Z}/N\mathbb{Z})^*$ qui engendre un sous-groupe d'ordre r (qui est la clé secrète du signataire). Le schéma est plus efficace que le schéma RSA car la génération des signatures est effectuée dans ce « petit » sous-groupe. Le but du prochain exercice est d'analyser la sécurité de ce schéma.

Problème 8.8 Sécurité du protocole de signature de Boyd

Nous considérons un protocole de signature numérique où la clé publique est un couple d'entiers (N, g) et la clé secrète est un entier r tels que

- (i) N est le produit de deux nombres premiers distincts p et q (*i.e.* N est un module RSA) ;
- (ii) r est un diviseur premier de $p - 1$;
- (iii) g est un élément d'ordre r dans $(\mathbb{Z}/N\mathbb{Z})^*$.

Nous notons k la taille en bits des nombres premiers p et q et ℓ la taille en bits de l'entier r . La signature σ d'un entier m de taille ℓ est la racine m -ième de g dans $(\mathbb{Z}/N\mathbb{Z})^*$ (*i.e.* $\sigma^m = g$).

1. Montrer que si r ne divise pas $q - 1$, alors la connaissance de (N, g) permet de factoriser l'entier N .

Nous supposerons dans la suite de l'exercice jusqu'à la question 4(a) que :

- (iv) r divise $q - 1$;
- 2. Proposer un algorithme probabiliste qui, prenant en entrée deux entiers k et ℓ , retourne un triplet (N, g, r) vérifiant les propriétés (i)–(iv) et comparer la complexité de l'algorithme de signature avec celle de la signature RSA classique.

3. Bris total.

- (a) Donner un algorithme de complexité $O(2^{\ell/2})$ opérations dans le groupe $(\mathbb{Z}/N\mathbb{Z})^*$ permettant de retrouver r à partir de la donnée publique (N, g) .
- (b) Montrer que si r est connu alors il est possible de factoriser N en $O(N^{1/4}/r)$ opérations dans le groupe $(\mathbb{Z}/N\mathbb{Z})^*$.

Indication

En notant $p = xr + 1$ et $q = yr + 1$ et $(N - 1)/r = ur + v$ avec $0 \leq v < r$, on pourra utiliser un algorithme de logarithme discret pour retrouver la « retenue » c définie par $x + y = v + cr$ et montrer que sa connaissance est suffisante pour retrouver p et q .

4. Contrefaçon universelle.

- (a) Montrer qu'il existe un algorithme polynomial qui prenant en entrée N et un entier m premier avec r , retourne un entier γ tel que $my \equiv 1 \pmod{r}$. En déduire une contrefaçon universelle sous une attaque à clé seule contre le schéma de signature de Boyd.

Nous supposerons désormais que :

- (ii') r est un diviseur *composé* de $p - 1$ de taille ℓ ;

et qu'il est difficile de calculer un multiple de r (et nous ne supposons plus que la condition (iv) est vérifiée).

- (b) Montrer que la connaissance de la signature de deux messages m_1 et m_2 premiers entre eux permet de calculer la signature du message produit $m = m_1m_2$ (et réciproquement). En déduire une contrefaçon universelle sous une attaque à deux messages choisis contre le schéma de signature de Boyd.

Nous supposerons désormais que :

- (v) La signature σ d'un message $m \in \{0, 1\}^*$ est la racine $H(m)$ -ième de g (*i.e.* $\sigma^{H(m)} = g$) où $H : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$ est une fonction de hachage cryptographique (dans la suite nous supposerons que H se comporte comme une fonction aléatoire).

5. Contrefaçon existentielle.

Montrer que la connaissance d'un ensemble de messages $\{m, m_1, \dots, m_t\}$ vérifiant :

- $H(m) = a_1 \cdot a_2 \cdots a_t$ où les a_i sont des entiers deux à deux premiers entre eux ;
- a_i divise $H(m_i)$ pour $i \in \{1, \dots, n\}$

est suffisante pour monter une contrefaçon existentielle sous une attaque à messages choisis. En déduire que le schéma n'est pas résistant aux contrefaçons existentielles lorsque ℓ est significativement plus petit que k .

Solution

1. Nous avons $g^r \equiv 1 \pmod{N}$ donc $g^r \equiv 1 \pmod{q}$ et l'ordre de g divise le nombre premier r et $q - 1$ (par le petit théorème de Fermat). Puisque r ne divise pas $q - 1$, g est d'ordre 1 modulo q , donc $g \equiv 1 \pmod{q}$ et le calcul de $\gcd(g - 1, N)$ révèle le facteur q .

2. Pour la génération des clés, il suffit de générer un nombre premier aléatoire r de ℓ bits en utilisant (par exemple) le test de primalité de Miller-Rabin. À partir de r , il est alors possible de générer deux nombres premiers p et q tels que r divise $p - 1$ et $q - 1$ en tirant α et β aléatoirement dans l'intervalle d'entiers $[(2^{k-1} - 1)/r, (2^k - 2)/r]$ jusqu'à ce que $p = \alpha r + 1$ et $q = \beta r + 1$ soient premiers. En posant $N = pq$, il reste à générer un élément g de $(\mathbb{Z}/N\mathbb{Z})^*$ d'ordre r . Pour cela, il suffit de construire un élément g_p de $(\mathbb{Z}/p\mathbb{Z})^*$ et un élément g_q de $(\mathbb{Z}/q\mathbb{Z})^*$ tous les deux d'ordre r et de construire g par le théorème chinois des restes tel que $g \equiv g_p \pmod{p}$ et $g \equiv g_q \pmod{q}$. Nous obtenons donc l'algorithme (8.1).

Algorithme 8.1 Génération de clés pour le protocole de signature de Boyd

ENTRÉE: $k, \ell \in \mathbb{N}$

SORTIE: (N, g, r) vérifiant les conditions (i)-(iv)

répéter

$$r \xleftarrow{u.a.} [2^{\ell-1}, 2^\ell - 1]$$

jusqu'à TESTPRIMALITÉ(r) = VRAI

▷ r est un nombre premier de ℓ bits

répéter

$$\alpha \xleftarrow{u.a.} [(2^{k-1} - 1)/r, (2^k - 2)/r]$$

$$p \leftarrow \alpha r + 1$$

jusqu'à TESTPRIMALITÉ(p) = VRAI

▷ p est un nombre premier de ℓ bits

répéter

$$\beta \xleftarrow{u.a.} [(2^{k-1} - 1)/r, (2^k - 2)/r]$$

$$q \leftarrow \beta r + 1$$

jusqu'à TESTPRIMALITÉ(q) = VRAI

▷ q est un nombre premier de ℓ bits

$$N \leftarrow pq$$

répéter

$$h_p \xleftarrow{u.a.} (\mathbb{Z}/p\mathbb{Z})^*$$

$$g_p \leftarrow h_p^\alpha$$

jusqu'à $g_p \neq 1 \pmod{p}$

▷ g_p est un élément d'ordre r dans $(\mathbb{Z}/p\mathbb{Z})^*$

répéter

$$h_q \xleftarrow{u.a.} \mathbb{Z}/q\mathbb{Z}^*$$

$$g_q \leftarrow h_q^\alpha$$

jusqu'à $g_q \neq 1 \pmod{q}$

▷ g_q est un élément d'ordre r dans $(\mathbb{Z}/q\mathbb{Z})^*$

$$g \leftarrow g_p \cdot q \cdot (q^{-1} \pmod{p}) + g_q \cdot p \cdot (p^{-1} \pmod{q}) \pmod{N}$$

retourner (N, g, r)

L'algorithme de signature est plus efficace que l'algorithme de signature RSA classique dès que $\ell < k/2$. Il faut en effet calculer l'inverse d de m modulo r puis calculer $g^d \pmod{N}$. Il consiste donc en deux exponentiations d'exposants de ℓ bits (au lieu de une avec un exposant de k bits).

3. (a) L'entier r est l'ordre de g ; il peut être vu comme le logarithme discret de 1 en base g dans le sous-groupe $\langle g \rangle \subseteq (\mathbb{Z}/p\mathbb{Z})^*$ (même si ce n'est pas la définition du logarithme discret). Un algorithme de logarithme discret générique comme l'algorithme de Shanks dans le sous-groupe engendré par g convient.

- (b) Avec les notations de l'indication, nous avons $(N - 1)/r = xy + (x + y) = ur + v$ où u et v sont connus mais x et y sont inconnus. Nous avons

$$x + y = v + cr, \quad xy = u - c$$

Nous avons $\lambda = \text{ppcm}(p - 1, q - 1) = \text{ppcm}(xr, yr)$ qui divise xyr . Pour un élément $a \in (\mathbb{Z}/N\mathbb{Z})^*$, nous avons donc

$$a^{ur} = a^{xyr+cr} = a^{cr}$$

et en posant $b = a^r \pmod{N}$, nous avons $b^u \equiv b^r \pmod{N}$. Puisque $cr \leq x + y$ et x et y sont de taille $k/2 - r$ bits, c est inférieur à \sqrt{N}/r^2 et un algorithme de logarithme discret générique retrouve la valeur de c en $O(N^{1/4}/r)$ opérations dans le groupe $(\mathbb{Z}/N\mathbb{Z})^*$.

4. (a) Supposons tout d'abord que m et $(N - 1)$ sont premiers entre eux. Notons γ l'inverse de m modulo $N - 1$. L'entier $N - 1$ divise $(m\gamma - 1)$ et r divise donc $m\gamma - 1$. La signature du message m est alors simplement obtenue comme

$$\sigma = g^\gamma \text{ puisque dans ce cas } \sigma^m = g^{\gamma \cdot m} = g$$

Si m et $N - 1$ ne sont pas premiers entre eux, il suffit de calculer γ comme l'inverse de m modulo $(N - 1)/\text{pgcd}(N - 1, m)$ et $\sigma = g^\gamma$ est encore une signature de m dans ce cas.

- (b) Notons σ_1 et σ_2 les signatures de m_1 et m_2 (respectivement). Nous avons

$$\sigma_1^{m_1} = g \text{ et } \sigma_2^{m_2} = g$$

Supposons que m_1 et m_2 sont premiers entre eux. Il existe deux entiers u et v tels que

$$um_1 + vm_2 = 1$$

En particulier, nous avons

$$g = g^{um_1 + vm_2} = g^{um_1}g^{vm_2} = \sigma_2^{um_1m_2}\sigma_1^{vm_1m_2} = (\sigma_2^u\sigma_1^v)^{m_1m_2}$$

et $\sigma_2^u\sigma_1^v$ est la signature de m_1m_2 . Réciproquement si σ est la signature de $m = m_1 \cdot m_2$, nous avons

$$g = \sigma^m = \sigma^{m_1m_2} = (\sigma^{m_1})^{m_2} = (\sigma^{m_2})^{m_1}$$

donc σ^{m_1} est la signature de m_2 et σ^{m_2} est la signature de m_1 .

5. Il suffit d'appliquer les résultats de la question précédente. La connaissance d'une signature σ_i de $H(m_i) = a_i b_i$ pour $i \in \{1, \dots, t\}$ permet d'obtenir une racine a_t -ième de g en calculant $\sigma_i^{b_i}$. Puisque les a_i sont deux à deux premiers entre eux, la question précédente permet de calculer de proche en proche la racine $(a_1 a_2)$ -ième de g , puis la racine $(a_1 a_2 a_3)$ -ième de g jusqu'à la racine $(a_1 \dots a_t)$ -ième de g , c'est-à-dire la signature de m .

Un adversaire peut donc produire une contrefaçon existentielle en appliquant un calcul d'indice (*cf.* Exercice (5.7)). En effet, en calculant des empreintes de messages aléatoires, il peut rechercher des messages m jusqu'à en obtenir un pour lequel $H(m)$ est friable. Il recherche simultanément des messages m_i tels que tous les nombres premiers de la base de facteurs divise l'une des empreintes $H(m_i)$. La connaissance d'un tel ensemble de messages permet de produire une signature sur m . La complexité de cette recherche est inférieure à celle d'un calcul d'indice dans $(\mathbb{Z}/r\mathbb{Z})$ et il est donc nécessaire que r soit sensiblement de la même taille que N pour se prémunir de cette attaque. Le protocole de signature de Boyd n'est dans ce cas pas plus efficace que le schéma RSA.

8.2 SIGNATURES D'ELGAMAL ET VARIANTES

Les protocoles de signature semblent nécessiter la propriété essentielle des fonctions à trappe : à savoir l'opposition entre la facilité pour tout utilisateur de vérifier la validité d'une signature et la difficulté pour tout utilisateur excepté le signataire légitime de produire cette signature. L'utilisation de la primitive RSA est donc un choix naturel pour construire des protocoles de signature numérique. En 1984, T. ELGAMAL [26] a proposé le premier exemple de signature dont la sécurité repose sur le problème du logarithme discret (pour lequel aucune trappe n'est connue). Initialement, il fut décrit sous la forme suivante que nous appelons protocole de *signature d'ElGamal naïf* :

Protocole de signature d'ElGamal naïf

Génération des clés : le signataire choisit un nombre premier p et g un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$. Il tire uniformément aléatoirement $x \in (\mathbb{Z}/(p-1)\mathbb{Z})$ et calcule $y = g^x \bmod p$. La clé publique est (p, g, y) et la clé secrète associée est x .

Signature : pour signer un message $m \in (\mathbb{Z}/(p-1)\mathbb{Z})$, le signataire tire uniformément aléatoirement $k \in (\mathbb{Z}/(p-1)\mathbb{Z})^*$ et calcule $r = g^k \bmod p$. Il calcule $s = (m - xr)/k \bmod p - 1$ et la signature est le couple (r, s) .

Vérification : un couple (r, s) est une signature valide de $m \in (\mathbb{Z}/(p-1)\mathbb{Z})$ si et seulement si $(r, s) \in (\mathbb{Z}/(p-1)\mathbb{Z})^2$ et

$$g^m = y^r r^s \bmod p$$

Exercice 8.9 Contrefaçon existentielle du schéma de signature d'ElGamal naïf

Montrer que le protocole de signature d'ElGamal naïf n'est pas résistant aux contrefaçons existentielles sous une attaque sans message.

Solution

Pour produire une contrefaçon existentielle pour une clé publique $y \in (\mathbb{Z}/p\mathbb{Z})^*$, l'attaquant doit fournir un message $m \in (\mathbb{Z}/(p-1)\mathbb{Z})$ et un couple $(r, s) \in (\mathbb{Z}/(p-1)\mathbb{Z})^2$ tels que

$$g^m = y^r r^s \bmod p$$

En posant $r = g^a y^b \bmod p$, cette relation donne

$$g^m = y^r g^{as} y^{bs}$$

En choisissant, par exemple, aléatoirement a dans $(\mathbb{Z}/(p-1)\mathbb{Z})$ et b dans $(\mathbb{Z}/(p-1)\mathbb{Z})^*$, en calculant $r = g^a y^b \bmod p$ et en fixant $s = -r/b \bmod p-1$, le couple (r, s) est une signature valide du message $m = as \bmod p-1$.

Exercice 8.10 Contrefaçon universelle du schéma de signature d'ElGamal naïf

Supposons que le générateur g de $(\mathbb{Z}/p\mathbb{Z})^*$ est égal à 2 et que $p \equiv 1 \pmod 4$.

- Montrer que le logarithme discret de $q = (p-1)/2$ en base $g = 2$ est égal à $(p-3)/2$.
- En déduire que, avec ce choix de générateur, le protocole de signature d'ElGamal naïf n'est pas résistant aux contrefaçons universelles sous une attaque sans message.

Solution

- Pour tout générateur $\alpha \in (\mathbb{Z}/p\mathbb{Z})^*$, nous avons avec $k = (p-3)/2$:

$$\alpha^k = \alpha^{(p-1)/2} \alpha^{-1} = (-1) \alpha^{-1} = (p-1)/\alpha \bmod p$$

donc le logarithme discret de $(p-1)/2$ en base 2 est égal à $(p-3)/2$ si 2 est générateur de $(\mathbb{Z}/p\mathbb{Z})^*$.

- Soit $z \in \{0, 1\}$ le logarithme discret de $y^{(p-1)/2} = \pm 1 \bmod p$ en base $2^{(p-1)/2} = -1 \bmod p$. Pour $r = (p-1)/2$ et $s \in (\mathbb{Z}/(p-1)\mathbb{Z})$, nous avons

$$y^r r^s = y^{(p-1)/2} ((p-1)/2)^s = y^{(p-1)/2} 2^{s(p-3)/2} = 2^{z(p-1)/2 + s(p-3)/2}$$

Soit $m \in (\mathbb{Z}/(p-1)\mathbb{Z})$, puisque $p \equiv 1 \pmod 4$, $p-1$ et $(p-3)/2$ sont premiers entre eux et en posant

$$s = (m - z(p-1)/2) \cdot [(p-3)/2]^{-1} \bmod p-1$$

le couple $(r = (p-1)/2, s)$ est une signature valide du message m .

Comme pour le protocole de signature RSA, un moyen simple pour renforcer la sécurité du protocole de signature d'ElGamal est d'appliquer une fonction de hachage au message à signer et d'appliquer le protocole précédent à l'empreinte obtenue.

Protocole de signature d'ElGamal

Génération des clés : le signataire choisit un nombre premier p , g un générateur de $(\mathbb{Z}/p\mathbb{Z})^*$ et une fonction de hachage \mathcal{H} . Il tire uniformément aléatoirement $x \in (\mathbb{Z}/(p-1)\mathbb{Z})$ et calcule $y = g^x \bmod p$. La clé publique est (p, g, y) et la clé secrète associée est x .

Signature : pour signer un message $m \in \{0, 1\}^*$, le signataire tire uniformément aléatoirement $k \in (\mathbb{Z}/(p-1)\mathbb{Z})^*$ et calcule $r = g^k \bmod p$. Il calcule $s = (\mathcal{H}(m) - xr)/k \bmod p-1$ et la signature est le couple (r, s) .

Vérification : un couple (r, s) est une signature valide de $m \in \{0, 1\}^*$ si et seulement si $(r, s) \in (\mathbb{Z}/(p-1)\mathbb{Z})^2$ et

$$g^{\mathcal{H}(m)} = y^r r^s \bmod p$$

L'exercice suivant montre que l'étape qui consiste à vérifier qu'une signature d'ElGamal (r, s) appartient bien à l'ensemble $(\mathbb{Z}/(p-1)\mathbb{Z})^2$ n'est pas à négliger. Si cette vérification n'est pas effectuée (par une implantation incorrecte par exemple), alors le schéma n'est pas résistant à la contrefaçon universelle.

Exercice 8.11 Vérification des signatures d'ElGamal

Montrer que si l'algorithme de vérification ne vérifie pas que le premier élément d'une signature d'ElGamal (r, s) d'un message appartient à l'ensemble $(\mathbb{Z}/(p-1)\mathbb{Z})$, alors le protocole de signature n'est pas résistant aux contrefaçons universelles sous une attaque sans message.

Solution

En suivant l'attaque de l'exercice 8.9, l'attaquant peut construire une valeur $m \in (\mathbb{Z}/(p-1)\mathbb{Z})$ inversible modulo $p-1$ et un couple $(r, s) \in (\mathbb{Z}/(p-1)\mathbb{Z})^2$ tels que

$$g^m = y^r r^s \bmod p$$

Il peut ensuite construire la signature d'un message m^* arbitraire, en posant

$$u = \mathcal{H}(m^*) m^{-1} \bmod p-1$$

vérifiant

$$g^{\mathcal{H}(m^*)} = g^{mu} = y^{ru} r^{su} \bmod p$$

En posant $s^* = su \bmod p - 1$ et en calculant à l'aide du théorème chinois des restes r^* vérifiant $r^* = ru \bmod p - 1$ et $r^* = r \bmod p$, le couple (r^*, s^*) satisfait bien l'équation de vérification de la signature d'ElGamal (mais r^* n'appartient pas à l'ensemble $(\mathbb{Z}/(p-1)\mathbb{Z})$).

Il existe plusieurs variantes du protocole de signature d'ElGamal. Dans ce livre, nous allons en étudier deux : le protocole de signature de Schnorr [58] et le *Digital Signature Algorithm*, plus connu sous le sigle **DSA**, qui est un algorithme de signature numérique standardisé par le NIST aux États-Unis.

Protocole de signature de Schnorr

Génération des clés : le signataire choisit un nombre premier p , q un diviseur premier de $p - 1$, g un élément de $(\mathbb{Z}/p\mathbb{Z})^*$ d'ordre q et une fonction de hachage \mathcal{H} . Il tire uniformément aléatoirement $x \in (\mathbb{Z}/q\mathbb{Z})$ et calcule $y = g^x \bmod p$. La clé publique est (p, q, g, y) et la clé secrète associée est x .

Signature : pour signer un message $m \in \{0, 1\}^*$, le signataire tire uniformément aléatoirement $k \in (\mathbb{Z}/q\mathbb{Z})^*$ et calcule $r = g^k \bmod p$. Il calcule $s = \mathcal{H}(m, r)$ puis $t = k - xs \bmod q$ et la signature est le couple (s, t) .

Vérification : un couple (s, t) est une signature valide de $m \in \{0, 1\}^*$ si et seulement si $(s, t) \in (\mathbb{Z}/q\mathbb{Z})^2$ et

$$\mathcal{H}(m, y^s g^t) = s \bmod p$$

Les deux différences essentielles entre le protocole de Schnorr et le protocole d'ElGamal sont :

- l'utilisation d'un sous-groupe d'ordre q de $(\mathbb{Z}/p\mathbb{Z})^*$ qui permet d'améliorer l'efficacité de la génération des signatures et la taille des signatures produites ;
- l'introduction de la valeur r dans la fonction de hachage qui permet de prouver la sécurité du schéma de signature dans un modèle où l'on suppose que la fonction \mathcal{H} se comporte comme une fonction aléatoire (alors qu'une telle preuve n'est toujours pas connue pour les signatures d'ElGamal).

Comme dans le cas des signatures **RSA**, un bon choix de la fonction de hachage utilisée est essentiel pour assurer la sécurité du schéma de Schnorr.

Exercice 8.12 Fonction de hachage et sécurité des signatures de Schnorr

Montrer que si la fonction de hachage utilisée dans le protocole de signature de Schnorr est définie par

$$\mathcal{H}(m, r) = m \cdot r \bmod p$$

alors le protocole de Schnorr n'est pas résistant aux contrefaçons existentielles.

Solution

Pour ce choix de fonction de hachage, un couple $(s, t) \in (\mathbb{Z}/q\mathbb{Z})^2$ est une signature sur un message $m \in (\mathbb{Z}/p\mathbb{Z})$ si et seulement si

$$m \cdot y^s g^t = s \pmod{p}$$

Il est relativement simple de construire un triplet vérifiant cette condition en choisissant la valeur de $(s, t) \in (\mathbb{Z}/q\mathbb{Z})^2$, puis en calculant $m = s/y^s g^t \pmod{p}$.

Le protocole de signature DSA est très proche du schéma d'ElGamal en reprenant l'idée de Schnorr de travailler dans un sous-groupe d'ordre q de $(\mathbb{Z}/p\mathbb{Z})^*$ et en y ajoutant une opération spécifique (la composition de la réduction modulo p puis la réduction modulo q) qui rend l'étude de la sécurité du schéma (et la cryptanalyse) difficile.

Protocole de signature DSA

Génération des clés : le signataire choisit un nombre premier p , q un diviseur premier de $p - 1$, g un élément de $(\mathbb{Z}/p\mathbb{Z})^*$ d'ordre q et une fonction de hachage \mathcal{H} . Il tire uniformément aléatoirement $x \in (\mathbb{Z}/q\mathbb{Z})$ et calcule $y = g^x \pmod{p}$. La clé publique est (p, q, g, y) et la clé secrète associée est x .

Signature : pour signer un message $m \in \{0, 1\}^*$, le signataire tire uniformément aléatoirement $k \in (\mathbb{Z}/q\mathbb{Z})$ et calcule $r = g^k \pmod{p} \pmod{q}$. Il calcule $s = (\mathcal{H}(m) + rx)/k \pmod{q}$ et la signature est le couple (s, t) .

Vérification : un couple (r, s) est une signature valide de $m \in \{0, 1\}^*$ si et seulement si $(r, s) \in (\mathbb{Z}/q\mathbb{Z})^2$ et

$$g^{\mathcal{H}(m)/s} y^{r/s} \pmod{p} \pmod{q} = r$$

Dans les protocoles de signature d'ElGamal, de Schnorr et DSA, les paramètres publics p , q et g ne comportent pas d'information secrète. Ces paramètres peuvent donc être générés par une autorité et ensuite utilisés par tous les utilisateurs d'un système (chacun devant uniquement créer sa clé secrète $x \in (\mathbb{Z}/q\mathbb{Z})$ et la clé publique associée $y = g^x \pmod{q}$). En 1996, S. VAUDENAY [66] a montré l'importance du choix des paramètres publics dans le protocole DSA.

Exercice 8.13 (avec programmation). Paramètres publics dans le protocole DSA

- Soit \mathcal{H} une fonction de hachage qui produit des hachés de ℓ bits. Montrer comment construire un nombre premier q de ℓ bits et deux messages m_1 et m_2 tels que $\mathcal{H}(m_1) \equiv \mathcal{H}(m_2) \pmod{q}$ lorsque les hachés $\mathcal{H}(m_1)$ et $\mathcal{H}(m_2)$ sont interprétés comme des entiers de ℓ bits.

2. En déduire un moyen pour une autorité de produire une contrefaçon sur le protocole DSA si elle choisit les paramètres communs de façon malhonnête.
3. Donner un exemple d'un triplet (m_1, m_2, p) tels que $\mathcal{H}(m_1) \equiv \mathcal{H}(m_2) \pmod{q}$ et q est un nombre premier de 160 bits lorsque \mathcal{H} est la fonction de hachage SHA-1.

Solution

1. Pour construire le nombre premier q et les messages m_1 et m_2 tels que demandés, il suffit de prendre des messages aléatoires et de tester jusqu'à ce que $|\mathcal{H}(m_1) - \mathcal{H}(m_2)|$ soit divisible par un nombre premier de ℓ bits. Par le théorème des nombres premiers, le nombre de tests nécessaires pour trouver ce nombre premier q est de l'ordre de ℓ .
2. En supposant qu'un utilisateur du protocole de signature d'ElGamal utilise ce nombre premier q et cette fonction de hachage, l'autorité malhonnête peut demander la signature du message m_1 et obtenir la signature du message m_2 puisque le procédé de vérification des signatures DSA ne dépend que de l'empreinte du message modulo q .
3. En faisant une recherche sur le message $m_1 = "Attaque sur DSA"$ et une famille de messages $m_2 = "Message test : ? ? ?"$ où les points d'interrogation sont remplacés par un entier entre 1 et 1000, nous obtenons plusieurs choix de m_2 tels que $|\mathcal{H}(m_1) - \mathcal{H}(m_2)|$ est un nombre premier. Plus précisément, les entiers entre 1 et 1000 pour lesquels l'attaque fonctionne (pour ce choix de messages) sont 186, 297, 615, 689 et 863.

En utilisant le message $m_2 = \text{Message test : } 186$, nous obtenons :

$$\text{SHA-1}(m_1) = \text{E0DC1F35AF258B358C5C051E335A5A9447616B2F$$

$$\text{SHA-1}(m_2) = 67E67FD7A3F3755BD3C98009E635559095ABF024$$

$$q = 1283722815356855959640339429484417700867290000175$$

et le test de primalité de Miller-Rabin (par exemple) montre que q est un nombre premier (de 160 bits).

L'exercice suivant montre que la sécurité des protocoles de signature d'ElGamal (comme celle des protocoles de Schnorr et DSA) repose intrinsèquement sur le caractère aléatoire de la *clé temporaire* k utilisée pour masquer la clé secrète dans la signature.

Exercice 8.14 Clé temporaire et sécurité des signatures d'ElGamal

1. Étudier la sécurité du protocole de signature d'ElGamal lorsque le signataire utilise toujours le même couple $[(r = g^k \pmod{p}), k]$ précalculé pour accélérer le calcul des signatures

2. Supposons que pour accélérer le calcul des signatures d'ElGamal, le signataire calcule deux couples $[(r = g^k \bmod p), k]$ et $[(a = g^\alpha \bmod p), \alpha]$ et utilise pour la i -ième signature la clé temporaire $[(r_i = g^{k+i\alpha} \bmod p), k + i\alpha]$ générée par une simple multiplication dans $(\mathbb{Z}/p\mathbb{Z})^*$. Étudier la sécurité du protocole de signature obtenu.

Solution

1. Dans ce cas, le schéma de signature d'ElGamal n'est pas résistant au bris total sous une attaque à deux messages connus. En effet, étant données deux signatures distinctes (r_1, s_1) et (r_2, s_2) sur deux messages m_1 et m_2 avec $r_1 = r_2 = g^k \bmod p$ et

$$s_1 = (\mathcal{H}(m_1) - xr_1)/k \bmod p - 1$$

$$s_2 = (\mathcal{H}(m_2) - xr_2)/k \bmod p - 1$$

À partir de ces deux égalités, nous obtenons

$$k = \frac{\mathcal{H}(m_1) - \mathcal{H}(m_2)}{s_1 - s_2} \bmod p - 1$$

puis

$$x = \frac{ks_1 - \mathcal{H}(m_1)}{r} \bmod p - 1$$

2. Dans ce cas, le schéma de signature d'ElGamal n'est pas résistant au bris total sous une attaque à trois messages connus. En effet, étant données trois signatures distinctes (r_1, s_1) , (r_2, s_2) et (r_3, s_3) sur trois messages m_1 , m_2 et m_3 avec $r_1 = g^{k+\alpha} \bmod p$, $r_2 = g^{k+2\alpha} \bmod p$ et $r_3 = g^{k+3\alpha} \bmod p$.

$$(k + \alpha)s_1 = \mathcal{H}(m_1) - xr_1 \bmod p - 1$$

$$(k + 2\alpha)s_2 = \mathcal{H}(m_2) - xr_2 = \mathcal{H}(m_2) - xr_1a \bmod p - 1$$

$$(k + 3\alpha)s_3 = \mathcal{H}(m_3) - xr_3 = \mathcal{H}(m_3) - xr_1a^2 \bmod p - 1$$

Nous avons $a = r_2/r_1 \bmod p$. En multipliant la première égalité par a et en lui sous-trayant la deuxième nous obtenons

$$(k + \alpha)s_1a - (k + 2\alpha)s_2 = \mathcal{H}(m_1)a - \mathcal{H}(m_2)$$

De même, avec la deuxième et la troisième ligne, nous obtenons

$$(k + 2\alpha)s_2a - (k + 3\alpha)s_3 = \mathcal{H}(m_2)a - \mathcal{H}(m_3)$$

Nous obtenons le système linéaire à deux inconnues

$$(as_1 - s_2)k + (as_1 - 2s_2)\alpha = \mathcal{H}(m_1)a - \mathcal{H}(m_2)$$

$$(as_2 - s_3)k + (2as_2 - 3s_3)\alpha = \mathcal{H}(m_2)a - \mathcal{H}(m_3)$$

qui nous donne les valeurs de k et de α en fonction des empreintes des messages et des signatures. À partir de ces valeurs, il est facile de retrouver x à partir d'une des trois équations initiales.

8.3 SIGNATURES DE LAMPORT ET VARIANTES

Une *signature de Lamport* (ou *signature jetable*) est une méthode pour construire un protocole de signature numérique dont la sécurité repose sur une fonction à sens unique $f : X \rightarrow Y$. Ces signatures ont été proposées par L. LAMPORT en 1979 [39] et, comme les signatures basées sur le logarithme discret, elles n'utilisent pas de trappe pour générer des signatures.

Protocole de signature de Lamport

Génération des clés : étant donnée une fonction à sens unique $f : X \rightarrow Y$ et un espace de message $\mathcal{M} = \{0, 1\}^k$, le signataire tire uniformément aléatoirement $2k$ valeurs

$$(x_1^{(0)}, x_1^{(1)}, x_2^{(0)}, x_2^{(1)}, \dots, x_k^{(0)}, x_k^{(1)}) \in X^{2k}$$

et calcule, pour $i \in \{1, \dots, k\}$ et $j \in \{0, 1\}$, $y_i^{(j)} = f(x_i^{(j)})$. La clé publique est le vecteur $(y_1^{(0)}, y_1^{(1)}, y_2^{(0)}, y_2^{(1)}, \dots, y_k^{(0)}, y_k^{(1)}) \in Y^{2k}$ et la clé secrète est le vecteur $(x_1^{(0)}, x_1^{(1)}, x_2^{(0)}, x_2^{(1)}, \dots, x_k^{(0)}, x_k^{(1)}) \in X^{2k}$.

Signature : pour signer un message $m = (m_1, \dots, m_k) \in \mathcal{M}$ où $m_i \in \{0, 1\}$ pour $i \in \{1, \dots, k\}$, le signataire révèle $\sigma = (x_1^{(m_1)}, \dots, x_k^{(m_k)}) \in X^k$.

Vérification : le k -uplet $\sigma = (\sigma_1, \dots, \sigma_k) \in X^k$ est une signature valide de $m = (m_1, \dots, m_k) \in \mathcal{M}$ où $m_i \in \{0, 1\}$ pour $i \in \{1, \dots, k\}$ pour la clé publique

$$(y_1^{(0)}, y_1^{(1)}, y_2^{(0)}, y_2^{(1)}, \dots, y_k^{(0)}, y_k^{(1)}) \in Y^{2k}$$

si et seulement si $f(\sigma_i) = y_i^{(m_i)}$ pour tout $i \in \{1, \dots, k\}$.

Exercice 8.15 Sécurité et efficacité des signatures de Lamport

- Évaluer la complexité de la génération des clés, de la génération des signatures, de la vérification des signatures du protocole de Lamport. Donner la taille des signatures produites et la taille des clés si l'espace de message \mathcal{M} est de taille 256 bits et la fonction à sens unique utilisée est une fonction de hachage qui retourne des hachés de 256 bits.
- Montrer que le protocole de signature de Lamport n'est pas résistant à la contre-façon universelle sous une attaque à deux messages choisis.

Solution

- Pour la génération des clés (*resp.* la vérification des signatures), il faut évaluer $2k$ fois (*resp.* k fois) la fonction à sens unique sous-jacente. La génération des signatures en revanche est immédiate et ne demande aucun calcul.

La taille des signatures produites est $256 \times 256 = 65536$ bits et la taille des clés est deux fois plus grande (131072 bits).

- Si un attaquant demande la signature du message $0^k = (0, 0, \dots, 0)$ et du message $1^k = (1, 1, \dots, 1)$, il obtient pour la première signature le vecteur

$$(x_1^{(0)}, \dots, x_k^{(0)}) \in X^k$$

et pour la seconde le vecteur

$$(x_1^{(1)}, \dots, x_k^{(1)}) \in X^k$$

Il possède donc l'intégralité de la clé secrète de signature et peut produire des signatures sur les messages de son choix.

En revanche, L. LAMPORT a montré que son protocole de signature est résistant aux contrefaçons sous une attaque à un message choisi si la fonction f est résistante aux pré-images. Les trois exercices suivants montrent qu'il est possible d'améliorer sensiblement l'efficacité du schéma de signature de Lamport.

Exercice 8.16 Espace de message de la signature de Lamport

- Montrer que le protocole de signature de Lamport ne peut pas être utilisé pour signer un message de longueur arbitraire $\ell \leq k$.
- Proposer une variante du protocole de signature de Lamport qui permet de signer un message de longueur arbitraire $\ell \leq k$ avec une clé publique de taille $O(k + \log k)$.

Solution

- Si le schéma de signature de Lamport est utilisé pour signer des messages de longueur arbitraire $\ell \leq k$, un attaquant peut facilement produire une contrefaçon existentielle sous une attaque à un message choisi. En effet, étant donnée la signature σ d'un message m , il peut produire immédiatement la signature de tout préfixe de m comme le préfixe correspondant de σ .
- Pour se prémunir de l'attaque de la question précédente, il faut que le signataire authentifie la longueur du message signé en plus du message lui-même. Pour un message de longueur $\ell \leq k$, la valeur ℓ peut être encodée en $\lceil \log_2 k \rceil$ bits et il suffit donc d'ajouter $2\lceil \log_2 k \rceil$ éléments dans la clé publique et de les utiliser pour signer la longueur du message. Les autres éléments de la clé publique sont utilisés pour signer le message lui-même (et les éléments non utilisés si $\ell < k$ ne doivent pas être utilisés pour un autre message au risque de perdre toute propriété de sécurité).

Exercice 8.17 Extension de l'espace des messages des signatures de Lamport *

Soit $k \in \mathbb{N}$, $k \geq 1$.

1. Considérons un sous-ensemble de $\{1, \dots, 2k\}$ de cardinal égal à k . En représentant ce sous-ensemble par le k -uplet (t_1, \dots, t_k) vérifiant $2k \geq t_1 > \dots > t_k \geq 1$, calculer son rang pour l'ordre lexicographique dans cette représentation.
2. En déduire un algorithme qui prenant en entrée un entier $n \in \{1, \dots, \binom{2k}{k}\}$, retourne un sous-ensemble de $\{1, \dots, 2k\}$ de cardinal k de sorte que pour deux entiers différents, l'algorithme retourne deux sous-ensembles différents.
3. Utiliser cet algorithme pour augmenter la longueur des messages qu'il est possible de signer avec le protocole de signature de Lamport sans augmenter la taille de la clé publique.

Solution

1. Les prédécesseurs d'un k -uplet (t_1, \dots, t_k) avec $2k \geq t_1 > \dots > t_k \geq 1$ pour l'ordre lexicographique sont les k -uplets (u_1, \dots, u_k) vérifiant l'une des conditions suivantes :

- $u_1 < t_1$
- $u_1 = t_1$ et $u_2 < t_2$
- ⋮
- $u_i = t_i$ pour $i \in \{1, \dots, k-1\}$ et $u_k < t_k$.

Le nombre de prédécesseurs du k -uplet (t_1, \dots, t_k) est donc égal à :

$$\binom{t_1 - 1}{k} + \binom{t_2 - 1}{k-1} + \dots + \binom{t_k - 1}{1} = \sum_{i=1}^k \binom{t_i - 1}{k-i+1} \quad (8.1)$$

et le rang du k -uplet (t_1, \dots, t_k) est égal à la somme (8.1) plus 1.

2. La représentation de la question précédente donne immédiatement un algorithme théorique en construisant l'ensemble formé des éléments du k -uplet (t_1, \dots, t_k) de rang n pour $n \in \{1, \dots, \binom{2k}{k}\}$. La difficulté vient de la conversion de l'entier n sous la forme (8.1). La conversion peut se faire efficacement en remarquant le caractère « supercroissant » de la suite des coefficients binomiaux :

$$\binom{t}{k} - \left(\binom{t-1}{k} + \binom{t-2}{k-1} + \dots + \binom{t-k-1}{1} \right) \geq 0$$

qui se montre facilement par récurrence :

$$\begin{aligned} & \binom{t}{k} - \left(\binom{t-1}{k} + \binom{t-2}{k-1} + \dots + \binom{t-k-1}{1} \right) \\ &= \binom{t}{k} - \binom{t-1}{k} - \left(\binom{t-2}{k-1} + \dots + \binom{t-k-1}{1} \right) \\ &= \binom{t-1}{k-1} - \left(\binom{t-2}{k-1} + \dots + \binom{t-k-1}{1} \right) \end{aligned}$$

L'élément t_k est donc nécessairement le plus grand entier tel que $\binom{t_k-1}{k} \leq n-1$ et la décomposition de l'entier $n-1$ sous la forme (8.1) peut donc se faire par un algorithme « glouton » (cf. Algorithme 8.2).

Algorithme 8.2 Construction de sous-ensemble de $\{1, \dots, 2k\}$

ENTRÉE: $n \in \{1, \dots, \binom{2k}{k}\}$
SORTIE: $S \subset \{1, \dots, 2k\}$, $\#S = k$

```

 $S \leftarrow \emptyset$ 
 $t \leftarrow 2k$ ;  $e \leftarrow k$ 
 $r \leftarrow n - 1$                                 ▷ Construction de l'ensemble avec  $r$  prédécesseurs
tant que  $t > 0$  faire
    si  $r \geq \binom{t-1}{e}$  alors
         $r \leftarrow r - \binom{t-1}{e}$ 
         $e \leftarrow e - 1$ 
         $S \leftarrow S \cup \{t\}$ 
    fin si
     $t \leftarrow t - 1$ 
fin tant que
retourner  $S$ 

```

3. La clé publique du protocole de Lamport contient $2k$ éléments dans l'image d'une fonction à sens unique. Pour signer un message $m \in \{1, \dots, \binom{2k}{k}\}$, le signataire peut révéler les k pré-images correspondants au sous-ensemble M de $\{1, \dots, 2k\}$ de cardinal k associé à m . La sécurité du schéma de signature est conservée puisque pour produire une contrefaçon sur un message $m' \neq m$ de l'ensemble $\{1, \dots, \binom{2k}{k}\}$, un attaquant doit inverser la fonction à sens unique sur une valeur de $\{1, \dots, 2k\}$ en dehors de M .
- Par la formule de Stirling $\binom{2k}{k} \simeq 2^{2k}/\sqrt{\pi k}$ et cette approche permet donc de signer des messages environ 2 fois plus longs sans augmenter la taille de la clé publique.

Note

Cette variante du schéma de signature de Lamport est due à J. N. E. Bos et D. CHAUM [11].

L'un des inconvénients majeurs du schéma de signature de Lamport est la taille de sa clé publique. Si un signataire veut pouvoir signer plusieurs messages, il doit disposer de nombreux éléments de X dans sa clé publique pour pouvoir les authentifier. Les *arbres de Merkle* ont été proposés en 1979 par R. MERKLE [46] pour permettre de gérer efficacement ces clés publiques.

Exercice 8.18 Arbres de Merkle

Soit \mathcal{H} une fonction de hachage supposée résistante à la seconde pré-image.

1. Proposer une méthode pour qu'un utilisateur du schéma de signature de Lamport ne révèle qu'une clé publique courte (indépendamment de la taille des messages à signer) sans compromettre la sécurité mais en augmentant la taille de sa signature.
2. En utilisant une structure d'arbre binaire, étendre cette technique pour construire un schéma de signatures qui permet de signer n messages de longueur k avec une clé publique de taille constante et des signatures de taille $O(k + \log(n))$.

Solution

1. Par définition, la clé publique dans le schéma de signature de Lamport est un vecteur $\Upsilon = (y_1^{(0)}, y_1^{(1)}, y_2^{(0)}, y_2^{(1)}, \dots, y_k^{(0)}, y_k^{(1)}) \in Y^{2k}$. Pour compresser ce vecteur, il suffit de remplacer la clé publique par une empreinte par la fonction de hachage $c = \mathcal{H}(\Upsilon)$ de ce vecteur.

En notant $X = (x_1^{(0)}, x_1^{(1)}, x_2^{(0)}, x_2^{(1)}, \dots, x_k^{(0)}, x_k^{(1)}) \in X^{2k}$, la clé secrète associée, la signature d'un message $m = (m_1, \dots, m_k) \in \mathcal{M}$ (où $m_i \in \{0, 1\}$ pour $i \in \{1, \dots, k\}$) dans le schéma de Lamport est le vecteur $\sigma = (x_1^{(m_1)}, \dots, x_k^{(m_k)}) \in X^k$.

Si la clé publique est remplacée par l'empreinte $\mathcal{H}(\Upsilon)$, un utilisateur ne peut plus vérifier la validité de σ . Il faut donc augmenter la taille de la signature en y ajoutant les éléments $y_i^{(1-m_i)}$ pour $i \in \{1, \dots, k\}$. Avec

$$\overline{\sigma} = (x_1^{(m_1)}, \dots, x_k^{(m_k)}, y_1^{(1-m_1)}, \dots, y_k^{(1-m_k)}) \in X^k \times Y^k$$

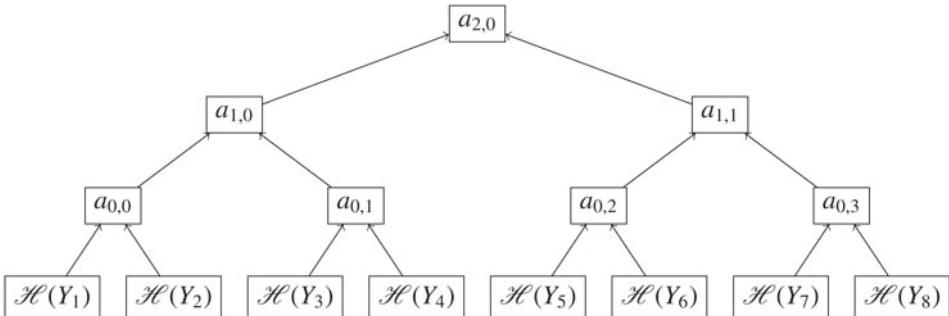
un utilisateur peut vérifier $\overline{\sigma}$ en s'assurant que l'égalité

$$c = \mathcal{H}(z_1^{(0)}, z_1^{(1)}, z_2^{(0)}, z_2^{(1)}, \dots, z_k^{(0)}, z_k^{(1)})$$

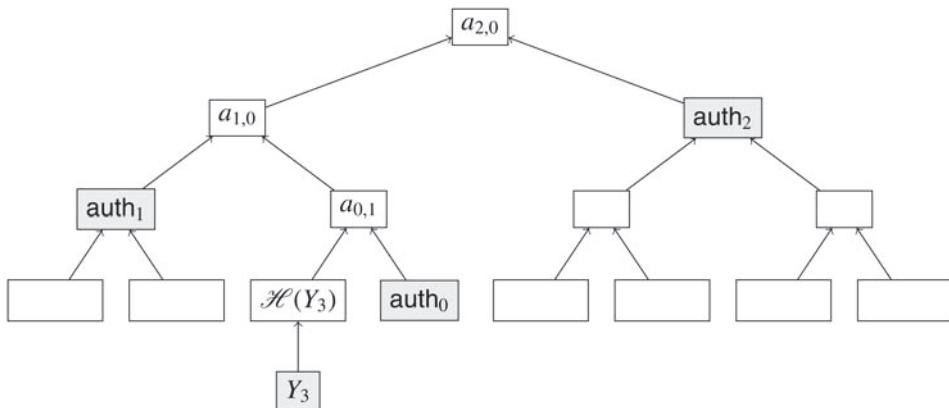
avec $z_i^{(m_i)} = f(\sigma_i)$ et $z_i^{(1-m_i)} = \sigma_{k+i}$, est bien vérifiée.

La sécurité du nouveau protocole de signature n'est pas modifiée si \mathcal{H} est résistante à la seconde pré-image car un attaquant potentiel devra soit produire une contrefaçon contre le schéma de Lamport original, soit produire un autre vecteur Υ' de Y^k tel que $\mathcal{H}(\Upsilon')$.

2. L'idée naturelle est de placer les clés publiques créées pour chacun des messages à signer aux feuilles d'un arbre binaire et d'utiliser de nouveau la fonction de hachage \mathcal{H} pour produire une clé publique réduite à un seul élément. La valeur à chaque nœud qui n'est pas une feuille dans l'arbre est obtenue comme étant l'empreinte de ses deux fils et le processus est répété jusqu'à n'obtenir qu'un nœud : la clé publique (cf. Figure 8.3).


 Figure 8.3 - Construction de la clé publique ($n = 8$)

En utilisant cet arbre, une signature doit maintenant contenir les valeurs permettant de reconstruire la clé publique. Pour signer un message, le signataire applique le protocole de signature de Lamport avec une clé encore non utilisée et révèle l'empreinte de la clé publique associée (comme dans la question 1). Il publie ensuite les valeurs de la fratrie de chaque ancêtre de cette feuille (notées auth_i pour $i \in \{0, \dots, \lfloor \log_2(n) \rfloor s - 1\}$ sur la figure 8.4). Un utilisateur peut ensuite vérifier une signature en recalculant les hachés de l'arbre jusqu'à obtenir la valeur de la clé publique à la racine de l'arbre.


 Figure 8.4 - Génération et vérification d'une signature ($n = 8$)

Note

Puisqu'ils sont intrinsèquement parallélisables, les arbres de Merkle sont également utilisés pour construire des fonctions de hachage cryptographiques itérées (comme alternative à la construction de Merkle-Damgård). Ils sont notamment utilisés dans la fonction de hachage MD6 qui a été proposée par R. RIVEST pour participer à la compétition de fonction de hachage du NIST.

Pour certaines fonctions à sens unique, il est encore possible d'améliorer l'efficacité de ce schéma en utilisant des propriétés supplémentaires (par exemple algébriques). Soit \mathbb{G} un groupe d'ordre premier q engendré par g où le problème du logarithme discret est supposé difficile. Nous considérons le protocole de signature suivant qui a été proposé par J. GROTH en 2006 [30].

Protocole de signature de Groth

Génération des clés : le signataire tire uniformément aléatoirement trois entiers x_1, x_2 et x_3 dans $\{1, \dots, q - 1\}$ et calcule $y_1 = g^{x_1}$, $y_2 = g^{x_2}$ et $y_3 = g^{x_3}$. La clé publique est $(y_1, y_2, y_3) \in \mathbb{G}^3$ et la clé secrète est le triplet $(x_1, x_2, x_3) \in (\mathbb{Z}/q\mathbb{Z})^3$.

Signature : pour signer un message $m \in (\mathbb{Z}/q\mathbb{Z})$, le signataire tire uniformément r_1 dans $(\mathbb{Z}/q\mathbb{Z})$ et calcule $r_2 = (x_2 - m - x_1 r_1)/x_3 \bmod q$. La signature est le couple $(r_1, r_2) \in (\mathbb{Z}/q\mathbb{Z})^2$.

Vérification : étant donné un message $m \in (\mathbb{Z}/q\mathbb{Z})$ et une signature supposée (r_1, r_2) , l'algorithme accepte la signature si et seulement si

$$g^m \cdot y_1^{r_1} \cdot y_2^{r_2} = y_3 \quad (8.2)$$

La signature d'un élément de $(\mathbb{Z}/q\mathbb{Z})$ consiste en seulement deux éléments de $(\mathbb{Z}/q\mathbb{Z})$ (alors que le protocole de Lamport demanderait d'en révéler $\log(q)$). Nous allons montrer que ce schéma de signature est résistant aux contrefaçons existentielles sous une attaque à un message choisi si le problème du logarithme discret est difficile à résoudre dans \mathbb{G} .

Problème 8.19 Sécurité du protocole de signature de Groth

- Montrer que ce protocole de signature numérique n'est pas résistant aux contrefaçons universelles sous une attaque à deux messages choisis.

Nous considérons un algorithme (probabiliste) \mathcal{A} qui retourne une contrefaçon existentielle sous une attaque à un message choisi avec une probabilité ε en temps τ . Nous allons construire explicitement un algorithme probabiliste \mathcal{B} qui résout le problème du logarithme discret dans \mathbb{G} avec une probabilité $\varepsilon' \geq \varepsilon/2$ en temps $\tau' \simeq \tau$. L'algorithme \mathcal{B} prend en entrée un élément $h \in \mathbb{G}$ et doit retourner l'entier $x \in (\mathbb{Z}/q\mathbb{Z})$ tel que $h = g^x$.

- Notons m le message pour lequel \mathcal{A} demande une signature et (r_1, r_2) la signature qu'il obtient. Montrer que si l'algorithme \mathcal{A} produit une contrefaçon existentielle (r_1^*, r_2^*) sur un message $m^* \neq m$, alors $(r_1^*, r_2^*) \neq (r_1, r_2)$.
- Dans cette question, nous considérons uniquement des adversaires \mathcal{A} qui retournent une contrefaçon telle que $r_2^* \neq r_2$.

- (a) L'algorithme \mathcal{B}_1 va exécuter l'algorithme \mathcal{A} sur la clé publique $(y_1, y_2, y_3) \in \mathbb{G}^3$ formée de la façon suivante :

$$y_1 = g^{a_s}, y_2 = h \text{ et } y_3 = g^{b_s} h^{c_s}$$

où a_s, b_s, c_s sont tirés uniformément aléatoirement dans $\{1, \dots, q - 1\}$. Montrer que la distribution de (y_1, y_2, y_3) est identique à celle produite par l'algorithme de génération de clés du protocole et que l'algorithme \mathcal{A} n'a aucune information sur la valeur de c_s .

- (b) Montrer que lorsque l'algorithme \mathcal{A} demande une signature sur le message m , l'algorithme \mathcal{B}_1 peut retourner une signature distribuée de façon identique à celles produites par l'algorithme de signature.
- (c) Montrer que si l'algorithme \mathcal{A} produit une contrefaçon existentielle valide sur un message $m^* \neq m$, l'algorithme \mathcal{B} peut en déduire le logarithme discret de h en base g avec probabilité 1.
4. Construire de même un algorithme \mathcal{B}_2 qui résout le problème du logarithme discret à partir d'un algorithme \mathcal{A} qui retourne une contrefaçon telle que $r_1^* \neq r_1$.
5. Conclure.

Solution

1. Étant données deux signatures (r_1, r_2) et (s_1, s_2) sur deux messages m_1 et m_2 , nous avons

$$g^{m_1} \cdot y_1^{r_1} \cdot y_2^{r_2} = y_3 = g^{m_2} \cdot y_1^{s_1} \cdot y_2^{s_2}$$

Pour tout $\lambda \in \{1, \dots, q - 1\}$, nous avons alors

$$(g^{m_1} \cdot y_1^{r_1} \cdot y_2^{r_2})^\lambda (g^{m_2} \cdot y_1^{s_1} \cdot y_2^{s_2})^{1-\lambda} = y_3$$

et

$$g^{\lambda m_1 + (1-\lambda)m_2} \cdot y_1^{\lambda r_1 + (1-\lambda)s_1} \cdot y_2^{\lambda r_2 + (1-\lambda)s_2} = y_3$$

En particulier le couple $(\lambda r_1 + (1 - \lambda)s_1, \lambda r_2 + (1 - \lambda)s_2) \in \{1, \dots, q - 1\}^2$ est une signature du message $\lambda m_1 + (1 - \lambda)m_2$ pour tout $\lambda \in \{1, \dots, q - 1\}$.

2. Une signature (r_1, r_2) n'est valide que pour un seul message. En effet, l'équation de vérification est

$$g^m \cdot y_1^{r_1} \cdot y_2^{r_2} = y_3$$

et l'unique message pour lequel elle est vérifiée est donné par $m = \log_g(y_3 \cdot y_1^{-r_1} \cdot y_2^{-r_2})$. En particulier si l'algorithme \mathcal{A} produit une contrefaçon existentielle (r_1^*, r_2^*) sur un message $m^* \neq m$, alors $(r_1^*, r_2^*) \neq (r_1, r_2)$.

3. (a) La clé publique $(y_1, y_2, y_3) \in \mathbb{G}^3$ est identique à celle produite par l'algorithme de génération de clés avec $x_1 = a_s$, $x_2 = x$ et $x_3 = b_s + x \cdot c_s \bmod q$. De plus, l'élément y_3 ne révèle aucune information sur la valeur c_s choisie puisque pour toute valeur $c'_s \in \{0, \dots, q-1\}$, il existe un entier b'_s tel que $b'_s + x \cdot c'_s = b_s + x \cdot c_s \bmod q$.
- (b) Lorsque l'algorithme \mathcal{A} demande la signature sur un message $m \in \{1, \dots, q-1\}$, l'algorithme \mathcal{B}_1 doit retourner un couple (r_1, r_2) vérifiant l'égalité (8.2). Avec le choix de la clé publique, nous avons

$$g^m \cdot y_1^{r_1} \cdot y_2^{r_2} = y_3 \iff g^m g^{a_s r_1} h^{r_2} = g^{b_s} h^{c_s}$$

En posant $r_2 = c_s$ et $r_1 = (b_s - m)/a_s$, nous obtenons une signature valide sur m distribuée de façon identique à celles produites par l'algorithme de signature (puisque c_s est parfaitement caché de \mathcal{A}).

- (c) Si \mathcal{A} produit une contrefaçon existentielle valide (r_1^*, r_2^*) sur un message $m^* \neq m$, nous avons

$$g^{m^*} \cdot y_1^{r_1^*} \cdot y_2^{r_2^*} = y_3, \text{ soit } g^{m^*} g^{a_s r_1^*} h^{r_2^*} = g^{b_s} h^{c_s}$$

Par hypothèse sur \mathcal{A} , $r_2^* \neq r_2 = c_s$, et nous avons $h = g^{(m^* + a_s r_1^* - b_s)/(c_s - r_2^*)}$. L'algorithme \mathcal{B}_1 peut donc retourner $x = (m^* + a_s r_1^* - b_s)/(c_s - r_2^*) \bmod q$ le logarithme discret de h en base g .

4. L'algorithme \mathcal{B}_2 va simplement exécuter l'algorithme \mathcal{A} sur la clé publique $(y_1, y_2, y_3) \in \mathbb{G}^3$ formée de la façon suivante :

$$y_1 = h, y_2 = g^{a_s} \text{ et } y_3 = g^{b_s} h^{c_s}$$

où a_s, b_s, c_s sont tirés uniformément aléatoirement dans $\{1, \dots, q-1\}$. Comme dans la question précédente, la distribution de (y_1, y_2, y_3) est identique à celle produite par l'algorithme de génération de clés et ne révèle aucune information sur la valeur de c_s . Lorsque l'algorithme \mathcal{A} demande la signature d'un message $m \in \{1, \dots, q-1\}$, l'algorithme \mathcal{B}_2 retourne le couple (r_1, r_2) avec $r_1 = c_s$ et $r_2 = (b_s - m)/a_s$. Comme précédemment, il s'agit d'une signature valide pour m distribuée de façon identique à celles produites par l'algorithme de signature.

Si \mathcal{A} produit une contrefaçon existentielle valide (r_1^*, r_2^*) sur un message $m^* \neq m$, nous avons

$$g^{m^*} \cdot y_1^{r_1^*} \cdot y_2^{r_2^*} = y_3, \text{ soit } g^{m^*} h^{r_1^*} g^{a_s r_2^*} = g^{b_s} h^{c_s}$$

Par hypothèse sur \mathcal{A} , $r_1^* \neq r_1 = c_s$ et \mathcal{B}_2 peut retourner $x = (m^* + a_s r_2^* - b_s)/(c_s - r_1^*) \bmod q$, le logarithme discret de h en base g .

5. D'après les questions précédentes, nous savons comment construire à partir de \mathcal{A} un algorithme qui résout le problème du logarithme discret suivant le type de contrefaçon que \mathcal{A} va produire (soit $r_1^* \neq r_1$, soit $r_2^* \neq r_1$). Ce comportement est cependant imprévisible et il semble difficile de construire un algorithme qui fonctionne tout le temps. L'idée est donc de faire deviner par \mathcal{B} le type d'adversaire avec qui il va interagir. L'algorithme \mathcal{B} va donc tirer uniformément aléatoirement un bit b (indépendamment de \mathcal{A}) et si $b = 0$, il va exécuter l'algorithme \mathcal{B}_1 de la question 3 et si $b = 1$, il va

exécuter l’algorithme \mathcal{B}_2 de la question 4. Si l’adversaire \mathcal{A} est du type prédit par \mathcal{B} , celui-ci pourra obtenir le logarithme discret de h si \mathcal{A} produit une contrefaçon valide. La probabilité que \mathcal{B} prédise correctement le comportement de \mathcal{A} est supérieure à $1/2$ (elle ne dépend pas de \mathcal{A} le choix du bit b étant indépendant de \mathcal{A}). La probabilité de succès de \mathcal{B} est donc supérieure à la probabilité de \mathcal{A} divisée par 2. Le temps d’exécution de \mathcal{B} est proche de celui de \mathcal{A} (le surcoût consiste essentiellement en trois exponentiations dans le groupe \mathbb{G} et quelques opérations modulaires dans $(\mathbb{Z}/q\mathbb{Z})$).

BIBLIOGRAPHIE

- [1] L. M. ADLEMAN – « A subexponential algorithm for the discrete logarithm problem with applications to cryptography (abstract) », *20th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1979, p. 55–60.
- [2] M. AGRAWAL, N. KAYAL et N. SAXENA – « PRIMES is in P », *Ann. Math.* **160** (2004), no. 2, p. 781–793.
- [3] W. R. ALFORD, A. GRANVILLE et C. POMERANCE – « There are infinitely many Carmichael numbers », *Ann. Math.* **139** (1994), no. 3, p. 703–722.
- [4] R. M. AVANZI – « The complexity of certain multi-exponentiation techniques in cryptography », *J. Cryptology* **18** (2005), no. 4, p. 357–373.
- [5] T. BETH et F. PIPER – « The Stop-and-Go Generator », *Advances in Cryptology – EUROCRYPT’84* (T. Beth, N. Cot et I. Ingemarsson, éds.), Lect. Notes Comput. Sci., vol. 209, Springer, avril 1985, p. 88–92.
- [6] E. BIHAM – « Cryptanalysis of ladder-des », *Fast Software Encryption – FSE’97* (E. Biham, éd.), Lect. Notes Comput. Sci., vol. 1267, Springer, janvier 1997, p. 134–138.
- [7] E. BIHAM, A. Biryukov et A. SHAMIR – « Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials », *Advances in Cryptology – EUROCRYPT’99* (J. Stern, éd.), Lect. Notes Comput. Sci., vol. 1592, Springer, mai 1999, p. 12–23.
- [8] E. BIHAM et A. SHAMIR – « Differential Cryptoanalysis of Feal and N-Hash », *Advances in Cryptology – EUROCRYPT’91* (D. W. Davies, éd.), Lect. Notes Comput. Sci., vol. 547, Springer, avril 1991, p. 1–16.
- [9] J. BLACK, P. ROGAWAY, T. SHRIMPTON et M. STAM – « An Analysis of the Blockcipher-Based Hash Functions from PGV », *J. Cryptology* **23** (2010), no. 4, p. 519–545.
- [10] D. BONEH et G. DURFEE – « Cryptanalysis of rsa with private key d less than $n^{0.292}$ », *IEEE Trans. Inf. Theory* **46** (2000), no. 4, p. 1339.
- [11] J. N. Bos et D. CHAUM – « Provably Unforgeable Signatures », *Advances in Cryptology – CRYPTO’92* (E. F. Brickell, éd.), Lect. Notes Comput. Sci., vol. 740, Springer, août 1993, p. 1–14.
- [12] C. BOYD – « Digital signature and public key cryptosystems in a prime order subgroup of z_n^* », *ICICS 97 : 1st International Conference on Information and Communication Security* (Y. Han, T. Okamoto et S. Qing, éds.), Lect. Notes Comput. Sci., vol. 1334, Springer, novembre 1997, p. 346–355.
- [13] D. COPPERSMITH, H. KRAWCZYK et Y. MANSOUR – « The Shrinking Generator », *Advances in Cryptology – CRYPTO’93* (D. R. Stinson, éd.), Lect. Notes Comput. Sci., vol. 773, Springer, août 1994, p. 22–39.
- [14] D. COPPERSMITH et I. SHPARLINSKI – « On polynomial approximation of the discrete logarithm and the diffie - hellman mapping », *J. Cryptology* **13** (2000), no. 3, p. 339–360.
- [15] J.-S. CORON et A. MAY – « Deterministic polynomial-time equivalence of computing the RSA secret key and factoring », *J. Cryptology* **20** (2007), no. 1, p. 39–50.

Exercices et problèmes de cryptographie

- [16] J. DAEMEN, L. R. KNUDSEN et V. RIJMDEN – « The Block Cipher Square », *Fast Software Encryption – FSE'97* (E. Biham, éd.), Lect. Notes Comput. Sci., vol. 1267, Springer, janvier 1997, p. 149–165.
- [17] I. DAMGÅRD – « A Design Principle for Hash Functions », *Advances in Cryptology – CRYPTO'89* (G. Brassard, éd.), Lect. Notes Comput. Sci., vol. 435, Springer, août 1990, p. 416–427.
- [18] W. DE JONGE et D. CHAUM – « Attacks on Some RSA Signatures », *Advances in Cryptology – CRYPTO'85* (H. C. Williams, éd.), Lect. Notes Comput. Sci., vol. 218, Springer, août 1986, p. 18–27.
- [19] —, « Some Variations on RSA Signatures and Their Security », *Advances in Cryptology – CRYPTO'86* (A. M. Odlyzko, éd.), Lect. Notes Comput. Sci., vol. 263, Springer, août 1987, p. 49–59.
- [20] B. DEBRAIZE et L. GOUBIN – « Guess-and-Determine Algebraic Attack on the Self-Shrinking Generator », *Fast Software Encryption – FSE 2008* (K. Nyberg, éd.), Lect. Notes Comput. Sci., vol. 5086, Springer, février 2008, p. 235–252.
- [21] J. D. DIXON – « Asymptotically fast factorization of integers », *Math. Comput.* **36** (1981), p. 255–260.
- [22] S. Even et Y. MANSOUR – « A construction of a cipher from a single pseudorandom permutation », *Advances in Cryptology – ASIACRYPT'91* (H. Imai, R. L. Rivest et T. Matsumoto, éds.), Lect. Notes Comput. Sci., vol. 739, Springer, novembre 1991, p. 210–224.
- [23] R. W. FLOYD – « Nondeterministic algorithms », *J. ACM* **14** (1967), no. 4, p. 636–644.
- [24] S. R. FLUHRER, I. MANTIN et A. SHAMIR – « Weaknesses in the Key Scheduling Algorithm of RC4 », *SAC 2001 : 8th Annual International Workshop on Selected Areas in Cryptography* (S. Vaudenay et A. M. Youssef, éds.), Lect. Notes Comput. Sci., vol. 2259, Springer, août 2001, p. 1–24.
- [25] K. FORD – « The distribution of integers with a division in a given interval », *Annals of Math.* **168** (2008), p. 367–433.
- [26] T. E. GAMAL – « A public key cryptosystem and a signature scheme based on discrete logarithms », *IEEE Trans. Inf. Theory* **31** (1985), no. 4, p. 469–472.
- [27] P. R. GEFFE – « How to protect data with ciphers that are really hard to break », *Electronics* (1973), p. 99–101.
- [28] J. D. GOLIC – « Linear statistical weakness of alleged rc4 keystream generator », *Advances in Cryptology – EUROCRYPT'97* (W. Fumy, éd.), Lect. Notes Comput. Sci., vol. 1233, Springer, mai 1997, p. 226–238.
- [29] A. GRANVILLE – « Smooth numbers : computational number theory and beyond. », *Algorithmic number theory. Lattices, number fields, curves and cryptography* (Buhler, J. P. et al., éd.), Mathematical Sciences Research Institute Publications, vol. 44, Cambridge University Press, 2008, p. 267–323.
- [30] J. GROTH – « Simulation-sound NIZK proofs for a practical language and constant size group signatures », *Advances in Cryptology – ASIACRYPT 2006* (X. Lai et K. Chen, éds.), Lect. Notes Comput. Sci., vol. 4284, Springer, décembre 2006, p. 444–459.
- [31] J. HÅSTAD – « Solving simultaneous modular equations of low degree », *SIAM J. Comput.* **17** (1988), no. 2, p. 336–341.

- [32] A. JOUX – « Multicollisions in iterated hash functions. application to cascaded constructions », *Advances in Cryptology – CRYPTO 2004* (M. Franklin, éd.), Lect. Notes Comput. Sci., vol. 3152, Springer, août 2004, p. 306–316.
- [33] A. L. W. JR. – « A polynomial form for logarithms modulo a prime », *IEEE Trans. Inf. Theory* **30** (1984), no. 6, p. 845–.
- [34] J. KELSEY et B. SCHNEIER – « Second preimages on n-bit hash functions for much less than $2n$ work », *Advances in Cryptology – EUROCRYPT 2005* (R. Cramer, éd.), Lect. Notes Comput. Sci., vol. 3494, Springer, mai 2005, p. 474–490.
- [35] J. KILIAN et P. ROGAWAY – « How to Protect DES Against Exhaustive Key Search », *Advances in Cryptology – CRYPTO’96* (N. Koblitz, éd.), Lect. Notes Comput. Sci., vol. 1109, Springer, août 1996, p. 252–267.
- [36] T. KLEINJUNG, K. AOKI, J. FRANKE, A. K. LENSTRA, E. THOMÉ, J. W. BOS, P. GAUDRY, A. KRUPPA, P. L. MONTGOMERY, D. A. OSVIK, H. J. J. TE RIELE, A. TIMOFEEV et P. ZIMMERMANN – « Factorization of a 768-Bit RSA Modulus », *Advances in Cryptology – CRYPTO 2010*, Lect. Notes Comput. Sci., Springer, août 2010, p. 333–350.
- [37] L. KNUDSEN – « DEAL - a 128-bit block cipher », Tech. Report 151, University of Bergen, Norway, Department of Informatics, février 1998.
- [38] L. R. KNUDSEN – « Truncated and Higher Order Differentials », *Fast Software Encryption – FSE’94* (B. Preneel, éd.), Lect. Notes Comput. Sci., vol. 1008, Springer, décembre 1994, p. 196–211.
- [39] L. LAMPORT – « Constructing digital signatures from a one-way function », Tech. Report SRI-CSL-98, SRI International Computer Science Laboratory, octobre 1979.
- [40] R. LEHMAN – « Factoring Large Integers », *Math. Comput.* **28** (1974), p. 637–646.
- [41] I. MANTIN et A. SHAMIR – « A Practical Attack on Broadcast RC4 », *Fast Software Encryption – FSE 2001* (M. Matsui, éd.), Lect. Notes Comput. Sci., vol. 2355, Springer, avril 2001, p. 152–164.
- [42] J. L. MASSEY – « Shift-register synthesis and bch decoding », *IEEE Trans. Inf. Theory* **15** (1969), no. 1, p. 122–127.
- [43] J. L. MASSEY – « SAFER K-64 : A byte-oriented block-ciphering algorithm », *Fast Software Encryption – FSE’93* (R. J. Anderson, éd.), Lect. Notes Comput. Sci., vol. 809, Springer, décembre 1993, p. 1–17.
- [44] M. MATSUI – « Linear Cryptoanalysis Method for DES Cipher », *Advances in Cryptology – EUROCRYPT’93* (T. Helleseth, éd.), Lect. Notes Comput. Sci., vol. 765, Springer, mai 1993, p. 386–397.
- [45] A. MAY – « Computing the RSA secret key is deterministic polynomial time equivalent to factoring », *Advances in Cryptology – CRYPTO 2004* (M. Franklin, éd.), Lect. Notes Comput. Sci., vol. 3152, Springer, août 2004, p. 213–219.
- [46] R. C. MERKLE – « A digital signature based on a conventional encryption function », *Advances in Cryptology – CRYPTO’87* (C. Pomerance, éd.), Lect. Notes Comput. Sci., vol. 293, Springer, août 1988, p. 369–378.
- [47] — , « A Certified Digital Signature », *Advances in Cryptology – CRYPTO’89* (G. Brassard, éd.), Lect. Notes Comput. Sci., vol. 435, Springer, août 1990, p. 218–238.
- [48] G. L. MILLER – « Riemann’s hypothesis and tests for primality », *J. Comput. Syst. Sci.* **13** (1976), no. 3, p. 300–317.

Exercices et problèmes de cryptographie

- [49] P. Q. NGUYEN et B. VALLÉE (éds.) – *The lll algorithm. survey and applications.*, Information Security and Cryptography, Springer, 2010.
- [50] H. NIEDERREITER – « A short proof for explicit formulas for discrete logarithms in finite fields », *Appl. Algebra Eng. Commun. Comput.* **1** (1990), p. 55–57.
- [51] S. C. POHLLIG et M. E. HELLMAN – « An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. », *IEEE Trans. Inf. Theory* **24** (1978), p. 106–110.
- [52] J. M. POLLARD – « Theorems of factorization and primality testing », *Proceedings of the Cambridge Philosophical Society* **76** (1974), no. 3, p. 521–528.
- [53] — , « A Monte Carlo method for factorization », *BIT Numerical Mathematics* **15** (1975), no. 3, p. 331–334.
- [54] — , « Monte Carlo methods for index computation (mod p) », *Math. Comput.* **32** (1978), p. 918–924.
- [55] J. M. POLLARD – « Kangaroos, Monopoly and Discrete Logarithms », *J. Cryptology* **13** (2000), no. 4, p. 437–447.
- [56] V. R. PRATT – « Every prime has a succinct certificate », *SIAM J. Comput.* **4** (1975), no. 3, p. 214–220.
- [57] M. O. RABIN – « Probabilistic algorithm for testing primality. », *J. Number Theory* **12** (1980), p. 128–138.
- [58] C.-P. SCHNORR – « Efficient signature generation by smart cards », *J. Cryptology* **4** (1991), no. 3, p. 161–174.
- [59] D. SHANKS – « Class number, a theory of factorization and genera », *Proceedings of Symposia in Pure Mathematics* **20** (1970), p. 415–440.
- [60] — , « Five number theoretic algorithms », *Proceedings of the Second Manitoba Conference on Numerical Mathematics*, 1973, p. 51–70.
- [61] R. SOLOVAY et V. STRASSEN – « A fast monte-carlo test for primality », *SIAM J. Comput.* **6** (1977), no. 1, p. 84–85.
- [62] D. R. STINSON – « Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem », *Math. Comput.* **71** (2002), no. 237, p. 379–391.
- [63] A. TARDY-CORFDIR et H. GILBERT – « A Known Plaintext Attack of FEAL-4 and FEAL-6 », *Advances in Cryptology – CRYPTO’91* (J. Feigenbaum, éd.), Lect. Notes Comput. Sci., vol. 576, Springer, août 1992, p. 172–181.
- [64] A. TONELLI – « Bemerkung über die auflösung quadratischer congruenzen », *Nachrichten von der Königlichen Gesellschaft der Wissenschaften und der Georg-Augusts-Universität zu Göttingen* (1891), p. 344–346.
- [65] S. VAUDENAY – « On the need for multipermutations : Cryptanalysis of MD4 and SAFER », *Fast Software Encryption – FSE’94* (B. Preneel, éd.), Lect. Notes Comput. Sci., vol. 1008, Springer, décembre 1994, p. 286–297.
- [66] — , « Hidden Collisions on DSS », *Advances in Cryptology – CRYPTO’96* (N. Koblitz, éd.), Lect. Notes Comput. Sci., vol. 1109, Springer, août 1996, p. 83–88.
- [67] — , « Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ... », *Advances in Cryptology – EUROCRYPT 2002* (L. R. Knudsen, éd.), Lect. Notes Comput. Sci., vol. 2332, Springer, avril / mai 2002, p. 534–546.

- [68] D. WAGNER – « The Boomerang Attack », *Fast Software Encryption – FSE'99* (L. R. Knudsen, éd.), Lect. Notes Comput. Sci., vol. 1636, Springer, mars 1999, p. 156–170.
- [69] M. J. WIENER – « Cryptanalysis of short RSA secret exponents », *IEEE Trans. Inf. Theory* **36** (1990), no. 3, p. 553–558.

INDEX

A

A5/1 128
AES 55
algorithme
 « éléver au carré et multiplier », 144
 de Berlekamp-Massey, 116
 de calcul d'indice, 157
 d'exponentiation dichotomique, 144
 de calcul d'indice, 154
 de Dixon, 206
 de Fermat, 188
 de Lehman, 191
 de multi-exponentiation, 146
 de Pohlig-Hellman, 152
 de Shanks, 147
 de Tonelli-Shanks, 203
 de Viterbi, 20
 des divisions successives, 188
 λ de Pollard, 165
 « pas de bébé, pas de géant », 147
 ρ de Pollard, 148
analyse fréquentielle 1
arbre de Merkle 268
attaque par corrélation 125
avantage d'un distingueur 42

B

base de facteurs 157
Berlekamp-Massey, algorithme de 116
bigramme 2
blanchiment 49
borne de Chernoff 74
bourrage 37
Boyd, signature de 254

C

César, chiffrement de 2
calcul d'indice 157
Carmichael, nombre de 176
carré latin 18

certificat de primalité 174
Chernoff, borne de 74
chiffrement
 RSA naïf, 219
chiffrement de Vernam 19
chiffrement par bloc,
 AES, 55
 DEAL, 89
 DES, 82, 83, 96
 DES-X, 49
 FEAL, 83
 Ladder-DES, 107
 SAFER, 99
 Triple-DES, 31
chiffrement par flot,
 A5/1, 128
générateur à signal d'arrêt, 116
générateur de Geffe, 124
générateur par auto-rétrécissement,
 118
générateur par rétrécissement, 118
RC4, 135
Toyocrypt, 120
chiffrement par substitution 1
 affine, 3
 de César, 2
 de Hill, 12
 de Vigenère, 9
 mono-alphabétique, 1
 poly-alphabétique, 8
chiffrement par transposition 14
 par colonnes, 16
 scytale, 14
chiffrement parfait 17
complexité linéaire 113
contrefaçon
 existentielle, 242
 universelle, 242
critère de Korselt 176
cryptanalyse
 différentielle, 82
 différentielle impossible, 89

Exercices et problèmes de cryptographie

intégrale, 104
linéaire, 96
par saturation, 104

D

Davies-Meyer, construction de 73
De Jonge et Chaum,
attaque de, 252
signature de, 243
DEAL 89
DES 45, 82, 83, 96
propriété de complémentation, 48
3-DES voir Triple-DES
DES-X 49
Dickman-De Bruijn, fonction de 154
Diffie-Hellman,
mise en accord de clé, 230
problème calculatoire de 232
Dirichlet, théorème d'approximation de 190
distance de Hamming 97
distinguier 40, 42
diversification de clé 40
divisions successives 188
Dixon, méthode de 206
DSA 262

E

ElGamal,
chiffrement de, 235
signature d', 258
Enigma 23
entier composé 173
entier friable 154
Euler, pseudo-premier d', 181
exponentiation discrète 143

F

FEAL 83
Feistel, index de 39
Fermat,
méthode de, 188
nombre de, 186

pseudo-premier de, 175
petit théorème de, 174
fonction de compression
de Davies-Meyer, 73
de Matyas-Meyer-Oseas, 73
fonction de hachage itérée
de Merkle-Damgaard 67
de Rabin 73
fonction de redondance 251
fonction interne 40

G

générateur
à signal d'arrêt, 116
de Geffe, 124
par auto-rétrécissement, 118
par rétrécissement, 118
Geffe, générateur de 124

H

Hamming, distance de 97
Hill, chiffrement de 12

I

indice de coïncidence 10
indice de Sinkov 25

J

Jacobi, symbole de 180

K

Kasiski, test de 9
Korselt, critère de 176
Kraitchik, méthode de 157

L

Ladder-DES 107
Legendre, symbole de 179
Lehman, algorithme de 191
lemme d'empilement 99
loi de reciprocité quadratique 180

M

- masque jetable 19
- matrice état 56
- Matyas-Meyer-Oseas, construction de 73
- MD5 69
- Merkle, arbre de 268
- Merkle-Damgaard, fonction de hachage itérée 67
- Miller Rabin, test de primalité 182
- mise en accord de clé de Diffie-Hellman 230
- mode opératoire,
 - CBC, 38
- mode opératoire,
 - CBC, 32, 36
 - CTR, 33
 - ECB, 32
- module RSA 213
- multi-collision 76

N

- nombre
 - de Carmichael, 176
 - de Fermat, 186
- nombres premiers, théorème des 154

P

- PKCS #1 248
- Pohlig-Hellman, algorithme de 152
- Pollard,
 - algorithme λ de, 165
 - algorithme ρ de, 148
- polynôme de rétroaction 113
- Pratt, certificat de 174
- problème
 - du logarithme discret, 146
 - RSA, 215
 - calculatoire de Diffie-Hellman, 232
- pseudo-premier
 - d'Euler, 181
 - de Fermat, 175

R

- Rabin, fonction de hachage itérée de 73
- réduite 223
- réseau de substitutions-permutations 55
- résidu quadratique 179
- RC4 135
- redondance à droite 252
- registre à décalage à rétroaction linéaire 112
 - à décalage irrégulier, 116
 - combiné, 116
 - filtré, 116
- RFC2040, processus de bourrage 37
- Rijndael voir AES
- RSA,
 - chiffrement, 219
 - signature, 241

S

- sécurité sémantique 33, 219
- SAFER 99
- schéma de Feistel 39
- Schnorr, signature de 261
- Shanks, algorithme de 147
- signature
 - d'ElGamal naïf, 258
 - de Boyd, 254
 - de De Jonge et Chaum, 243
 - de Groth, 271
 - de Lamport, 265
 - de Schnorr, 261
 - DSA, 262
 - jetable, voir de Lamport
 - RSA naïf, 241
- Sinkov, indice de 25
- Solovay-Strassen, test de 182
- suite chiffrante 111
- surchiffrement
 - double, 50
 - triple, 52
- symbole de Jacobi 180
- symbole de Legendre 179
- système de décomposition 166

Exercices et problèmes de cryptographie

T

- table d'approximation linéaire [96](#)
- test de primalité
 - de Fermat, [175](#)
 - de Miller-Rabin, [182](#)
 - de Pépin, [186](#)
 - de Solovay-Strassen, [182](#)
- théorème
 - de Fermat (petit), [174](#)
- d'approximation de Dirichlet [190](#)
- des nombres premiers [154](#)

- Tonelli-Shanks, algorithme de [203](#)
- Toyocrypt [120](#)
- Triple-DES [31](#)

V

- vecteur d'initialisation [32](#)
- Vernam, chiffrement de [19](#)
- Vigenère, chiffrement de [9](#)
- Viterbi, algorithme de [20](#)



Pour l'éditeur, le principe est d'utiliser des papiers composés de fibres naturelles, renouvelables, recyclables et fabriquées à partir de bois issus de forêts qui adoptent un système d'aménagement durable.

En outre, l'éditeur attend de ses fournisseurs de papier qu'ils s'inscrivent dans une démarche de certification environnementale reconnue.