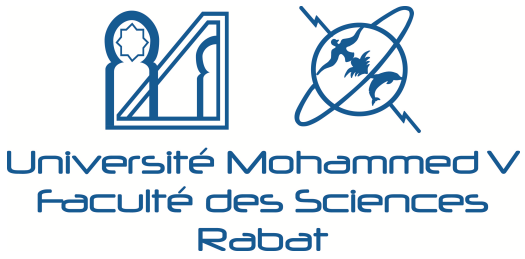
UNIVERSITÉ MOHAMMED V de Rabat
Faculté des Sciences



Département d'Informatique

Master
informatique appliquée offshoring

PROJET DE CRYPTOGRAPHIE

intitulé :

Cryptosystème d'ElGamal

Présenté par : Aouga Siham
Benyoussef Mehdi

Année universitaire 2018-2019

Table des matières

Introduction

Les communications ont toujours constitué un aspect important dans l'acquisition de nouvelles connaissances et l'essor de l'humanité. Le besoin d'être en mesure d'envoyer un message de façon sécuritaire est probablement aussi ancien que les communications elles-mêmes. D'un point de vue historique, c'est lors des conflits entre nations que ce besoin a été le plus vif. Dans notre monde moderne, où diverses méthodes de communication sont utilisées régulièrement, le besoin de confidentialité est plus présent que jamais à une multitude de niveaux. Par exemple, il est normal qu'une firme désire protéger ses nouveaux logiciels contre la piraterie, que les institutions bancaires veuillent s'assurer que les transactions sont sécuritaires et que tous les individus souhaitent que l'on protège leurs données personnelles. Le besoin de communications sécuritaires a donné naissance à la science que nous appelons cryptologie.

Le présent rapport abordera en premier lieu une présentation de la cryptographie classique et moderne, puis le cryptosystéme El Gamal, ensuite des Algorithmes de cryptanalyse El Gamal, et enfin, une partie sur la sécurité El Gamal.

Chapitre 1

Cryptographie

1.1 Cryptographie classique

La cryptographie classique décrit la période avant les ordinateurs. Elle traite des systèmes reposant sur les lettres et les caractères d'une langue naturelle (allemand, anglais, français, etc...). Les principaux outils utilisés remplacent des caractères par des autres et les transposent dans des ordres différents. Les meilleurs systèmes (de cette classe d'algorithmes) répètent ces deux opérations de base plusieurs fois. Cela suppose que les procédures (de chiffrement ou déchiffrement) soient gardées secrètes.

1.1.1 Chiffrement par Transposition

Elles consistent, par définition, à changer l'ordre des lettres. C'est un système simple, mais peu sûre pour de très brefs messages car il y a peu de variantes. Ainsi, un mot de trois lettres ne pourra être transposé que dans 6 (=3 !) positions différentes. Par exemple, "col" ne peut se transformer qu'en "col", "clo", "ocl", "olc", "lco" et "loc". Lorsque le nombre de lettres croît, il devient de plus en plus difficile de retrouver le texte original sans connaître le procédé de brouillage. Ainsi, une phrase de 35 lettres peut être disposée de $35! = 1040$ manières différentes. Ce chiffrement nécessite un procédé rigoureux convenu auparavant entre les parties. Une transposition rectangulaire consiste à écrire le message dans une grille rectangulaire, puis à arranger les colonnes de cette grille selon un mot de passe donné (le rang des lettres dans l'alphabet donne l'agencement des colonnes).

Exemple de chiffrement par transposition :

D'après des documents1 déclassifiés dans les années 1990, le message suivant a été intercepté le 12 mars 1942 par le service de renseignement radio (Radio Intelligence Division) de la FCC américaine, depuis sa station d'écoute à Laredo. Le message était envoyé par un agent allemand basé à Rio de Janeiro.

Message chiffré : CARTM IELHX YEERX DEXUE VCCXP EXEEM OEUNM CMIRL XRTFO CXQYX EXISV NXMAH GRSML ZPEMS NQXXX ETNIX AAEXV

UXURA FOEAH XUEUT AFXEH EHTEN NMFXA XNZOR ECSEI OAINE MRCFX SENSD PELXA HPRE

Clé de transposition : 8 4 9 14 1 2 16 10 3 17 15 19 11 5 20 6 7 12 13 18

Déchiffrement : le déchiffrement se fait en remplissant les colonnes verticalement, dans l'ordre défini par la clé. On commence par remplir de haut en bas la colonne numérotée 01 avec les huit premiers caractères du message chiffré : CARTMIEL. On continue en remplissant de la même façon la colonne numérotée 02 avec les huit caractères suivants HxYEERxD etc.

08	04	09	14	01	02	16	10	03	17	15	19	11	05	20	06	07	12	13	18
S	P	R	U	C	H	x	S	E	C	H	S	N	U	L	L	x	V	O	N
V	E	S	T	A	x	A	N	x	S	T	E	I	N	x	x	Q	U	E	E
N	x	M	A	R	Y	x	Q	U	E	E	N	x	M	A	R	Y	x	A	M
x	E	L	F	T	E	N	x	E	I	N	S	A	C	H	T	x	U	H	R
M	E	Z	x	M	E	Z	x	V	O	N	D	A	M	P	F	E	R	x	C
A	M	P	E	I	R	O	x	C	A	M	P	E	I	R	O	x	A	U	F
H	O	E	H	E	x	R	E	C	I	F	E	x	R	E	C	I	F	E	x
G	E	M	E	L	D	E	T	x	.	.	.	.	.	.	.	.	.	.	.

FIGURE 1.1.1 – Application d'une Transposition

Le texte clair est écrit horizontalement. Il s'avère être en Allemand : Spruch 60. Von VESTA An STEIN. QUEEN MARY am Elften eins acht Uhr MEZ von Dampfer CAMPEIRO auf hoehe RECIFE gemeldet.

Soit, en Français : Texte 60, de VESTA pour STEIN. Queen Mary signalé au large de Recife le 11 à 18 heures HEC par le vapeur Campeiro.

1.1.2 Chiffrement par Substitution monoalphabétique et Polyalphabétique

La substitution consiste à effectuer des dérivations pour que chaque caractère du message chiffré soit différent des caractères du message en clair. Le destinataire légitime du message applique la dérivée inverse au texte chiffré pour recouvrer le message initial. La complexité des systèmes à substitutions dépend de trois facteurs :

— la composition spécifique de l'alphabet utilisé pour chiffrer ou pour communiquer,
— le nombre d'alphabets utilisés dans le cryptogramme,
— la manière spécifique dont ils sont utilisés.

Substitution monoalphabétique

Chaque lettre est remplacée par une autre lettre ou symbole. Parmi les plus connus, on citera le chiffre de Vignére, le chiffre affine. ces chiffres sont sensibles à l'analyse de fréquence d'apparition des lettres (nombre de fois qu'apparait une même lettre dans un texte). De nos jours, ces chiffres sont utilisés pour le grand public, pour les énigmes de revues ou de journaux.

Chiffre de Vignére

C'est une amélioration décisive du chiffre de César. Sa force réside dans l'utilisation non pas d'un, mais de 26 alphabets décalés pour chiffrer un message. On parle du carré de Vigenère. Ce chiffre utilise une clef qui définit le décalage pour chaque lettre du message (A : décalage de 0 cran, B : 1 cran, C : 2 crans, ..., Z : 25 crans).

Chiffre affine

On dit qu'une fonction est affine lorsqu'elle est de la forme x $\rightarrow$ a $\times$ x + b, c'est-à-dire un polynôme de degré 1. Une fonction linéaire est une fonction affine particulière.
L'idée est d'utiliser comme fonction de chiffrement une fonction affine du type y = (ax + b) mod 26,
où a et b sont des constantes, et x et y sont des nombres correspondant aux lettres de l'alphabet (A=0,B=1,...). On peut remarquer que si a = 1, alors on retrouve le chiffre de César où b est le décalage (le k du chiffre de César).

Propriété de neutralité :
si b = 0, alors "a" est toujours chiffré "A" car il ne subit aucun décalage. En effet, si aucun décalage n'a lieu, l'alphabet de départ se retrouve chiffré par lui même, et donc ne subit aucune modification.
Pour le chiffre affine, la clé est constituée de (k_1, k_2) où $k_1, k_2 \in$[0,25] et telle que

$$gcd(k_1, 26) = 1.$$

Le chiffrement en lui-même est donné par

$$c_i = \mathrm{f}(m_i) = k_1 \times m_i + k_2 \bmod 26.$$

Pour le déchiffrement, il vient

$$m_i = f^{-1}(c_i) = k_1^{-1} \times (c_i - k_2) \bmod 26.$$

Par le chiffre affine, on obtient 312 clés possibles. En effet, pour obéir à la propriété de k_1, il n'y a que 12 choix possibles. Et puisque k_2 peut prendre n'importe quelle valeur dans [0,25], il vient $12 \times 26 = 312$.

Exemple :
Soient la clé = (k1,k2) = (3,11)
Transformation de chiffrement :

$$c_i = \mathrm{f}(m_i) = 3 \times m_i + 11 \bmod 26.$$

Transformation de déchiffrement :

$$k_1^{-1} = (3 - 1) \bmod 26. \quad [\text{car } 3 \times 9 \bmod 26 = 1]$$

$$m_i = f^{-1}(c_i) = 9 \times (c_i - 11) \bmod 26.$$

Ainsi, pour une suite de lettres telle que 'NSA' $\rightarrow$ 13 18 0 $\rightarrow$ 24 13 11 $\rightarrow$'YNL'.

Substitution Polyalphabétique

Il s'agit ici de chiffrer un groupe de n lettres par un autre groupe de n symboles. On citera notamment le chiffre de Playfair et le chiffre de Hill. Ce type de chiffrement porte également le nom de substitutions polygrammiques.

Chiffre de Playfair

Le chiffre de Playfair utilise un tableau de 5×5 lettres, contenant un mot clé ou une phrase. La mémorisation du mot clé et de 4 règles à suivre suffisent pour utiliser ce chiffrement.

Remplir le tableau avec les lettres du mot clé (en ignorant les doublons), puis le compléter avec les autres lettres de l'alphabet dans l'ordre (soit en omettant la lettre w, soit en occupant une même case pour les lettres I et J suivant les versions). Le mot clé peut être écrit en ligne, en colonne ou même en spirale.

Pour chiffrer un message, il faut prendre les lettres 2 par 2 et appliquer les règles suivantes en fonction de la position des lettres dans la table :

- si les 2 lettres sont identiques (ou s'il n'en reste qu'une) mettre un 'X' après la première lettre. Chiffrer la nouvelle paire ainsi constituée et continuer avec la suivante. Dans certaines variantes, on utilise 'Q' au lieu du 'X', mais n'importe quelle lettre peut faire l'affaire,
- si les lettres se trouvent sur la même ligne de la table, il faut les remplacer par celles se trouvant immédiatement à leur droite (en bouclant sur la gauche si le bord est atteint), si les lettres apparaissent sur la même colonne, les remplacer par celles qui sont juste en dessous (en bouclant par le haut si le bas de la table est atteint),
- sinon, remplacer les lettres par celles se trouvant sur la même ligne, mais dans le coin opposé du rectangle défini par la paire originale.

Pour chiffrer le digramme 'OR' par exemple, trois configurations peuvent se présenter dans le tableau :

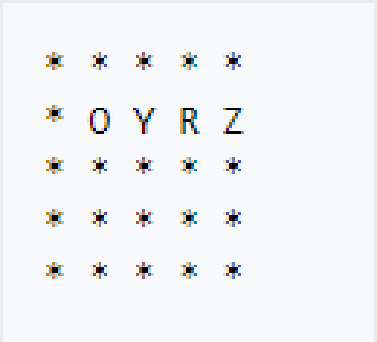


FIGURE 1.1.2 – OR → YZ

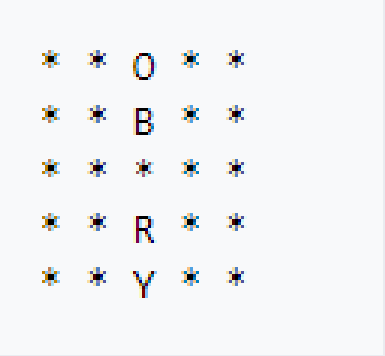


FIGURE 1.1.3 – OR → BY

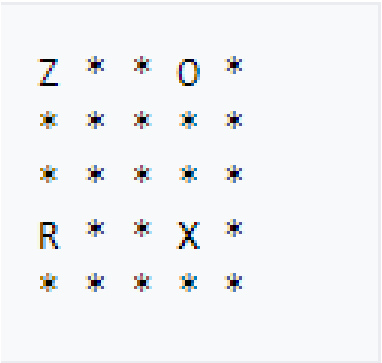


FIGURE 1.1.4 – OR → ZX

Chiffre de Hill

Chaque caractère est d'abord codé par un nombre compris entre 0 et n-1 (son rang dans l'alphabet diminué de 1 ou son code ASCII diminué de 32). Les caractères sont alors regroupés par blocs de p caractères formant un certain nombre de vecteurs $X = (x_1, x_2, \cdots, x_p)$. Les nombres x_i étant compris entre 0 et n-1, on peut les considérer comme des éléments de $\mathbb{Z}/n\mathbb{Z}$ et X est alors un élément de $(\mathbb{Z}/n\mathbb{Z})^p$.

On a construit au préalable une matrice p ×S p d'entiers : A. Le bloc X est alors chiffré par le bloc Y = AX, le produit s'effectuant modulo n.

Pour déchiffrer le message, il s'agit d'inverser la matrice A modulo n. Cela peut se faire si le déterminant de cette matrice possède un inverse modulo n (c'est-à-dire, d'après le théorème de Bachet-Bézout, si det(A) est premier avec n).

En effet, le produit de A et de la transposée de sa comatrice donne

$$A\ {}^{t}\mathrm{com}A = {}^{t}\mathrm{com}A\ A = \det A\ I_p$$

(où I_p désigne la matrice identité de taille p) donc s'il existe un entier k tel que

$$k \times \det(A) \equiv 1 \pmod{n}$$

alors, en notant B n'importe quelle matrice congrue modulo n à k tcom(A), on aura

$$\mathrm{AB} \equiv \mathrm{BA} \equiv I_p \pmod{}\ \mathrm{n},$$

soit encore

$$\mathrm{Y}{=}\mathrm{AX} \Leftrightarrow \mathrm{X}{=}\mathrm{BY}.$$

1.2 Cryptographie moderne

La cryptographie entre dans son ère moderne avec l'utilisation intensive des ordinateurs, c'est-à-dire à partir des années septante. Dans la cryptographie moderne, on utilise aussi des problèmes mathématiques que l'on ne sait pas (encore) résoudre, par exemple factoriser des grands nombres (chiffre RSA) ou résoudre le problème général du sac à dos (chiffre de Merkle-Hellman). Plus anecdotique, on voit aussi apparaître les deux personnages récurrents les plus célèbres de la cryptographie : Alice et Bob (ou Bernard en français).

1.2.1 Chiffrement symétrique

Le chiffrement symétrique, que l'on nomme couramment chiffrement conventionnel, est basé sur des fonctions mathématiques réversibles. Le chiffrement symétrique repose sur un principe de clé unique pour chiffrer et déchiffrer.

Cette clé possède plusieurs appellations :

— Clé secrète
— Clé partagée

On parle de chiffrement conventionnel puisque c'est le premier chiffrement par clé à avoir été découvert et utilisé.

Principe de fonctionnement

Le chiffrement symétrique se déroule en 4 étapes, de la manière suivante :

1ère étape

Génération de la clé secrète par Alice
Envoi de cette clé secrète à Bob, de manière sécurisée

2ème étape

Chiffrement du message par Alice, avec la clé secrète
Envoi de ce message chiffré à Bob

3ème étape

Réception du message chiffrée par Alice
Déchiffrement du message avec la clé secrète reçue auparavant

Il existe 2 types d'algorithme de chiffrement :

— Le chiffrement de flux

— Le chiffrement par bloc

Chiffrement symétrique par flux

Définition

Un algorithme de chiffrement par flux (ou chiffrement par flot de l'anglais stream cipher) est un algorithme agissant en continu sur les données. Il ne nécessite pas d'avoir toutes les données pour commencer à chiffrer, le chiffrement de flux agit sur chaque bit, l'un après l'autre. Ce type de chiffrement est souvent utilisé pour les communications en temps réel telles que le WI-FI (RC4), puisqu'il a la particularité d'être beaucoup plus rapide que n'importe quel algorithme de chiffrement par bloc. De plus, au niveau des données chiffrées en sortie, le chiffrement par flux ne donnera pas forcement le même résultat en sortie alors que pour un bloc donné un chiffrement par bloc aura toujours le même résultat. Un algorithme de flux fonctionne avec ce que l'on appelle un générateur pseudo-aléatoire (keystream en anglais), c'est une séquence de bits précise utilisée en tant que clé. Le chiffrement se fait par la combinaison du keystream et du message, le plus souvent par une opération XOR (OU exclusif).

Chiffrement de flux « synchronous »

Avec un chiffrement de flux synchronous, un flux de nombre pseudo-aléatoire est généré indépendamment du texte de base et du texte chiffré. Ce flux est utilisé pour chiffrer le texte de base, ou pour déchiffrer le texte chiffré. En général, le flux est composé de chiffres binaires, et le chiffrement se fait par une opération XOR entre le keystream et les données de base. Les deux communicants doivent être exactement à la même étape (synchrone) pour que le déchiffrement se passe correctement. Si des ajouts ou des suppressions sont réalisés pendant le transfert du message, on perd la synchronisation. Cependant, si un bit est altéré pendant le transfert, cela n'affecte pas le reste du message et donc la synchronisation est toujours présente.

Chiffrement de flux self-synchronizing

Les algorithmes de chiffrement de flux appelés self-synchronizing, calcule le keystream à partir du message lui-même. Le plus simple algorithme de chiffrement self-synchronizing est l'autokey, il utilise le message lui-même comme clé. Lorsqu'un bit est ajouté ou supprimé pendant le transfert des données, l'algorithme l'identifiera puisque la clé est issue du message d'origine lui-même, contrairement aux algorithmes synchronous qui génèrent des keystream aléatoires indépendamment du message d'origine. Les algorithmes de chiffrement de flux self-synchronizing sont beaucoup moins répandus que les synchronous

Méthode XOR

La méthode XOR, appelée plus généralement fonction OU Exclusif est un opérateur logique. Le principe repose sur 2 opérandes qui peuvent avoir comme valeur VRAI (1) ou FAUX (0), le résultat prendra lui aussi comme valeur VRAI ou FAUX ; VRAI dans le cas ou seulement l'un des deux est VRAI.

Chiffrement symétrique par bloc

Définition

Un algorithme de chiffrement par bloc (Block Cipher en anglais) transforme des blocs de données de taille fixe en bloc de données chiffrées de la même taille. Les blocs font généralement 128 bits, mais ils peuvent aller de 32 à 256 bits selon l'algorithme. La transformation reste la même pour chaque bloc. Il existe 4 modes de chiffrement par bloc : Electronic CodeBook (ECB), Cipher Block Chaining (CBC), Cipher FeedBack (CFB) ou Output FeedBack (OFB). Ces 4 modes ne dépendent pas de l'algorithme utilisé, néanmoins tous les algorithmes ne peuvent pas forcément utiliser ces 4 modes.

1.2.2 Chiffrement asymétrique

C'est en 1976 que Whitfield Diffie et Martin Hellman, de l'Université Stanford, proposent un principe de chiffrement entièrement nouveau : la cryptographie à clé publique, ou asymétrique.

Expliquons leur procédé de façon imagée :

Alice doit recevoir un message de Bob, mais elle ne fait pas confiance au facteur qui pourrait ouvrir sa lettre. Comment peut-elle être sûre de recevoir ce message sans qu'il soit lu ?... Alice va d'abord envoyer à Bob un cadenas ouvert, dont elle seule possède la clé. Ensuite, Bob va placer son message dans une boîte, qu'il fermera à l'aide de ce cadenas, avant de l'envoyer à Alice. Le facteur ne pourra donc pas ouvrir la boîte, puisque seule Alice possède la clé !

Ainsi, un système cryptographie à clé publique est en fait basé sur deux clés :

Une clé publique, pouvant être distribuée librement, c'est le cadenas ouvert
Une clé secrète, connue uniquement du receveur, c'est le cadenas fermé
C'est la raison pour laquelle on parle de chiffrement asymétrique.

En résumé, on dispose d'une fonction P sur les entiers, qui possède un inverse S. On suppose qu'on peut fabriquer un tel couple (P,S), mais que connaissant uniquement P, il est impossible (ou au moins très difficile) de retrouver S. Autrement dit, il faut déterminer mathématiquement des fonctions difficilement inversibles, ou "à sens unique".

1.3 Cryptanalyse

La cryptanalyse est la technique qui consiste à déduire un texte en clair d'un texte chiffré sans posséder la clé de chiffrement. Le processus par lequel on tente de comprendre un message en particulier est appelé une attaque.

Une attaque est souvent caractérisée par les données qu'elle nécessite :

— **attaque sur texte chiffré seul (ciphertext-only en anglais) :** le cryptanalyste possède des exemplaires chiffrés des messages, il peut faire des hypothèses sur les messages originaux qu'il ne possède pas. La cryptanalyse est plus ardue de par le manque d'informations à disposition.
— **attaque à texte clair connu (known-plaintext attack en anglais) :** le cryptanalyste possède des messages ou des parties de messages en clair ainsi que les versions chiffrées. La cryptanalyse linéaire fait partie de cette catégorie.
— **attaque à texte clair choisi (chosen-plaintext attack en anglais) :** le cryptanalyste possède des messages en clair, il peut créer les versions chiffrées de ces messages avec l'algorithme que l'on peut dès lors considérer comme une boîte noire. La cryptanalyse différentielle est un exemple d'attaque à texte clair choisi.
— **attaque à texte chiffré choisi (chosen-ciphertext attack en anglais) :** le cryptanalyste possède des messages chiffrés et demande la version en clair de certains de ces messages pour mener l'attaque.

Chapitre 2

Le Cryptosystéme El Gamal

2.1 présentation

Le cryptosystème d'ElGamal, ou chiffrement El Gamal (ou encore système d'El Gamal) est un protocole de cryptographie asymétrique inventé par Taher Elgamal en Août 1984 d'où l'algorithme tiens son nom, et construit à partir du problème du logarithme discret et l'Échange de clés Diffie-Hellman.

Ce protocole est utilisé par le logiciel libre GNU Privacy Guard dont les versions récentes implantent jusque sa version sur les courbes elliptiques. Contrairement au chiffrement RSA, il n'a jamais été sous la protection d'un brevet.

L'article fondateur par Taher Elgamal présente un protocole de chiffrement, mais aussi une signature numérique, qui malgré leurs similarités (ils sont tous deux construit sur le problème du logarithme discret).

2.2 Fonctions à sens unique – probléme du logarithme discret (DLP)

La cryptographie à clef publique est basée sur le fait de disposer d'une fonction à sens unique (FSU dans la suite, en anglais one way function).

Autrement dit, il faut trouver une fonction E de chiffrement qui soit rapide à calculer mais que son inverse D= E^{-1} soi extrêmement longue à calculer. Les bonnes FSU sont des fonctions telles que la recherche de x à partir de F(x) soit un probléme mathématique réputé difficile (typiquement NP-complet).

Les sections suivantes présentent des fonctions à sens uniques qui sont couramment utilisées en cryptographie à clé publique. Ces fonctions sont essentiellement basées sur la difficulté du probléme du logarithme discret.

2.2.1 Un premier exemple de FSU : l'Exponentiation Modulaire

L'exemple le plus courant de fonction à sens uniques est l'exponentiation modulaire,qui se construit comme suit :

— Soient p et q deux entiers premiers et n=p.q.

— Soit e un entier inférieur à n, premier avec ϕ(n)=(p-1)(q-1)
La fonction d'exponentiation modulaire est alors définie par :

$$\begin{aligned} F_e \ : \ & \mathbb{Z}_n \to \mathbb{Z}_n \\ & \text{x} \to x^e \bmod \text{n} \end{aligned}$$

Cette fonction est rapide à calculer (en fait en temps linéaire de e et donc de n) grâce à un algorithme de type exponentiation rapide, encore appelé élévation récursive au carré.
La fonction inverse de F_e n'est autre que la fonction F_d , où d=e^{-1} mod $\phi(n)$.
Autrement dit on a : d.e $\equiv$ 1 mod (p-1)(q-1).
En effet, si y=F_e(x), alors $y^d = x^{e.d}$ mod n = $x^{1+\lambda\phi(n)}$ mod n =x.
Le calcul de d=e^{-1} mod $\phi(n)$ peut être fait facilement à partir de l'algorithme d'Euclide Etendu puisque e et $\phi(n)$ sont premiers entre eux.
En fait, la véritable difficulté pour calculer d est liée à l'évaluation de $\phi(n)$ =(p-1)(q-1).

Celle-ci est aisée si on connait la valeur de p et q. Sinon, le probléme se raméne à factoriser l'entier n en produits de nombres premiers.Cependant,malgré tous les traveaux menés sur le probléme de la factorisation d'entiers depuis 300 ans, aucun algorithme rapide n'a été publié à ce jour.

2.2.2 Un autre exemple de FSU basée sur la difficulté du logarithme discret

La fonction exponentielle dans un groupe cyclique est trés utilisée en cryptographie pour la construction de fonctions à sens unique.
En effet, son inverse, le logarithme discret est un probléme considéré extrêmement coûteux à résoudre pour de trés nombreuses instances. On poura citer par exemple le protocole d'échange de clé de Diffie-Hellman et l'algorithme de **El Gamal**.

Denition.
Un groupe cyclique G, est un groupe dans lequel il existe un élément g tel que tout élément du groupe puisse s'exprimer sous forme d'un multiple/puissance de g, cet élément g est appelé générateur du groupe et on note G =< g >.

— **En notation additive :** (G,+) [n]g
— **En notation multiplicative :** (G,·) g^n

Denition.
Soit G un groupe et g $\in$ G, alors le groupe sous-groupe H généré par g, noté H =< g >, est le plus petit sous-groupe de G contenant g.

Denition.

L'ordre d'un élément g d'un groupe G est l'ordre du sous-groupe H =< g > généré par cet élément. L'ordre de g est noté ordre(g) ou o(g). Si l'ordre est ni, il est le plus petit entier m > 0

— En notation additive : mg = 0
— En notation multiplicative : gm = 1

On peut dire que si l'ordre de g est fini ordre(g) = |H| = | < g > | la cardinalité sous-groupe H =< g > généré par g.

Exemple Trouver l'ordre de l'élément 2 dans

— $F_{11}^* = (\mathbb{Z}/_{11}\mathbb{Z})^* = \{1,2,...,10\}$ le groupe multiplicatif de restes modulo 11
— $F_{17}^* = (\mathbb{Z}/_{17}\mathbb{Z})^* = \{1,2,...,16\}$ le groupe multiplicatif de restes modulo 17

Dans F_{11}^* nous avons $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 \equiv 5$,$2^5 \equiv 10$, $2^6 \equiv 9$, $2^7 \equiv 7$, $2^8 \equiv 3$, $2^9 \equiv 6$. Mais $2^{10} \equiv 1$ et on commence à répéter les éléments.
Donc ordre(2) = 10 et il génére tout le groupe G.

Dans F_{17}^*. nous avons $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$,$2^5 \equiv 15$, $2^6 \equiv 13$, $2^7 \equiv 9$
Mais $2^8 \equiv 1$ et on commence à répéter les éléments.Donc 2 n'est pas un générateur pour F_{17}^*.

Il génére que le sous-groupe H =<2>= {1,2,4,8,9,13,15,16} et son ordre est la cardinalité de ce sous-groupe, donc 8.
Le probléme DLP peut être formulé pour un groupe géenérique H et < g >= G ⊂ H. On va étudier aprés le DLP sur des groupes particuliers : le groupe multiplicatif $(\mathbb{Z}/_P\mathbb{Z})^*$. Pour certains groupes le probléme du logarithme discret est "difficile" : soit on ne connaît aucun algorithme sous-exponentiel qui résout le probléme DLP, soit on ne connaît aucun algorithme qui résout le probléme DLP pour des tailles grandes (disons de 1024 bits et plus).

2.3 Construction des clefs de chiffrements pour ELGamal

Si on prend p premier, $E_p = (\mathbb{Z}/_{11}\mathbb{Z})^*$ est un groupe cyclique à p-1 éléments. Si g est un générateur de E_p, on a :

$$E_p = \{\ 1,\ g,\ g^2,\ \ldots,\ g^{p-2}\ \}$$

Donc si on prend un élément x de E_p ; on a :

— D'une part x est compris entre 0 et p-1,
— D'autre part il existe un entier s dans 1, 2, ..., p-2 tel que :

$$\text{x} = g^s \bmod \text{p}$$

Supposons qu'on dispose d'un nombre premier p, on est certain de pouvoir trouver un générateur g ($1 \leq \text{g} \leq$ p-1), de E_p.
On choisit un entier s quelconque dans {1, 2, ..., p-2}, et on calcule :

$$\text{x} = g^s \bmod \text{p}$$

Le triplet (p, g, x) forme alors la clef publique du chiffrement ELGamal, et l'entier s la clef privée. Pour des raisons algorithmiques, on ajoutera l'entier p à la clef privée qui devient alors le couple (p, s).

La fiabilité du système est assurée par le fait que toute personne ayant seulement connaissance de la clef publique ; se trouve dans l'incapacité de déterminer s (car cela revient à déterminer un logarithme discret).

2.4 Le chiffrement par ELGamal

Prenons comme message élémentaire, un nombre m compris entre 0 et p-1.
Comme nous l'avons déjà dit, le schéma de ELGamal est un chiffrement de type résiduel.
Le chiffrement s'effectue par m' = a×m mod p.
Le déchiffrement par m" = (1/a) × m'mod p, avec pgcd(a, p) = 1.

Pour chiffrer, il faut déterminer un élément a, premier avec p, qui reste secret aux yeux de tous sauf à ceux du destinataire du message.

Comme p est un nombre premier, tout nombre entier a, choisi dans l'intervalle [1, ..., p-1], est premier avec p. Soit k un nombre pris au hasard entre 2 et p-2, k peut alors être considéré comme l'indice par rapport à g, d'un élément de E_p. Soit α cet élément, on a :

$$\alpha = g^k \bmod \text{p (logarithme discret).}$$

Posons alors

$$\text{a} = x^k \bmod \text{p.}$$

a est bien premier avec p, et convient donc pour le chiffrement résiduel. On chiffre alors le message m par :

$$\text{m'} = \text{m } x^k \bmod \text{p.}$$

On envoie m' ainsi que α.

La sécurité du chiffrement est assuré par le fait que, connaissant α et le clef publique, personne ne peut déterminer k, et donc ne peut déchiffrer en calculant :

$$m'' = m' \ (1/x^k) \bmod p.$$

Remarque : Pourquoi k est compris entre 2 et p-2 ?

Par définition, k étant l'indice d'un élément de E_p par rapport au générateur g, il est compris entre 0 et ϕ(p)-1 = p-2 acr p est premier.

— Néanmoins, si $k = 0$, alors on chiffre le message m par :
$$m' = m \ x^0 \bmod p = m \bmod p \ .$$
On a donc m' = m : le message chiffré est le même que le message original ; chiffrer ne sert donc à rien.
— De même, si k=1, le message chiffré est alors
$$m' = m \ x \bmod p.$$

Cette fois, le message a subi une transformation, mais toute personne connaissant le clef publique (p, g, x) peut alors déchiffrer en calculant l'inverse de x modulo p, et en posant :

$$m'' = m' \ (1/x) \bmod p.$$

Le chiffrement n'est alors pas efficace, car k est facilement determinable.

En prenant k compris entre 2 et p-2, on s'assure donc qu'une personne connaissant α, p, g et x, ne pourra pas déterminer x^k (ni son inverse).

2.5 Le déchiffrement par ELGamal

La connaissance de l'inverse de x^k modulo p est nécessaire pour déchiffrer le message. Montrons donc en quoi la connaissance de la clef privée permet de s'en sortir.

Lors du chiffrement, on avait $\alpha = g^k \bmod p$ et $m' = m \ x^k \bmod p$. Avec l'élévation à la puissance s, on obtient : $\alpha^s = g^{ks} \bmod p$. De plus on a : $x = g^s \bmod p$, donc $x^k = g^{sk} \bmod p$.On a donc

$$x^k = \alpha^s \bmod p$$

Un inverse de x^k modulo p sera donc un inverse de α^s modulo p. La connaissance de s, secret, permet donc de déterminer $1/x^k$. On peut donc déchiffrer par

$$m'' = m' \ (1/\alpha^s \ ) \bmod p.$$

Comme on a choisit $0 \leq m \leq p-1$, on a bien

$$m'' = m.$$

2.6 Résumé

Clef publique :

— p premier (peut être partagé par un groupe d'utilisateurs).
— g < p (peut être partagé par un groupe d'utilisateurs).
— x = g^s mod p.

Clef privée :

— s < p.

Chiffrement :

— k choisi aléatoirement et premier avec p-1.
— $\alpha = g^k$ mod p.
— c = x^k m mod p.

Déchiffrement :

— m = c / α^s mod p.

2.7 Exemple

Prenons le nombre premier p = 181. Un générateur de E181 = $(\mathbb{Z}/181\mathbb{Z})^*$ est 23. On prend s=7. On a alors x = g^s mod p = 23^7 mod 181, donc x = 57. On obtient :

— La clef publique : (181, 23, 57).
— La clef privée : (181, 7).

Suposons que le message à chiffrer est une date de naissance : 07 12 90

Chiffrement :

Comme p est de longueur 3, il nous faut prendre des blocs de longueur 2 : [07, 12, 90], qui pour l'ordinateur devient : [7, 12, 90]. On prend alors un nombre k au hasard compris entre 2 et p-2, par exemple k = 6, on détermine alors

$$\alpha = g^k \text{ mod p} = 23^7 \text{ mod } 181 = 152 \text{ mod p.}$$

Puis on chiffre chaque élément de ce tableau en multipliant par x^k mod p = 57^6 mod 181 :

$$7 \times 576 \bmod 181 = 146 \bmod 181$$
$$12 \times 576 \bmod 181 = 121 \bmod 181$$
$$90 \times 576 \bmod 181 = 93 \bmod 181$$

Ce qui donne : [146, 121, 93], p étant de longueur 3, on rajoute des 0, si nécessaire, afin de n'avoir que des éléments de 3 chiffres.

[146, 121, 093]

On envoie alors le message : 146121093, ainsi que α : 152.

Déchiffrement :
On reçoit le message précédent que l'on décompose en bloc de 3 chiffres (car p est longueur 3), ce qui donne, pour l-ordinateur : [146, 121, 93]. On calcule alors l'inverse de α^s mod p = 176 mod 181, et on obtient $1/\ \alpha^s$ mod p = 36 mod 181 (car 36 $\times$ 176 mod 181 = 1 mod 181). On applique à chaque élément a du tableau la formule :

$$a \times (1/\alpha^s) \bmod 181.$$

$$146 \times 36 \bmod 181 = 7 \bmod 181$$
$$121 \times 36 \bmod 181 = 12 \bmod 181$$
$$93 \times 36 \bmod 181 = 90 \bmod 181$$

Et on obtient le tableau : [7, 12, 90]. Il ne reste plus qu'à compléter chaque bloc par des 0, si nécessaire, afin d'avoir des éléments de longueur p-1, c'est à dire 2. On obtient [07, 12, 90]. Ce qui correspond bien, après concaténation, à notre message original : 07 12 90.

2.8 Signatures ElGamal

Pour engendrer une paire de clefs, choisissez d'abord un nombre premier p et deux nombres g et s avec g $<$ p un générateur de E_p et s $<$ p un nombre aléatoire. Ensuite on calcule :

$$x = g^s \bmod p$$

— La clef publique est Le triplet (p, g, x).
— La clef privée est le couple (p, s).

Pour signer un message m, choissez d'abord un nombre aléatoire k, tel que k et p-1 soient premier entre eux. Ensuite calculez :

$$\alpha = g^k \bmod p$$

et utilisez l'algorithme d'Euclide étendu pour trouver la valeur de β qui satisfait l'équation suivante :

$$\text{m} = (\text{s}\alpha + \text{k}\beta) \bmod (\text{p-1}).$$

La signature est la paire : α et β. La valeur aléatoire k doit être tenue secrète.
Pour vérifier une signature, il faut confirmer que :

$$x^{\alpha}\ \alpha^{\beta} \bmod \text{p} = g^{m} \bmod \text{p}.$$

On a le **résumé** suivant :

Clef publique :

— p premier (peut être partagé par un groupe d'utilisateurs).
— g $<$ p (peut être partagé par un groupe d'utilisateurs).
— x $= g^{s} \bmod$ p

Clef privée :

— s $<$ p.

Signature :

— k choisi aléatoirement et premier avec p-1.
— α (signature) $= g^{k} \bmod$ p.
— β (signature) tel que m = s α + kβ mod (p-1).

Vérification :

— La signature est valide si $x^{\alpha}\ \alpha^{\beta} \bmod \text{p} = g^{m} \bmod$ p.

Il faut prendre une nouvelle valeur de k à chaque chiffrement et à chaque signature, et cette valeur doit être choisie aléatoirement. Si une troisième personne C récupère une valeur de k que A a utilisé, elle peut retrouver la clef privée. Si jamais C récupère deux messages signées ou chiffrés avec la même valeur k, elle peut retrouver x.

Exemple :

On prend p =11 et g=2 (on a déjà vu que 2 est un générateur de E11). La clef privée est s = 8. On calcule :

$$\text{x} = g^{s} \bmod \text{p} = 2^{8} \bmod 11 = 3.$$

— La clef publique est : (3, 2, 11)
— La clef privée est : (8, 11).

Pour authentifier M = 5, choisissez d'abord un nombre aléatoire k=9. On vérifie que pgcd(9, 10)=1. On calcule :

$$\alpha = g^k \text{ mod p} = 2^9 \text{ mod } 11 = 6$$

et on utilise l'algorithme d'Euclide étendu pour résoudre :

$$\text{m} = (\alpha\text{s} + \text{k}\beta\ ) \text{ mod } 10$$
$$5 = (8{\times}6 + 9{\times}\beta) \text{ mod } 10.$$

La solution est $\beta = 3$ et la signature est la paire *alpha*=6 et β=3.
Pour vérifier la signature :

$$x^\alpha \times \alpha^\beta \text{ mod p} = g^m \text{ mod p.}$$
$$3^6 \times 6^3 \text{ mod } 11 = 2^5 \text{ mod } 11.$$

2.9 El Gamal généralisé

En fait, si lon dispose dun groupe cyclique G pour lequel le logarithme discret est difficile et dun générateur g de G on peut utiliser l'algorithme suivant :
Alice choisit un entier a et calcule A = g^a, elle publie A et la description du groupe G mais garde a secret. Pour chiffrer le message m ∈ G, Bob choisit un entier k et calcule K = g^k, il calcule ensuite c = mA^k et envoie le couple m' = (K,c).
Pour déchiffrer, Alice calcule cK^{-a}.
Bien que tout groupe cyclique de cardinal n soit isomorphe à $\mathbb{Z}/_n\mathbb{Z}$ la diffculté du probléme du logarithme discret dépend fortement du groupe. Il y a des groupes où ce probléme est trés facile à résoudre, par exemple le groupe $\mathbb{Z}/_n\mathbb{Z}$ lui-même, en effet un générateur de ce groupe est un élément inversible et résoudre le logarithme discret consiste pour tout y ∈ $\mathbb{Z}/_n\mathbb{Z}$ à résoudre l'équation x = y ce qui est trés facile. Le fait que pour certains groupe G le probléme du logarithme discret soit diffcile implique que pour ces groupes on ne parvient pas expliciter l'isomorphisme entre G et $\mathbb{Z}/_n\mathbb{Z}$. En pratique, un groupe sera jugé meilleur qu'un autre si il présente un meilleur rapport entre la sécurité qu'il offre (le nombre d'opérations nécéssaires pour résoudre le problŁme du logarithme dicret associé) et la taille de sa description (taille de la clé publique). Selon ce critére il y a des meilleurs groupes que $\mathbb{Z}/_n\mathbb{Z}$ et c'est un domaine de recherche actuel trés actif que de déterminer des groupes toujours meilleurs. Notons que si l'on trouve un groupe avec de bonnes propriétés mais non cyclique, il suffit d'en prendre un sous-groupe engendré par un élément pour obtenir un groupe cyclique.
Les recherches portent principalement sur les groupes multiplicatifs des corps finis et sur les groupes des courbes elliptiques.

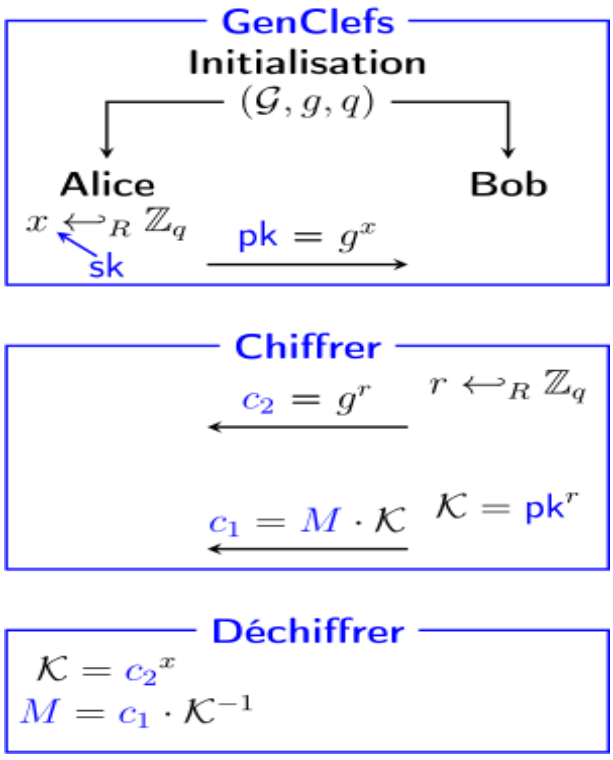


FIGURE 2.9.1 – El Gamal

Chapitre 3

Algorithmes de cryptanalyse El Gamal

3.1 Algorithme de Shanks

3.1.1 Introduction

En théorie algorithmique des nombres et en théorie des groupes, l'algorithme baby-step giant-step permet de résoudre le problème du logarithme discret dans un groupe cyclique quelconque. Il est dû à Daniel Shanks en 1971. C'est essentiellement un compromis temps-mémoire à partir de l'algorithme naïf par recherche exhaustive.

La difficulté du problème du logarithme discret est une hypothèse calculatoire sur laquelle reposent (plus ou moins directement) plusieurs schémas cryptographiques à clef publique, comme le chiffrement El Gamal, l'échange de clés Diffie-Hellman ou le protocole de Schnorr. C'est pourquoi pouvoir évaluer la difficulté de ce problème est une question importante en cryptographie.

3.1.2 Théorie

Soit G un groupe cyclique de générateur α d'ordre n, dont la loi est noté multiplicativement. Le problème du logarithme discret revient à chercher, pour β dans G, un entier x tel que $\alpha^x = \beta$.

La méthode naïve serait d'essayer successivement les entiers à partir de 0 jusqu'à trouver l'entier x solution, ce qui peut demander n essais dans le pire des cas, un temps exponentiel en la taille de n.

Par division euclidienne par un entier m, x=im+j avec $0 \leq j < m$ et $0 \leq i \leq n/m$. On a alors que $\alpha^x = \beta$ si et seulement si $\alpha^j = \beta(\alpha^{-m})i$

L'algorithme baby-step giant-step utilise ceci pour un m bien choisi, le plus souvent m=$[\sqrt{n}]$: pour trouver l'entier x on calcule la liste des (j, αj) (les baby-steps), puis les (i, $\beta(\alpha^{-m})$i) (les giant-steps) jusqu'à trouver un second membre déjà présent dans la liste des baby-steps, le couple (i,j) correspondant donne x.

En prenant m = $[\sqrt{n}]$, l'algorithme demande au pire $2[\sqrt{n}]$ opérations de multiplications dans le groupe, contre n par recherche exhaustive, auquel il faut ajouter le temps nécessaire pour écrire la liste et chercher à un élément commun, sous une forme qui permet d'optimiser cette recherche, sachant qu'une opération de comparaison de deux éléments du groupe est de toute façon beaucoup moins coûteuse a priori qu'une multiplication. Une possibilité est d'utiliser un tableau trié sur le second membre pour la première liste, et de procéder par dichotomie pour la recherche dans celui-ci, on a alors $O(\sqrt{n}\log n)$ opérations de comparaison, une autre d'utiliser une table de hachage.

L'algorithme demande par contre de stocker $[\sqrt{n}]$ éléments du groupe ce qui peut s'avérer à son tour rédhibitoire pour des groupes de taille importante (typiquement ceux utilisés en cryptographie).

3.1.3 Algorithme

Entrée : un groupe cyclique G d'ordre n, ayant un générateur α et un élément β.
Sortie : une valeur x vérifiant $\alpha^x=\beta$.
m $\leftarrow [\sqrt{n}]+1$
Pour j tel que $0 \leq j < m$: //Baby-Step

Calculer α^j et sauvegarder la paire (j,α^j), par exemple dans une table de hachage.

Calculer α^{-m} (l'inverse modulaire de α^m mod n).
$\gamma \leftarrow \beta$.(Faire $\gamma = \beta$) Pour i tel que $0 \leq i < m$: //Giant-Step
Vérifier si γ est le second composant (α^j) d'une paire quelconque dans la table.
Si oui, retourner im + j.
Si non , $\gamma \leftarrow \alpha^{-m}$.

3.1.4 Complexité

L'algorithme requiert un espace $\mathcal{O}(\sqrt{n})$ en mémoire (pour le stockage des baby-steps), et $\mathcal{O}(\sqrt{n}$ log n) opérations de multiplication (une exponentiation par itération et **m** itérations)

3.1.5 Exemple

algorithme baby-step giant-step pour les logarithmes $\mathbb{Z}^*_{113}$

soit p=113 l'element α=3 est un génerateur de $\mathbb{Z}^*_{113}$ d'ordre n=112. On considère β=57. Alors $log_3 57$ est calculé de la maniére suivante :

1. m $\leftarrow [\sqrt{112}=11]$
2. Construire une table dont les entrées sont (j,α^j mod p) pour $0 \leq i < 11$:

j	0	1	2	3	4	5	6	7	8	9	10
3^j mod 113	1	3	9	27	81	17	51	40	7	21	63

et trier le tableau selon le deuxième composant :

j	0	1	2	3	4	5	6	7	8	9	10
3^j mod 113	1	3	7	9	17	21	27	40	51	63	81

3. en utilisant l'algorithme du calcul de l'inverse modulaire on a $\alpha^{-1} = 3^{-1}$mod 113 = 38, puis on calcule $\alpha^{-m} = 38^{11}$mod 113 = 58
4. Ensuite, $\gamma = \beta\alpha^{-mi}$ mod 113 pour i=0,1,2.... est calculé jusqu'à l'obtention d'une valeur dans la deuxième ligne de la table. Cela donne :

i	0	1	2	3	4	5	6	7	8	9
$\gamma = 57\times58^i$ mod 113	57	29	100	37	112	55	26	39	2	3

Finalement, puisque $\beta\alpha^{-9m} = 3 = \alpha^1$, $\beta = \alpha^{100}$. Et Donc $log_3 57 = 100$.

3.2 Algorithme rho de Pollard pour le logarithme discret

3.2.1 Introduction

L'algorithme rho de Pollard a été introduit par John M. Pollard en 1978 pour résoudre le problème du logarithme discret. Il est analogue à l'algorithme rho de Pollard que le même auteur avait introduit pour résoudre le problème factorisation entière.
C'est un algorithme randomisé ayant le même temps d'exécution attendu que l'algorithme baby-step giant-step, mais nécessite un espace mémoire négligeable. Pour cette raison, l'algorithme de Shanks est de loin préférable pour les problèmes d'intérêt pratique.

3.2.2 Théorie

Pour simplifier, nous supposons dans cette sous-section que G est un groupe cyclique dont l'ordre n est premier. Le groupe G est divisé en trois ensembles S_1, S_2 et S_3 de taille à peu près égale sur la base de propriétés faciles à tester. Il faut être prudent dans la sélection de la partition. par exemple, $1 \notin S_2$. On définit une séquence d'éléments de groupe x_0,x_1,x_2... par x_0=1 et

$$x_{i+1} = \mathrm{f}(x_i) = \begin{cases} \beta\times\ x_i, & \text{si } \ x_i \in S_1, \\ \mathrm{x}_i^2, & \text{si } \ x_i \in S_2, \\ \alpha\times\ x_i, & \text{si } \ x_i \in S_3, \end{cases} \tag{3.2}$$

pour i ≥ 0. Cette séquence d'éléments de groupe définit à son tour deux séquences d'entiers a_0,a_1,a_2... et b_0,b_1,b_2... satisfaisant $x_i = \alpha^{ai}\beta^{bi}$ pour i ≥ 0 : $a_0 = 0$, $b_0 = 0$, et pour i ≥ 0,

$$a_{i+1} = \begin{cases} \mathrm{a}_i, & \text{si } \ x_i \in S_1, \\ 2\mathrm{a}_i \text{ mod n}, & \text{si } \ x_i \in S_2, \\ \mathrm{a}_i{+}1 \text{ mod n}, & \text{si } \ x_i \in S_3, \end{cases} \tag{3.3}$$

et

$$b_{i+1} = \begin{cases} \text{b}_i\text{+1 mod n}, x_i, & \text{si } x_i \in S_1, \\ 2\text{b}_i \text{ mod n}, & \text{si } x_i \in S_2, \\ \text{b}_i, & \text{si } x_i \in S_3, \end{cases} \tag{3.4}$$

L'algorithme de Floyd cycle-finding peut être utilisé pour trouver deux éléments de groupe x_i et x_{2i} tel que $x_i = x_{2i}$. Donc $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$, et donc $\beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i}$.
on fait le logarithme à base α des deux côtés de cette dernière équation donne :

$$(b_i \text{ - } b_{2i}) log_\alpha \beta \equiv (a_{2i} \text{ - } a_i) (\text{mod n})$$

Lorsque $b_i \not\equiv b_{2i}$ ($b_i \equiv b_{2i}$ se produit avec une probabilité négligeable) cette équation peut alors être résolue efficacement pour déterminer $log_\alpha \beta$.

3.2.3 Algorithme

— **Entrée :** un groupe cyclique G d'ordre premier n, ayant un générateur α et un élément β.
— **Sortie :** le logarithme discret x = $log_\alpha \beta$.

1. $x_0 \leftarrow 1$, $a_0 \leftarrow 0$, $b_0 \leftarrow 0$.
2. pour i=1,2,.... faire
   (a) en utilisant les x_{i-1},a_{i-1},b_{i-1} et x_{2i-2},a_{2i-2},b_{2i-2} calculés précédemment, on calcul x_i,a_i,b_i et x_{2i},a_{2i},b_{2i} en utilisant les équations (3.2),(3.3)et (3.4)
   (b) si x_i=x_{2i} alors
   r $\leftarrow$ b_i - b_{2i} mod n
   si r = 0 alors terminer l'algorithme avec échec ;
   sinon, calculer x=$r^{-1}(a_{2i}$ - $a_i)$ mod n
   returner(x)

Dans le cas rare où l'algorithme se termine avec un échec, la procédure peut être répétée en sélectionnant des entiers aléatoires a_0, b_0 dans l'intervale [1,n-1], et commençant par $x_0 = \alpha^{a_0}\beta^{b_0}$

3.2.4 Compléxité

Le temps d'exécution moyen est en $\mathcal{O}(\sqrt{n})$. Si cet algorithme est utilisé avec l'algorithme de Pohlig-Hellman, le temps d'exécution des deux algorithmes combinés est en $\mathcal{O}(\sqrt{n})$, où p est le plus grand facteur premier de n.

Exemple

algorithme rho Pollard pour les logarithmes $\mathbb{Z}^*_{383}$

soit l'élement $\alpha = 2$ un génerateur du sous-groupe G de $\mathbb{Z}^*_{383}$ d'ordre n=191. On considère β=228.
On répartie alors les élements de G en trois sous-ensembles selon la règle x $\in S_1$ si x $\equiv 1$ (mod 3), x $\in S_2$ si x $\equiv 0$ (mod 3), et x $\in S_3$ si x $\equiv 2$ (mod 3).la table ci-dessous montre les valeurs de x_i, a_i, b_i, x_{2i}, a_{2i}, et b_{2i},
à la fin de chaque itération de l'étape 2 de l'algorithme on a $x_{14} = x_{28} = 144$. Finalement on calcule r = b_{14} - b_{28} mod 191 =125, $r^{-1} = 125^{-1}$ mod 191 =136, et $r^{-1}(a_{28}$ - $a_{14})$ mod 191 =110 .Donc $log_2 228 = 110$.

i	x_i	a_i	b_i	x_{2i}	a_{2i}	b_{2i}
1	228	0	1	279	0	2
2	279	0	2	184	1	4
3	92	0	4	14	1	6
4	184	1	4	256	2	7
5	205	1	5	304	3	8
6	14	1	6	121	6	18
7	28	2	6	144	12	38
8	256	2	7	235	48	152
9	152	2	8	72	48	154
10	304	3	8	14	96	118
11	372	3	9	256	97	119
12	121	6	18	304	98	120
13	12	6	19	121	5	51
14	144	12	38	144	10	104

3.3 Algorithme pohlig-Hellman

3.3.1 Introduction

L'algorithme pohlig-Hellman pour le calcul des logarithmes tire parti de la factorisation d'ordre n du groupe G.
soit n = $p_1^{e_1} p_2^{e_2} .... p_r^{e_r}$ la factorisation en nombres premiers de n. si x = $log_\alpha \beta$, alors il faut déterminer x_i = x mod $p_i^{e_i}$ pour 1 $\leq$ i $\leq$ r, et ultiliser l'algorithme de Gauss pour trouver x mod n. chaque entier x_i est déterminé par le calcul des chiffres l_0,l_1,.....l_{e_i-1} avec x_i = $l_0 + l_1 p_1 +$ ..... $+ l_{e_i-1} p_i^{e_i-1}$ et $0 \leq l_j \leq p_i$ - 1

3.3.2 Algorithme

— **Entrée :** un groupe cyclique G d'ordre premier n, ayant un générateur α et un élément $\beta \in$ G.
— **Sortie :** le logarithme discret x = $log_\alpha \beta$.

1. trouver la factorisation en nombres premiers de n : n = $p_1^{e_1} p_2^{e_2} .... p_r^{e_r}$, avec $e_i \geq$ 1.

2. pour i de 1 à n faire :
   (calculer $x_i = l_0 + l_1 p_1 + \ldots\ldots + l_{e_i-1} p_i^{e_i-1}$, avec x_i = x mod $p_i^{e_i}$.)
   (a) (pour simplifier) q $\leftarrow p_i$, et e $\leftarrow e_i$.
   (b) $\gamma \leftarrow 1$ et $l_{-1} \leftarrow 0$
   (c) calculer $\overline{\alpha} \leftarrow \alpha^{n/q}$.
   (d) (calculer l_j) pour j de 0 à $_e$ - 1 faire :
   calculer $\gamma \leftarrow \gamma\alpha^{l_{j-1}q^{j-1}}$ et $\overline{\beta} \leftarrow (\beta\gamma^{-1})^{n/q^{j+1}}$.
   calculer $l_j \leftarrow log_{\overline{\alpha}}\overline{\beta}$
   (e) $x_i \leftarrow l_0 + l_1 q + \ldots\ldots + l_{e-1} q^{e-1}$.
3. utiliser l'algorithme de Gauss pour calculer l'entier x, 0 $\leq$ x $\leq$ n - 1, tel que x $\equiv x_i$ (mod $p_i^{e_i}$) pour 1 $\leq$ i $\leq$ r.
4. retourner(x).

3.3.3 Exemple

algorithme pohlig-Hellman pour les logarithmes $\mathbb{Z}_{251}^*$

soit p=251 l'element α=71 est un génerateur de $\mathbb{Z}_{251}^*$ d'ordre n=250. On considère β=210. Alors x = $log_{71}57$ est calculé de la maniére suivante :

1. la factorisation en nombres premiers de n est $250 = 2 \times 5^3$.
2. (a) (calculer x_1 = x mod 2)
   calculer $\overline{\alpha} = \alpha^{n/2}$ mod p = 250 et $\overline{\beta} = \beta^{n/2}$ mod p = 250. donc $x_1 = log_{250} = 1$.

   (b) (calculer x_2 = x mod $5^3 = l_0 + l_1 5 + l_2 5^2$)

      i. calculer $\overline{\alpha} = \alpha^{n/5}$ mod p = 20.
      ii. calculer $\gamma = 1$ et $\overline{\beta} = (\beta\gamma^{-1})^{n/5}$ mod p = 149. En utilisant la recherche exaustive, calculer $l_0 = log_{20}149 = 2$.
      iii. calculer $\gamma = \gamma\alpha^2$ mod p = 21 et $\overline{\beta} = (\beta\gamma^{-1})^{n/25}$ mod p = 113.En utilisant la recherche exaustive, calculer $l_1 = log_{20}113 = 4$.
      iv. calculer $\gamma = \gamma\alpha^{4.5}$ mod p = 115 et $\overline{\beta} = (\beta\gamma^{-1})^{(p-1)/125}$ mod p = 149.En utilisant la recherche exaustive, calculer $l_2 = log_{20}149 = 2$. Donc, $x_2 = 2 + 4 \times 5 + 2 \times 5^2 = 72$.

   (c) Finalement,résoudre la paire de congruence x $\equiv$ 1 (mod 2),x $\equiv$ 72 (mod 125) pour trouver x = $log_{71}210 = 197$.

3.3.4 complexité

Pour une factorisqtion n donnée le temps d'execution de Pohlig-Hellman est O($\sum_{i=1}^{r} e_i(logn + \sqrt{p_i})$) groupe de multiplications.

3.4 Algorithme Index-calculus

3.4.1 Introduction

Bien que la méthode du calcul d'indice se montre aussi performante pour factoriser de grands entiers, nous nous intéresserons uniquement au cas des algorithmes de cette famille qui permettent de résoudre le problème du logarithme discret, encore une fois dans un groupe cyclique G engendré par g. Les algorithmes de cette famille s'articulent autour de plusieurs phases :

3.4.2 Algorithme

— **Entrée :** un groupe cyclique G d'ordre premier n, ayant un générateur α et un élément β.
— **Sortie :** le logarithme discret y = $log_\alpha \beta$.

1. Sélectionnez une base de facteurs S=$\{p_1, p_2, p_3, ..., p_t\}$ pour qu'un nombre significatif d'éléments de puisse être efficacement exprimé en tant que produits d'éléments.
2. (Recueillir des relations linéaires impliquant des logarithmes d'éléments dans S)
   (a) Sélectionnez un entier aléatoire k, $0 \leq k \leq n-1$, et calculer α^k
   (b) Essayez d'écrire α^k comme un produit d'éléments de S

$$\alpha^k = \prod_{i=1}^{t} p_i^{c_i} \ , c_i \geq 0$$

pour chaque k . Alors,

$$k \equiv \sum_{i=1}^{t} c_i log_\alpha p_i \ (mod\ n) \qquad (*)$$

   (c) Répétez les étapes (a) et (b) jusqu'à obtenir $t + c$ relations de la forme $(*)$.

3. (Trouvez les logarithmes des éléments dans S), Travail modulo n, Résoudre le système linéaire de $t + c$ équations de la forme $(*)$ collectée à l'étape 2 pour obtenir les valeurs de $log_\alpha p_i$,$1 \leq i \leq t$.
4. (calcul de y)
   (a) Sélectionnez un entier aléatoire k, $0 \leq k \leq n-1$, et calculer $\beta.\alpha^k$.
   (b) Ecrire $\beta.\alpha^k$ comme produit d'éléments de S :
   $$\beta.\alpha^k = \prod_{i=1}^{t} p_i^{d_i} \ ,d_i \geq 0 \qquad (**)$$

   si l'attent n'aboutit pas, répétez l'étape $(**)$,
   sinon, en prenant les logarithmes des deux côtés de l'équation $(**)$, on obtient
   $$\log_\alpha \beta = (\sum_{i=1}^{t} d_i log_\alpha p_i) - k$$

   calculer $y = (\sum_{i=1}^{t} d_i log_\alpha p_i) - k$

3.4.3 Exemple de l'algorithme Index-calculus dans $\mathbb{Z}_p^*$

Pour $\mathbb{Z}_p^*$, p est premier, le facteur de base S peut être choisie comme les premiers nombres premiers t. La relation est obtenue en calculant α^k mod p et en utilisant la division successive afin de vérifier si cet entier est un produit des nombres premiers dans S.L'exemple suivant illustre l'algorithme Index-calculus dans $\mathbb{Z}_p^*$ pour un probléme avec des paramétres de petite taille.

algorithme Index-calculus pour les logarithmes $\mathbb{Z}_{229}^*$

soit p = 229. L'élement $\alpha = 6$ un génerateur de $\mathbb{Z}_{229}^*$ d'ordre n=228. On considère β=13. Alors $log_6 13$ est calculé avec l'algorithme Index-calculus de la maniére suivante :

1. Le facteur de base est choisit comme les 5 premiers nombres premiers : S = {2,3,5,7,11}.
2. les 6 relations obtenues contenants les éléments du facteur de base sont :

$$6^{100} \bmod 229 = 180 = 2^2 \times 3^2 \times 5$$
$$6^{18} \bmod 229 = 176 = 2^4 \times 11$$
$$6^{12} \bmod 229 = 165 = 3 \times 5 \times 11$$
$$6^{62} \bmod 229 = 154 = 2 \times 7 \times 11$$
$$6^{143} \bmod 229 = 198 = 2 \times 3^2 \times 11$$
$$6^{206} \bmod 229 = 210 = 2 \times 3 \times 5 \times 7$$

Ces relations engendrent les équations suivantes :

$$\begin{aligned} 100 \equiv & 2log_6 2 + 2log_6 3 + log_6 5 \pmod{228} \\ 18 \equiv & 4log_6 2 + log_6 11 \pmod{228} \\ 12 \equiv & log_6 3 + log_6 5 + log_6 11 \pmod{228} \\ 62 \equiv & log_6 2 + log_6 7 + log_6 11 \pmod{228} \\ 143 \equiv & log_6 2 + 2log_6 3 + log_6 11 \pmod{228} \\ 206 \equiv & log_6 2 + log_6 3 + log_6 5 + log_6 7 \pmod{228} \end{aligned}$$

3. Résoudre le système linéaire de six équations à cinq inconnues(les logarithmes $x_i = log_6 p_i$) donne les solutions $log_6 2 = 21$, $log_6 3 = 208$,$log_6 5 = 98$, $log_6 7 = 107$, et $log_6 3 = 208$,
4. Pour $k = 77$, $\beta.\alpha^k = 13.6^{77} \bmod 229 = 147 = 3.7^2$,
ensuite $\log_6 13 = (\log_6 3 + 2\log_6 7 - 77) \pmod{228} = 117$

3.4.4 Complexité

En supposant une sélection optimale de la base de facteurs, le temps d'exécution attendu (en utilisant la notation L) de l'algorithme de calcul d'indice peut être défini comme $L_n[1/2, \sqrt{2} + o(1)]$

Chapitre 4

Sécurité El Gamal

4.1 Face aux attaques à texte clair choisi

En observant les informations publiques : g^x, g^s ; on se rend compte que seuls des éléments de G sont rendus visibles et non pas les exposants (ici x et s respectivement). Informellement on peut remarquer que le problème du logarithme discret peut s'interpréter comme le fait qu'il est difficile de retrouver les informations secrètes (sk =$\log_g$(h) par exemple) qui permettraient de retrouver le message.

De manière plus précise, c'est le problème de décision de Diffie-Hellmann (ou DDH) qui permet de garantir la sécurité sémantique du schéma.

Démonstration

En supposant qu'on ait un adversaire $\mathcal{A}$ contre la sécurité sémantique du chiffrement ElGamal qui gagne avec une probabilité non négligeable ϵ. Il devient alors possible de construire un adversaire $\mathcal{B}$ contre DDH, ce qui contredirait la difficulté supposée de ce problème. Cette réduction que l'on vient de décrire va former la preuve de sécurité du schéma.

Pour construire cet adversaire, qui sera appelé dans la suite «la réduction», on suppose qu'il reçoive un triplet DDH : (g^a, g^b, g^c) , son but est alors de décider si c=a· b ou si c$\in_R \mathbb{Z}_p$ avec une probabilité non négligeable. Pour cela il dispose d'une interface avec l'adversaire décrit plus haut face à la sécurité sémantique du cryptosystème ElGamal.

La réduction va donc envoyer à l'adversaire contre ElGamal la clef publique pk=(G,q,g,h=g^a). Au moment de produire le challenge pour l'adversaire contre la sécurité sémantique du cryptosystème, la réduction va envoyer comme chiffré : C=$(g^b, m_i \cdot g^c)$ pour i$\in \{0,1\}$ de son choix. Si jamais le triplet est un triplet DDH, c'est-à-dire si c=a· b, alors C=$(g^b, m_i \cdot (g^a)^b)$=$(g^b, m_i \cdot h^b)$ serait formé comme un chiffré valide pour ElGamal au regard de la clef publique pk. En revanche, si l'exposant c est aléatoire, alors l'adversaire contre ElGamal ne sera pas en mesure de distinguer les deux messages du challenge autrement qu'en répondant au hasard.

On n'a plus qu'à répondre «1» (correspondant au fait que le challenger pour DDH a envoyé un triplet DDH) si jamais notre adversaire réussit, et «0» (correspondant au fait

que le challenger pour DDH a envoyé un triplet aléatoire) dans le cas contraire.

Analyse

On va désormais borner l'avantage de gagner de notre réduction à partir de l'avantage ϵ de l'adversaire supposé contre notre schéma.

On commence par remarquer que l'on a deux cas : soit le challenge envoyé par notre challenger est un vrai triplet DDH, soit il s'agit d'un triplet tiré uniformément.

$$\begin{aligned}
\mathrm{Adv}^{DDH}(\mathcal{B}) &= |\Pr[\mathcal{B} \to 1 \mid \mathrm{DDH}] - \Pr[\mathcal{B} \to 1 \mid \mathrm{Aleatoire}]| \\
&= |\Pr[\mathcal{A} \to i \mid \mathrm{DDH}] - \Pr[\mathcal{A} \to i \mid \mathrm{Aleatoire}]| \\
&= |\Pr[\mathcal{A} \to 0 \mid i = 0 \wedge \mathrm{DDH}]\Pr[i = 0] + \Pr[\mathcal{A} \to 1 \mid i = 1 \wedge \mathrm{DDH}]\Pr[i = 1] - \frac{1}{2}| \\
&= |\frac{1}{2}(1 - \Pr[\mathcal{A} \to 1 \mid i = 0 \wedge \mathrm{DDH}] + \Pr[\mathcal{A} \to 1 \mid i = 1 \wedge \mathrm{DDH}]) - \frac{1}{2}| \\
&= \frac{1}{2}| - \Pr[\mathcal{A} \to 1 \mid i = 0 \wedge \mathrm{DDH}] + \Pr[\mathcal{A} \to 1 \mid i = 1 \wedge \mathrm{DDH}]| \\
&= \frac{1}{2}\mathrm{Adv}^{\mathrm{ElGamal}}(\mathcal{A}) \\
&= \frac{\varepsilon}{2}
\end{aligned}$$

Ainsi on a un avantage qui reste non négligeable : pour atteindre la même sécurité entre DDH et notre cryptosystème, il suffit que le problème DDH reste sûr avec un bit de sécurité supplémentaire.

4.2 Face aux attaques à chiffré choisi

Dans des modèles avec un attaquant possédant plus de puissance, comme sous attaques à chiffrés choisis, le cryptosystème d'ElGamal n'est pas sûr en raison de sa malléabilité ; en effet, étant donné un chiffré C=(c_1, c_2) pour le message m, on peut construire le chiffré $C^{'}$=$(c_1$,2$\cdot$ $c_2)$, qui sera valide pour le message 2$\cdot$ m.

Cette malléabilité (il s'agit d'un homomorphisme multiplicatif) en revanche permet de l'utiliser pour le vote électronique par exemple.

Il existe cependant des variantes qui atteignent la sécurité face aux attaques à chiffrés choisis, comme le cryptosystème de Cramer-Shoup qui peut être vu comme une extension du chiffrement ElGamal.

4.2.1 Exemple

Pour l'exemple, on peut prendre le groupe G=$(\mathbb{Z}$ /100000007$\mathbb{Z}^*, \times)$, avec comme générateur g=5 (Mise en garde : ce n'est pas un groupe sûr, ces valeurs ont été prises

uniquement pour produire un exemple simple).

Une clef publique possible pourrait donc être : pk=(G,100000006,5,29487234), et comme clef secrète sk=81996614.

On remarquera que comme le chiffrement est un algorithme probabiliste, il y a différentes sorties possibles pour le même message. Un chiffré possible pour le message 42 pourrait donc être (58086963, 94768547), mais aussi (83036959, 79165157) pour les aléas r valant 6689644 et 83573058 respectivement.

Néanmoins, si on fait le calcul $\frac{c_2}{c_1^{sk}}$ pour nos deux chiffrés, on obtiendra bien 42 en sortie.

Conclusion

L'atout principal de la cryptographie à clé publique (chiffrement asymétrique) réside dans la facilité de gestion du parc des clés des utilisateurs. En effet, l'augmentation du nombre d'utilisateurs ne complexifie pas le protocole. De plus, l'arrivée de nouveaux utilisateurs et leur intégration demande très peu d'efforts et ne modifie en rien les paramètres des autres. Ainsi, la cryptographie à clé publique résout le problème de distribution des clés que l'on peut rencontrer dans la cryptographie à clé privée. Toutefois, les techniques asymétriques souffrent de leur grande lenteur. Chiffrer un message est 100 à 1000 fois plus long que certaines techniques symétriques.

Webographie

[1] Article par Phillipe Huysmans : https ://www.levilainpetitcanard.be/cryptographie-partie-ii-cryptographie-classique/

[2] Apprendre en ligne : https ://www.apprendre-en-ligne.net/crypto/bibliotheque/

[3] Aspect Technique du chiffrement :http ://nopb.chez.com/crypto2.html

[4] Article par Pierrick Gaudry :
https ://www.enseignement.polytechnique.fr/profs/informatique/Georges.Gonthier/pi2000/pro/gaudry/index.html

[5] Wikipedia : https ://fr.wikipedia.org

[6] github : https ://github.com/mcerovic/PohligHellman/blob/master/

[7] https ://rosettacode.org/wiki/Tonelli-Shanks_algorithm

[8] stackoverflow : https ://stackoverflow.com/

Bibliographie

[1] La Cryptographie de la théorie à la pratique .Pr. Mohamed El Marraki. Faculté des sciences Rabat.

[2] Arithmétique et Cryptographie, Partie I Arithmétique A. Bonnecaze R. Rolland Institut de Mathématiques de Luminy (IML).

[3] Chiffrement,théorie de l'information et applications, Notes de complément au cours . JL Roch.

[4] Cryptographie et sécurité Yves GERARD Université Lyon 1.

[5] Histoire de la cryptographie Fréedéric Havet MASCOTTE, commun I3S(CNRS/UNSA)-INRIA Sophia Antipolis.

[6] Le logarithme discret dans les corps finis.Cecile Pierrot.Université Pierre et Marie Curie..

[7] Andreea Dragut (dragut@univmed.fr) Cours de cryptographie Chapitre IV.

[8] Handbook of Applied Cryptography de Alfred J. Menezes (Auteur), Paul C. van Oorschot (Auteur), Scott A. Vanstone (Auteur).

[9] Chapitre 3 La cryptographie classique par R. Dumont

[10] Cours de Cryptanalyse : Guilhem Castagnos.