Rapport de Projet : ElGamal

GERTNER Estelle
HUAUT Antonin
LAHOUGUE Ludovic
LE ROUX Clémence

Décembre 2019
DUT Informatique
IUT Grand Ouest Normandie
Université de Caen

Table des matières

1 Introduction

Le protocole ElGamal est un protocole de cryptographie asymétrique créé en 1984 par Taher ElGamal. Il est asymétrique car on utilise deux éléments distincts pour chiffrer et déchiffrer le message : une clé publique et une clé privée. Le protocole ElGamal a été créé pour répondre à un besoin : sécuriser les communications entre deux machines distinctes.

2 Nombres premiers

2.1 Qu'est-ce qu'un nombre premier ?

Un nombre premier est un nombre divisible uniquement par 1 et par lui même. Il en existe une infinité. Aujourd'hui, le plus grand nombre premier est $2^{82589933} - 1$, il a été déterminé grâce à la méthode des nombres premiers de Mersenne et il possède plus de 24 millions de chiffres en écriture décimale. Trouver un nombre premier peut prendre beaucoup de temps à une machine : plus le nombre est grand, plus le nombre d'étapes est grand, plus la machine va mettre de temps à calculer.

2.2 ElGamal et les nombres premiers

Pour assurer la sécurité du chiffrement ElGamal, il faut aujourd'hui choisir un nombre premier p de l'ordre de 300 chiffres ainsi que deux nombres a et b de l'ordre de 100 chiffres. La clé doit être composée d'un nombre premier p très grand car elle devient ainsi très complexe à craquer : il est très difficile voire impossible de retrouver le nombre premier choisi pour chiffrer. Comme la complexité du cryptosystème ElGamal est liée au logarithme discret, aucun algorithme polynomial ne permet aujourd'hui de craquer la clé. Si un algorithme permettant de résoudre un problème lié à la complexité algorithmique est créé un jour, ElGamal pourrait tomber.

3 ElGamal

3.1 Comment ça marche ?

3.1.1 Créer une clé

Premièrement, on choisit un nombre premier p. Ensuite, on choisit deux entiers, le premier compris entre 0 et $p-2$ et le deuxième compris entre 0 et $p-1$. On résout l'équation $n \equiv m^a[p]$ pour obtenir le "dernier morceau" la clé publique sera (p, m, n).

3.1.2 Chiffrer un message

Pour chiffrer le message on récupère la clé publique (p, m, n) de la personne à qui on veut envoyer le message. On transforme les lettres du message en nombres (avec le placement des lettres dans l'alphabet par exemple mais ce n'est pas très conseillé car peu sécurisé).
On divise la suite de chiffres en blocs de telle sorte que la valeur de chaque bloc soit inférieure à p. On choisit aléatoirement un entier k entre 0 et $p-1$. Pour chiffrer chaque bloc x on calcule y_1 et y_2 avec $y_1 \equiv m^k[p]$ et $y_2 \equiv xn^k[p]$. Le bloc x est donc chiffré avec le couple (y_1, y_2).

3.1.3 Déchiffrer un message

Pour déchiffrer le message on a besoin de la clé publique et de la clé privée. Pour trouver le bloc x chiffré précédemment il faut résoudre $x = y_2 \times y_1^{p-1-a}[p]$, démonstration :

- On a chiffré le bloc avec les formules $y_1 \equiv m^k[p]$ et $y_2 \equiv x \times n^k[p]$
- Avec la clé publique et la clé privée on a $n \equiv m^a[p]$

- On remplace les termes que l'on connaît, on simplifie :

$$\begin{aligned} y_1^{p-1-a} \times y_2 &\equiv (m^k)^{p-1-a} x n^k [p] \\ &\equiv m^{k(p-1-a)} x m^{ka} [p] \\ &\equiv m^{k(p-1)} x [p] \end{aligned}$$

d'après le petit Théorème de Fermat on a :

$$\begin{aligned} m^{k(p1)} x[p] &\equiv (\underbrace{m^{p-1}}_{1})^k x[p] \\ &\equiv x[p] \end{aligned}$$

On a donc bien $x = y_2 \times y_1^{p-1-a}[p]$ à résoudre pour trouver le message.

Le petit Théorème de Fermat :
Soit p un nombre premier, a un entier naturel non divisible par p et $k \in [1; p-1]$.

[1] Montrons par l'absurde que $\frac{k \times a}{p} \neq 0$:
Posons $\frac{k \times a}{p} = 0$. Or p ne divise pas a.
Donc $\frac{k}{p} = 0$, mais p est premier, donc c'est impossible !

Soit $k' \in [1; p-1]$ et $k' \neq k$

[2] Montrons par l'absurde qu'avec n'importe quel entier n, $\frac{n \times a}{p} \neq 0$:
Supposons que $k \times a = k' \times a[p]$.

$$\begin{aligned} &\Leftrightarrow k \times a - k' \times a = 0[p] \\ &\Leftrightarrow a \times (k - k') = 0[p] \end{aligned}$$

Or p ne divise pas a :

$$\Leftrightarrow (k - k') = 0[p]$$

Or $|k - k'| < p - 2$ car on a généré k inférieur à $p - 1$
Donc p ne divise pas $k - k'$

[3] D'après [1] et [2] : $a, 2 \times a, 3 \times a, ...(p-1) \times a$ ont un reste après division par $p \neq 0$

Donc les restes valent (dans le désordre) : $1, 2, 3, ..., (p-1)$

Cela signifie que :

$$\begin{aligned} &\Leftrightarrow a \times 2a \times ... \times a(p-1) \equiv 1 \times 2 \times ... \times (p-1)[p] \\ &\Leftrightarrow a^{p-1} \times (p-1)! \equiv (p-1)![p] \\ &\Leftrightarrow (a^{p-1} - 1) \times (p-1)! \equiv 0[p] \end{aligned}$$

Or p est premier avec $(p-1)!$, donc :

$$\begin{aligned} &\Leftrightarrow a^{p-1} - 1 \equiv 0[p] \\ &\Leftrightarrow a^{p-1} \equiv 1[p] \end{aligned}$$

4 Démonstration

4.1 Algorithmes de génération de nombres premiers

4.1.1 Méthode naïve

Soit n un entier aléatoire impair dont on va vérifier la primalité. Si n est divisible par un $i \in \{2, 3, ..., \sqrt{n}\}$ alors n est composé et on prend un nouveau n aléatoire jusqu'à ce que n ne soit plus divisible.

Cet algorithme devient long pour de grands nombres car on fait beaucoup de tours de boucle. Par exemple, on génère avec cet algorithme des nombres premiers d'ordre de grandeur :
10^7 en 1,3 ms
10^{12} en 380 ms
10^{14} en 4000 ms

Pour utiliser ElGamal, nous avons besoin de générer des nombres premiers beaucoup plus grands. Il faut donc utiliser une autre méthode.

4.1.2 Test de primalité de Miller-Rabin

Soit n un entier aléatoire impair dont on va vérifier la primalité. Soit a un entier qui n'est pas divisible par n, appelé témoin de n.
Le test de primalité de Miller-Rabin est un test probabiliste. Nous pouvons cependant réduire autant que l'on souhaite la probabilité que n soit composé en faisant passer le test à n pour un nombre k de témoins a. La probabilité que n soit composé est alors de 4^{-k}.
Le test de primalité de Miller-Rabin utilise une propriété de l'entier n, dépendant d'un témoin a, qui est vraie si n est un nombre premier.

Propriété : Soit s et d deux entiers vérifiant $p - 1 = 2^s \times d$.
Si p est premier, alors $\forall a,\ a^d \equiv 1\,[p]$ *ou* $\exists r \in \{0, 1, ..., s-1\}\ a^{2^r d} \equiv 1\,[p]$
Si cette propriété n'est pas vérifiée, on prend un nouveau n aléatoire jusqu'à ce que n passe le test avec ses k témoins a.

Le test de Miller-Rabin est beaucoup plus rapide que la méthode précédente. Par exemple, on génère avec cet algorithme des nombres premiers d'ordre de grandeur :
10^{14} en 0.67 ms pour k = 80 témoins
10^{100} en 117 ms pour k = 80 témoins
10^{1000} en 361 s pour k = 1 témoin (probabilité de 0.75 d'être premier)

Exemple d'utilisation de la propriété :

Prenons un entier $n = 661$
Nous devons trouver s et d tel que $p - 1 = 2^s \times d$
Pour cela, initialisons $s = 0$ *et* $d = n - 1 = 6$

Tant que $d \equiv 0\,[2]$, $d = d//2$ et $s = s + 1$.

$660 \equiv 0\,[2]$
$d = 660//2 = 330$
$s = 0 + 1 = 1$

$330 \equiv 0\,[2]$
$d = 330//2 = 165$
$s = 1 + 1 = 2$

$165 \equiv 1\,[2]$

$d = 165$ et $s = 2$.

Prenons un entier témoin $a = 28$ et $r = 1$

$28^{165} \equiv 192\,[661]$
$28^{2^1 165} \equiv 1\,[661]$

Donc 661 est premier.

4.1.3 Test de primalité AKS

Nous n'avons pas créé d'algorithme pour ce test car le test de Miller-Rabin suffit pour utiliser ElGamal.

Le test de primalité Agrawal-Kayal-Saxena est un test primalité déterministe et généraliste. La particularité de ce test est qu'il ne repose pas sur hypothèse non démontrée, contrairement aux autres tests de primalité déterministes. Des variantes de ce test sont apparues plus tard comme Lenstra et Pomerance (2002) ou encore Berrizbeitia (2003), qui améliorent la vitesse d'exécution.

Principe :

Pour tout entier n et tout entier a premier avec n, n est premier si et seulement si $(X + a)^n \equiv X^n + a\,[n]$.

4.2 Chiffrer

Imaginons que Bob souhaite envoyer le message “DUT” chiffré avec l’algorithme d’ElGamal à Alice

Nous imaginons qu’Alice et Bob vivent dans un monde où il n’existe que trois lettres “D”, “U” et “T”. On leur associe les valeurs :

Lettre	Valeur
D	10
U	11
T	12

Alice : Génération d’une paire de clés
Soit p, un nombre premier, $p \geq 10^{300}$
Soit a, $10^{100} \leq a \leq p2$
Soit m, $10^{100} \leq m \leq p1$
Soit n, $n \equiv m^a \mod n$
$p, m, n, a \, \epsilon \, N$

Formant une clé publique (p, m, n) donc par exemple (661, 23, 566)
Et formant une clé privée (a) donc par exemple (7)

4.2.1 Récupérer la clé publique

Dans un premier temps, Bob va récupérer la clé publique d’Alice, qui va lui permettre de chiffrer son message.

4.2.2 Convertir le message

Ensuite, il convertit son message en chaîne de chiffres, suivant le tableau associatif.
Il obtient “10 11 12”

4.2.3 Séparer le message

Il sépare sa chaîne de chiffres par bloc de 3 (il complète avec un/des 0 tant que son nombre de chiffres n’est pas multiple de 3)
Il obtient (101, 112) (Note : chaque bloc doit être inférieur à p)

4.2.4 Chiffrer le message

Le chiffrement s'effectue bloc par bloc.
Nous allons transformer chaque bloc en double bloc $(x) => (y, z)$

Pour chaque bloc, on choisit un nombre k, $k \epsilon N^+, k \leq p1$

1^{er} bloc : $b = 101$
Prenons $k = 13$
L'objectif est de transformer chaque bloc en bloc du type (y_1, y_2)

- $y_1 = m^k[p] = 23^{13}[661] = 105$
- $y_2 = b \times m^k[p] = 101 \times 566^{13}[661] = 135$

Donc on remplace le bloc 101 par le bloc $(105; 135)$

$2^{\text{ème}}$ bloc : $b = 112$
Prenons $k = 275$

- $y_1 = 23^{275}[661] = 309$
- $y_2 = 112 \times 566^{275}[661] = 425$

Donc on remplace le bloc 112 par le bloc $(309; 425)$

Il obtient alors $((105; 135), (309; 425))$
Bob obtient comme message chiffré "105 135 309 425".
Il envoie son message à Alice.

4.3 Déchiffrer

Alice reçoit le message et souhaite le déchiffrer.
Elle va avoir besoin de sa clé privée et de sa clé publique.

4.3.1 Reconstruire le message en bloc de bloc

Alice reprend le message et le transforme en $((105; 135), (309; 425))$

4.3.2 Déchiffrer le message

Le déchiffrage se fait bloc par bloc

1^{er} bloc : $b = (105; 135)$
L'objectif est de transformer chaque bloc en bloc du type $(nombre)$

- $puissance = p - 1 - a = 661 - 1 - 7 = 653$
- $nombre = y_2 \times y_1^{puissance}[p] = 135 \times 105^{653}[661] = 101$

Le bloc (re)devient 101

$2^{ème}$ bloc : $b = (309; 425)$
L'objectif est de transformer chaque bloc en bloc du type $(nombre)$

- $puissance = 66117 = 653$
- $nombre = 425 \times 309^{653}[661] = 112$

Le bloc (re)devient 112

Nous obtenons la liste de bloc $(101; 112)$

4.3.3 Recomposer le message

On fusionne chaque liste de blocs pour former une chaîne de chiffres
Soit le message “101112”

4.3.4 Convertir le message

En prenant les nombres 2 à 2 (car notre alphabet de 3 lettres se base sur des nombres de 2 à 2), on récupère leur équivalent dans le tableau associatif.

Nous obtenons le message “10 11 12”
Et après conversion, nous retrouvons notre message original “DUT”

4.4 Exponentiation modulaire

L'exponentiation modulaire permet de réaliser des calculs du type $m^k[p]$ en évitant de s'encombrer de gros nombres.
Dans un premier temps, décomposons notre puissance suivant les puissances de 2 :
$23^{13} = 23^{8+4+1} = 23^{2^3+2^2+2^0} = 23^{2^3} + 23^{2^2} + 23^{2^0} = ((23^2)^2)^2 \times (23^2)^2 \times 23$

Nous avons besoin de connaître le reste de la division par 2.
En effet, si le reste est 0, c'est qu'elle n'apparaît pas dans l'égalité et donc ne doit pas être ajoutée au résultat final.

13	|	2					
1	|	6	|	2			
		0	|	3	|	2	
				1	|	1	| 2
						1	| 0

4.4.1 Avec un algorithme

```
Initialisation :
res = 1
n = 13
c = 23

Tant que n > 0 # tant que la division euclidienne n'est pas terminée
    Si n % 2 == 1 # si le reste égal 1
        res *= c [p] # on ajoute au résultat final

    n = n//2 # reste de la division entière
    c = c^2 [p] # on passe au carré de deux suivant pour notre nombre

res contient le résultat de l'exponentiation modulaire
```

4.4.2 A la main

$S_0 = 23^{2^0} = 23[661]$ et nous le gardon de côté
$S_1 = 23^{2^1} = S_0^2 = 529[661]$ et il ne sert que de calcul temporaire
$S_2 = 23^{2^2} = S_1^2 = 238[661]$ et nous le gardons de côté
$S_3 = 23^{2^3} = S_2^2 = 459[661]$ et nous le gardons de côté

Donc : $res = S_0 \times S_2 \times S_3[661] = 105$
Note : on peut faire le modulo entre chaque multiplication pour éviter les grands nombres

5 Pour aller plus loin

5.1 La signature électronique et ElGamal

"La signature électronique apposée sur le document apporte la garantie de l'identité du signataire, son engagement et l'intégrité du document." *(ChamberSign, autorité de certification)*

Aujourd'hui, la signature électronique est utilisée pour certifier les emails, fichiers, factures, contrats etc. L'algorithme ElGamal est utilisé pour créer des signatures électroniques, mais il est légèrement différent de celui utilisé pour chiffrer des données.

5.1.1 La génération de la signature

On génère une clé publique avec l'algorithme ElGamal de la même façon que précédemment.
On 'code' le message selon une table qu'on a choisi.
On a besoin d'une fonction de hachage résistante aux collisions, c'est à dire qu'elle a une très faible probabilité de pour que deux nombres A et B donnent la même valeur de hachage et créent donc une confusion dans le chiffrement.
Nombre p premier (appartenant à la clé générée)

On choisi un nombre aléatoire k compris entre $0 < k < p1$ et tel que le $pgcd(k, p1) = 1$.
$r \equiv m \times k[p]$
$s \equiv (H(message)a \times r)k^1[p1]$

Si $s = 0$, on recommence l'algorithme.

La paire de nombre générés (r, s) est la signature du message que l'on va transférer. Elle est différente pour chaque message.

5.1.2 La vérification de la signature

On vérifie la signature (r, s) d'un message en s'assurant d'avoir :
$0 < r < p$ et $0 < s < p1$ et $m^{H(message)} \equiv n^r \times r^s[p]$

Les protocoles de cryptographie asymétrique comme ElGamal sont très utilisés pour faire des signatures numériques. Ils permettent de générer rapidement des clés aléatoires à partir d'un message.

5.2 Chiffrement homomorphe

5.2.1 En quoi ça consiste ?

Alice veut effectuer une addition entre deux chiffres a et b en passant par un serveur, mais elle ne veut pas que l'on puisse intercepter ces chiffres et le résultat de son opération. Elle chiffre a et b en $a1$ et $b1$ à l'aide de sa clé publique et envoie $a1$ et $b1$ au serveur avec sa clé publique.

Le serveur calcule le produit homomorphe de $a1$ et $b1$ et renvoie un résultat r obtenu à Alice qui, en utilisant sa clé pour déchiffrer r, récupère bien le contenu de l'addition de a, b. Le serveur a manipulé des chiffres sans connaître leur signification et sans pouvoir en exploiter leur résultat.

5.2.2 Une méthode encore en développement

C'est en 1978 que le problème du chiffrement homomorphe est posé pour la première fois. Seulement, il est difficile de le mettre en place avec les cryptosystèmes comme ElGamal ou le chiffrement RSA : ils ne sont que partiellement homomorphes. (Dans le cas du cryptosystème ElGamal, les chercheurs ont notamment rencontré des problèmes d'exponentiations modulaires illimitées).

En 2009 Craig Gentry, de l'université de Stanford est le premier à proposer un algorithme complètement homomorphe qui utilise la cryptographie à base de réseaux euclidiens et qui n'est donc pas basé sur les problèmes algorithmiques utilisés par les cryptosystèmes RSA et ElGamal. L'algorithme proposé par C.Gentry a pour avantage d'être résistant aux ordinateurs quantiques, mais a l'inconvénient de générer des clés très grandes et de prendre plus de temps que pour générer des clés avec des cyptosystèmes non-homomorphes. C'est pourquoi les chercheurs concentrent actuellement leurs recherches sur la réduction de la taille des clés générées et la réduction du temps de génération des clés.

6 Bibliographie / Sitographie

Elyotna, Le chiffrement homomorphe, dernière révision le 13/01/14, [en ligne]
consulté le 09/12/2019
Disponible sur :
https://linuxfr.org

GODEFROY Laurent, Chapitre 06 - Systèmes R.S.A. et ElGamal, [en ligne]
consulté le 28/11/2019
Disponible sur :
https://www.supinfo.com

NITULESCU Anca, Authentification et Intégrité : Signature numérique et Hachage,
dernière révision le 06/12/2016, [en ligne]
consulté le 02/12/2019
Disponible sur :
https://www.di.ens.fr

NITULESCU Anca, Cryptosystème ElGamal, dernière révision le 28/06/2016, [en ligne]
consulté le 28/11/2019
Disponible sur :
https://www.di.ens.fr

Bibmath.net, Le chiffre ElGamal, [en ligne]
consulté le 14/11/2019
Disponible sur :
http://bibmath.net

Bibmath.net, Test de primalité de Miller-Rabin, [en ligne]
consulté le 14/11/2019
Disponible sur :
http://www.bibmath.net

Cryptography.fandom.com, ElGamal encryption, dernière révision le 02/03/2017, [en ligne]
consulté le 21/11/2019
Disponible sur :
https://cryptography.fandom.com

laBRIi.fr, Nombres premiers,
consulté le 14/11/2019
Disponible sur :
https://www.labri.fr

Wikipédia, Cryptosystème de ElGamal, dernière révision le 27/12/2018, [en ligne]
consulté le 02/12/2019
Disponible sur :
https://fr.wikipedia.org

Wikipédia, Homomorphic Encryption, dernière révision le 08/12/2019, [en ligne]
consulté le 09/12/2019
Disponible sur :
https://en.wikipedia.org

Wikipédia, Test de primalité AKS, dernière révision le 25/10/2017, [en ligne]
consulté le 14/11/2019
Disponible sur :
https://fr.wikipedia.org/wiki/Test_de_primalit%C3%A9_AKS

Wikipédia, Test de primalité de Miller-Rabin , dernière révision le 27/11/2019, [en ligne]
consulté le 14/11/2019
Disponible sur :
https://fr.wikipedia.org