Problème du logarithme discret et cryptosystème Elgamal

christina.boura@uvsq.fr

6 avril 2018

1 Sécurité du protocole Diffie-Hellman

Afin de garantir la sécurité du protocole Diffie-Hellman, il faut que ce soit impossible pour un adversaire écoutant le trafic entre Alice et Bob de retrouver la clé secrète que les deux parties établissent à la fin du protocole.

Plus précisément, la sécurité du protocole est basée sur la difficulté de ce qu'on appèle le *problème Diffie-Hellman* qu'on peut décrire informelement comme suit :

Problème Diffie-Hellman. Etant donné un nombre premier p et un élément primitif α de $\mathbb{Z}_p^*$, il est difficile pour un adversaire de calculer $\alpha^{ab} \mod p$ avec seule la connaissance de $\alpha, p, \alpha^a \mod p$ et de $\alpha^b \mod p$.

En effet, si Éve se mets à travers du canal de communication entre Alice et Bob, il faut que ce soit difficile pour elle de deviner la clé établie $K = \alpha^{ab} \mod p$ juste en observant $A = \alpha^a \mod p$ et $B = \alpha^b \mod p$.

Le problème de Diffie-Hellman n'a pas de nos jours une solution triviale connue. La seule façon admise pour résoudre ce problème est de résoudre à la place un autre problème, celui du *logarithme discret* dans $\mathbb{Z}_p^*$. Pour cette raison on dit que la sécurité du protocole Diffie-Hellman est basée sur le problème du logarithme discret.

Nous avons vu que le groupe $\mathbb{Z}_p^*$ est un groupe cyclique fini. Par conséquent, il existe (au moins un) générateur α de ce groupe de façon que tout élément β du groupe s'écrit de façon unique sous la forme

$$\beta = \alpha^x \mod p,$$

pour $x \in \{1, \ldots, p-1\}$. L'exposant x est appelé le *logarithme discret de l'élément β de $\mathbb{Z}_p^*$*, car $x = \log_\alpha \beta$. Le problème du logarithme discret consiste à retrouver x à partir de β. On donne ensuite une définition un peu plus formelle de ce problème.

Définition 1.1. Problème du Logarithme Discret dans $\mathbb{Z}_p^*$.

Soit le groupe $\mathbb{Z}_p^*$ d'ordre $p-1$. Soit α un élément primitif dans $\mathbb{Z}_p^*$ et β un élément de $\mathbb{Z}_p^*$. Le *problème du logarithme discret* consiste à trouver un entier $1 \leq x \leq p-1$ tel que :

$$\alpha^x = \beta \mod p.$$

Il est facile de voir que si Éve sait résoudre le problème du logarithme discret dans $\mathbb{Z}_p^*$, alors elle est en mesure de retrouver la clé secrète sur laquelle Alice et Bob se mettent d'accord. En effet, si Éve sait résoudre ce problème, elle peut alors retrouver la clé secrète a de Alice à partir de $A = \alpha^a \mod p$ ainsi que la clé secrète b de Bob à partir de $B = \alpha^b \mod p$. Avec la connaissance de a et b elle peut alors facilement calculer $K = \alpha^{ab} \mod p$.

Il existe trois conditions nécessaires pour que le problème du logarithme discret soit difficile :

1. Le nombre premier p doit être grand.
2. Le plus petit entier positif m tel que $\alpha^m \equiv 1 \mod p$ doit être grand.
3. $p-1$ doit avoir au moins un grand facteur premier.

1.1 La taille de p

Exemple 1.2. Soit $p = 23423429$ $\alpha = 2$ et soit

$$\beta = \alpha^x = 19556038 \mod p.$$

Est-il difficile de retrouver x ?

Dans cet exemple il est relativement facile de retrouver x en simplement essayant une par une toutes les valeurs possibles pour x : On commence par $x = 0$, ensuite $x = 1$ etc. jusqu'à trouver x tel que $\alpha^x = \beta \mod p$. Les nombres dans cet exemple sont donc très petits, $p \approx 2^{25}$ et donc il y a au plus autant de nombres à tester afin de trouver une solution.

Pour cette raison il est recommandé de choisir des nombres premiers d'au moins 1024 bits, c'est-à-dire $p > 2^{1024}$.

1.2 L'ordre de α modulo p

Exemple 1.3. Soit le nombre premier $p = 589640540809904183368339807$.

Soient $\alpha = 151369338077029664651937484$ et

$$\beta = 438271202732874518716402322 \mod p.$$

Est-il difficile de trouver a tel que $\beta \equiv \alpha^a \mod p$?

Dans cet exemple, le problème du logarithme discret est facile puisque nous pouvons vérifier que $\alpha^3 \equiv 1 \mod p$. Par conséquent, $\alpha^a \equiv \alpha^{a \mod 3} \mod p$ et le β de cet exemple est forcement une des trois valeurs $\alpha^0 \mod p, \alpha^1 \mod p$ ou $\alpha^2 \mod p$. En particulier, comme $\beta \not\equiv 1, \alpha \mod p$ on voit directement que $a = 2$.

Le problème de cet exemple est que la valeur $\mathrm{ord}(\alpha)$ est très petite. Si α était choisi en étant un élément primitif de $\mathbb{Z}_p^*$ ce problème aurait été évité.

1.3 Les facteurs premiers de $p - 1$

1.3.1 Recherche exhaustive

Une recherche exhaustive consiste à calculer des puissances successives du générateur α, jusqu'à que le résultat soit égal à β :

$$\begin{aligned} \alpha^1 &\stackrel{?}{=} \beta \\ \alpha^2 &\stackrel{?}{=} \beta \\ &\vdots \\ \alpha^x &\stackrel{?}{=} \beta \end{aligned}$$

Cette méthode nécessite dans le pire cas $p-1$ étapes, car l'ordre du groupe $|\mathbb{Z}_p^*|$ est $p-1$. La complexité est alors $\mathcal{O}(p)$. Plus p est grand, plus la complexité de l'attaque par force brute est élevée. L'attaque par force brute n'est cependant pas l'attaque générique la plus puissante contre le problème du logarithme discret. Les algorithmes *Pas de Bébé, pas de géant* et *Rho-pollard* sont des algorithmes génériques (c.-à-d. n'exploitant pas des propriétés algébriques spécifiques au groupe) plus puissantes de compléxité $\mathcal{O}(\sqrt{p})$. On ne verra pas ces algorithmes ici.

Un troisième algorithme générique basé sur la factorisation en nombres premiers de la cardinalité du groupe est l'algorithme *Pohlig-Hellman*.

1.3.2 Algorithme Pohlig-Hellman

L'algorithme de Pohlig et Hellman est une méthode basée sur le théorème des restes chinois. Soit

$$|\mathbb{Z}_p^*| = p - 1 = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$$

la décomposition de la cardinalité du groupe $\mathbb{Z}_p^*$ en nombres premiers. On cherche de nouveau à calculer à partir de α, β et p, l'entier $x \in \mathbb{Z}_p$ tel que

$$\beta = \alpha^x \mod p.$$

Cet algorithme propose d'effectuer ce calcul en utilisant une approche *diviser pour régner*. Plus précisément, l'idée centrale est qu'à la place de résoudre ce problème dans le groupe $\mathbb{Z}_p^*$, c'est-à-dire trouver $x = \log_\alpha \beta \mod p$, on résout l'équation

$$x_i \equiv x \mod p_i^{e_i}$$

dans chaque sous-groupe d'ordre $p_i^{e_i}$ et on utilise le théorème des restes chinois pour déduire la solution finale. La complexité en temps de cet algorithme dépend clairement des facteurs premiers de $p-1$. Pour cette raison, afin d'éviter ce genre d'attaques, $p-1$ doit avoir au moins un très grand facteur premier. Un choix pour générer un nombre premier p possédant un grand diviseur premier est de générer un grand nombre premier q tel que $p = 2q+1$ est également premier. Des nombres premiers p ayant cette propriété sont appelés *nombres premiers sûrs* et un nombre premier q qui génère un nombre premier sûr est appelé *premier de Sophie Germain*.

2 Système de chiffrement Elgamal

Le système de chiffrement Elgamal a été inventé par Taher Elgamal en 1985. Ce système est une extension naturelle du protocole d'échange de clés Diffie-Hellman. Sa sécurité est basée alors aussi sur la difficulté du problème du logarithme discret. Ce système est décrit ici pour un groupe multiplicatif $\mathbb{Z}_p^*$, où p est un nombre premier, mais il peut être appliqué dans n'importe quel groupe cyclique pour lequel le problème du logarithme discret est connu d'être difficile.

2.1 De Diffie-Hellman à Elgamal

Afin de mieux comprendre le système de chiffrement Elgamal, nous commençons par voir comment ce dernier résulte de façon très naturelle du protocole de Diffie-Hellman.

Supposons qu'Alice souhaite envoyer un message chiffré à Bob. Alice et Bob se mettent alors d'accord sur un grand nombre premier p et sur un élément primitif $\alpha \in \mathbb{Z}_p^*$ et exécutent le protocole d'échange de clés Diffie-Hellman afin de dériver une clé commune K_{AB}. La nouvelle idée est maintenant qu'Alice utilise cette clé comme un masque multiplicatif pour chiffrer m en $c \equiv m \cdot K_{AB} \mod p$. Cette procédure est illustrée sur la figure 1.

Ce protocole consiste en deux étapes, l'échange classique de clés Diffie-Hellman suivi par les opérations de chiffrement et de déchiffrement. Pour la phase de chiffrement, Alice multiplie le message clair m par K_{AB} dans le groupe multiplicatif $\mathbb{Z}_p^*$. De l'autre coté, Bob inverse cette opération en multipliant par la clé inverse. Comme $\mathbb{Z}_p^*$ est un groupe, un élément est toujours inversible.

2.2 Le protocole Elgamal

Le chiffrement Elgamal consiste essentiellement en les mêmes opérations que le protocole décrit ci-dessus. Cependant, les différentes étapes sont légèrement modifiées et réarrangées afin de se résumer en un système plus efficace. On peut distinguer trois phases différentes. La phase de mise en place n'est effectuée qu'une seule fois par la personne qui établit la clé publique et qui sera le destinataire des messages. Les phases de chiffrement et de déchiffrement sont accomplies à chaque fois qu'un message est envoyé. Contrairement au protocole Diffie-Hellman, c'est Bob qui génère les paramètres p et α et les rends ensuite publiques, en les mettant par exemple sur une base de donnés publique ou sur son propre site web. Ce système est illustré sur l'image suivante.

Le protocole de chiffrement Elgamal réarrange l'ordre des opérations de la méthode naïve présentée à la section précédente. La raison pour cette modification est qu'Alice envoie maintenant qu'un seul message à Bob, contrairement aux deux messages envoyés précédemment.

Le chiffré consiste en deux morceaux, la clé éphémère k_E et le message masqué, c. Nous voyons alors que le chiffré (k_E, c) est deux fois plus long que le message.

On démontre maintenant la justesse du protocole. Plus précisément, on montre que l'opération $c \cdot k_M^{-1} \mod p$ produit effectivement le message clair m.

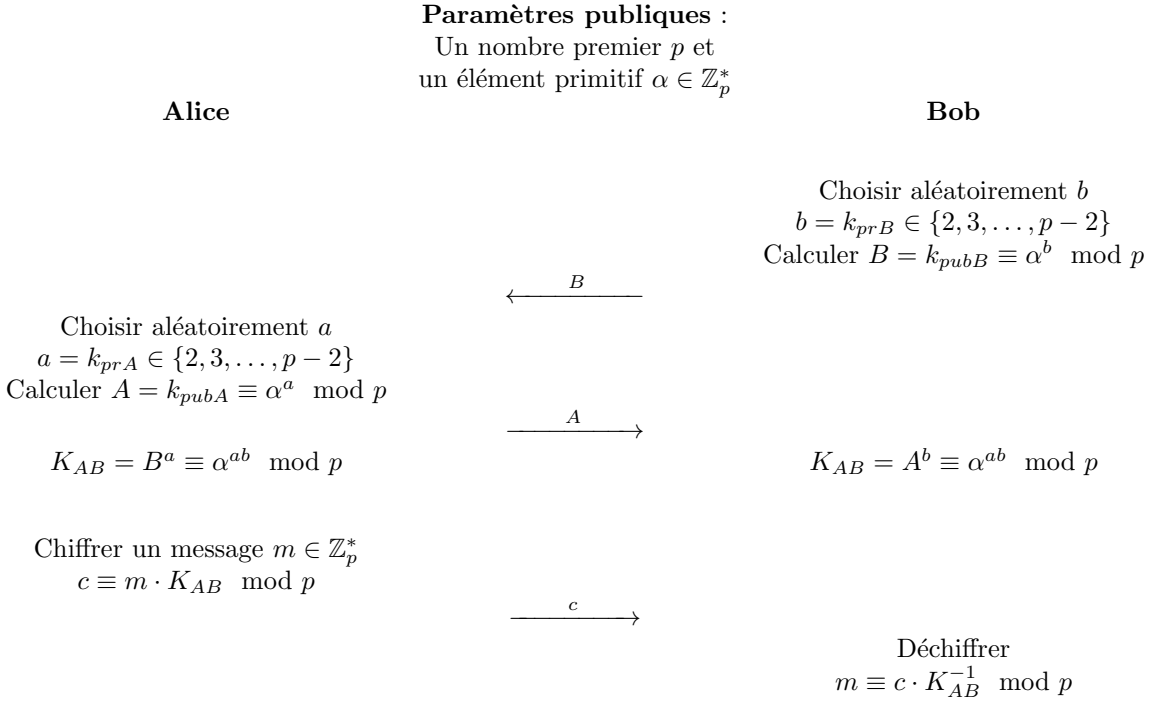

FIGURE 1 – Développement d'un système de chiffrement à clé publique à partir du protocole Diffie-Hellman

$$\begin{aligned} c \cdot k_M^{-1} &= (m \cdot k_M) \cdot (\beta^k)^{-1} \mod p \\ &= (m \cdot k_M) \cdot ((\alpha^d)^k)^{-1} \mod p \\ &= (m \cdot k_M) \cdot ((\alpha^k)^d)^{-1} \mod p \\ &= (m \cdot k_M) \cdot (k_E^d)^{-1} \mod p \\ &= (m \cdot (\alpha^d)^k)((\alpha^k)^d)^{-1} \mod p \\ &= m \cdot \alpha^{dk-dk} \mod p \\ &= m \mod p. \end{aligned}$$

Il est important de noter que le système de chiffrement Elgamal est un chiffrement *randomisé*, c'est-à-dire que chaque message m peut être chiffré de plusieurs manières possibles et chacun de ces chiffrés donne le même message après le déchiffrement. Ceci vient du fait qu'Alice choisit aléatoirement pour chaque message un exposant k différent. Par conséquent, la clé éphémère $k_M = \beta^k$ est choisie aléatoirement pour chaque chiffrement. Cette propriété a l'avantage suivant : si plusieurs messages sont chiffrés simultanément, des clairs identiques (dans $\mathbb{Z}_p$) seront transformés avec une grande probabilité en chiffrés différents. Ceci est une propriété cryptographique très désirable dont nous avons vu qu'on peut satisfaire dans le cadre des modes opératoires pour le chiffrement par bloc avec l'utilisation d'une valeur initiale (IV) pour chaque nouveau chiffrement. Le revers de la médaille est que le message chiffré est plus grand que le message clair. Plus précisément, le nombre de bits du message chiffré est le double de celui du message clair.

2.2.1 Le rôle de l'entier k

L'entier k est choisi par la personne souhaitant chiffrer un message (Alice dans notre exemple). On peut remarquer que la personne qui déchiffre le message (ici Bob) n'a pas besoin de connaître k afin de déchiffrer m. Il est très important pour la sécurité du système que retrouver k à partir de $k_E \equiv \alpha^k \mod p$ soit difficile. En effet, si un attaquant trouvait k, il pourrait calculer la clé éphémère $k_M \equiv \beta^k \mod p$ et retrouver le message m. Ceci nécessite à l'attaquant de calculer le logarithme discret $k \equiv \log_\alpha k_E \mod p$. Si l'entier p est choisi grand et de façon que $p-1$ est sans petits facteurs premiers ce problème devient calculatoirement très difficile.

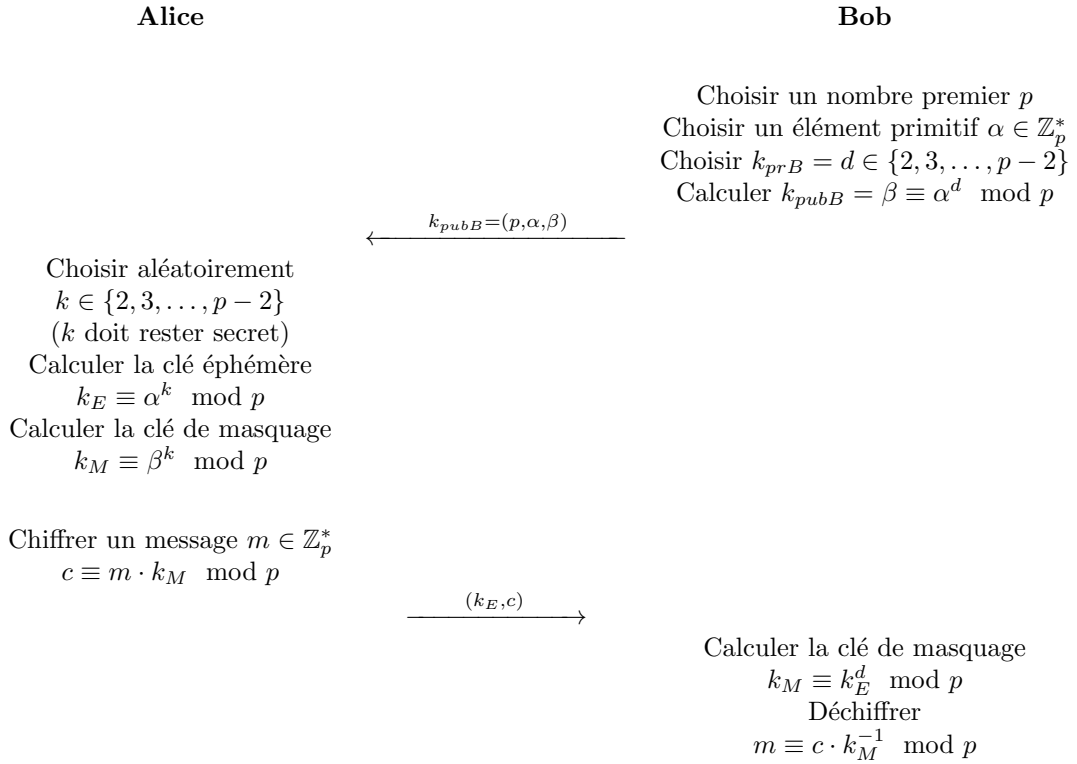

FIGURE 2 – Le protocole de chiffrement Elgamal

Réutilisation de k. Il faut faire très attention à ce que l'exposant k ne soit jamais réutilisé pour chiffrer un deuxième message. Supposons le contraire, c'est-à-dire que le même k est généré deux fois pendant le chiffrement des messages m_1 et m_2 et supposons qu'un attaquant connaît le premier message m_1. Nous allons montrer que l'attaquant peut alors automatiquement calculer m_2. En effet, si k est le même, alors la clé de masquage k_E utilisée pendant les deux chiffrements sera la même. On note c_1 et c_2 les deux chiffrés correspondant. L'attaquant sait alors que

$$k_M \equiv c_1 \cdot m_1^{-1} \equiv c_2 \cdot m_2^{-1} \mod p,$$

et il peut calculer

$$m_2 = c_2 \cdot c_1^{-1} \cdot m_1 \mod p.$$

2.3 Aspects calculatoires

2.3.1 Génération de clés

Pendant la phase de génération de clés, le récepteur du message (Bob dans notre exemple) doit générer un nombre premier p et calculer ensuite sa clé publique et sa clé privée. Puisque la sécurité du chiffrement Elgamal dépend du problème du logarithme discret, l'entier p doit avoir les propriétés que nous avons déjà discuté, c'est-à-dire p doit être un nombre très grand (d'au moins 1024 bits) avec $p-1$ ayant au moins un facteur premier très grand pour éviter l'attaque de Pohlig-Hellman. Comme dans le cadre de RSA, le nombre premier p doit être généré en utilisant un bon générateur aléatoire non prévisible et testé ensuite avec un test probabiliste de primalité (Fermat, Miller-Rabin ou Solovay-Strassen). La génération de la clé publique nécessite une exponentiation modulaire, pour laquelle l'algorithme `square and multiply` doit être utilisé.

2.3.2 Chiffrement

Pendant la phase de chiffrement, deux exponentiations modulaires et une multiplication modulaire sont nécessaires pour calculer la clé éphémère et la clé de masquage et pour chiffrer le message. L'algorithme `square and multiply` doit alors être utilisé de nouveau pour les deux premières opérations. Il est important de noter que les deux exponentiations de cette phase sont indépendantes du message clair.

Par conséquent, pour certaines applications, ces deux opérations peuvent être effectuées à l'avance et leur résultat peut être stocké et utilisé quand ceci sera nécessaire. Ceci constitue un avantage important dans la pratique.

2.3.3 Déchiffrement

Les étapes principales pendant le déchiffrement sont l'exponentiation pour obtenir la clé $k_M \equiv k^d \mod p$, avec l'algorithme `square and multiply`, suivi par une inversion modulaire de k_M en utilisant l'algorithme d'Euclide étendu. Cependant, il existe un raccourcis basé sur le Petit Théorème de Fermat, qui combine les deux étapes en une seule. De ce théorème, il s'en suit que

$$k_E^{p-1} \equiv 1 \mod p,$$

pour tout $k_E \in \mathbb{Z}_p^*$. On peut maintenant fusionner les deux étapes comme suit :

$$\begin{aligned} k_M^{-1} &\equiv (k_E^d)^{-1} \mod p \\ &\equiv (k_E^d)^{-1} k_E^{p-1} \mod p \\ &\equiv k_E^{p-d-1} \mod p. \end{aligned}$$

Cette méthode nous permet alors de calculer l'inverse de la clé de masquage en utilisant une seule exponentiation modulaire avec l'exposant $(p-d-1)$. Par conséquent, le déchiffrement nécessite essentiellement une seule application de l'algorithme `square and multiply`, suivi d'une multiplication modulaire pour déchiffrer le message.