Pourquoi les versions théoriques d'ElGamal et de RSA ne sont pas sûres ?

Sylvain Pasini

SSC

Projet de Semestre

Mars 2005

Responsable
Prof. Serge Vaudenay
EPFL / LASEC

Superviseur
Thomas Baignères
EPFL / LASEC

Merci

Tout a commencé en mars 2004, le jour où je me suis rendu à mon premier cours de *Sécurité et Cryptographie*. Ce cours m'intéressait, certes, mais à ce moment, j'étais loin de penser que je pourrais un jour réaliser un projet dans ce domaine... J'aimerais donc commencer par remercier le Professeur Serge Vaudenay de m'avoir fait découvrir, tout au long de son cours, une nouvelle passion : la Cryptographie. J'aimerais également le remercier de m'avoir permis de réaliser un projet au sein de son laboratoire.

Je voudrais également dire un grand merci à Thomas Baignères, l'assistant m'ayant aidé tout au long de ce travail. Je tiens à préciser qu'au delà de son aide en cryptographie, il m'a entre autre assisté en programmation C ou encore pour mes débuts avec LaTeX. Je le remercie également pour sa patience et sa disponibilité ainsi que pour les conseils qu'il m'a fournis pour mon orientation future. En effet, grâce à ses précieux conseils, j'ai choisi de continuer l'aventure au LASEC. Merci Thomas !

Un grand merci également à Matthieu Finiasz qui m'a donné quelques conseils tout au long de ce projet.

Il ne faut évidemment pas oublier mes parents. Grâce à eux, j'ai l'occasion de suivre une excellente formation et je les en remercie énormément. Finalement, je remercie mon amie Nadia de m'avoir laissé du temps libre pour pouvoir mener à bien ce projet. Je les remercie également tous les trois pour leur soutien.

Table des matières

Chapitre 1

Introduction

Dans la littérature, les descriptions données pour les chiffrements d'ElGamal et de RSA sont appelées "textbook". Elles représentent les versions théoriques de ces cryptosystèmes. Selon la manière dont elles sont utilisées, ces versions ne satisfont pas les critères basiques de sécurité. L'objectif du travail présenté dans ce document est de montrer cette insécurité.

Pour obtenir un système sûr en utilisant l'un des ces deux algorithmes de chiffrement, il est préférable de réaliser un prétraitement sur le texte clair avant le chiffrement. Il est donc recommandé de ne pas utiliser les versions "textbook" telles quelles mais de les utiliser au sein d'un standard. Beaucoup utilisent pour RSA le standard "Optimal Asymmetric Encryption Padding", connu sous le nom RSA-OAEP[7]. En ce qui concerne ElGamal, il n'existe pas un standard mais un certain nombre de propositions.

Le chiffrement asymétrique est énormément plus lent que le chiffrement symétrique. Ceci est du aux calculs réalisés sur de très grand nombres. Le problème du chiffrement symétrique est l'échange de la clé de session. C'est pourquoi, souvent le chiffrement asymétrique est utilisé pour transmettre une clé de session. Les communications suivantes sont quand à elles chiffrées de manière symétrique pour garantir de meilleures performances temporelles.

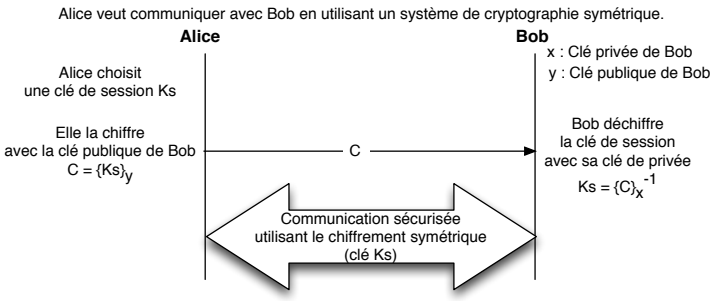

FIG. 1.1 – Cryptographie asymétrique utilisée pour la transmission d'une clé symétrique K_s

Dans ce document, une attaque sur ElGamal ainsi que sur RSA tels qu'ils

sont donnés théoriquement est décrite. Cette attaque illustre le fait que les versions "textbook" ne sont fondamentalement pas sûres.

L'attaque est basée sur le fait que le chiffrement à clé publique est, par conséquent, souvent utilisé pour échanger des clés de sessions. Ces clés sont typiquement courtes, en principe moins de 256 bits. DES, par exemple, utilise des clés de 56 bits.

Ce travail est basé sur la publication de Dan Boneh, Antoine Joux et Phong Q. Nguyen publiée lors de Asiacrypt 2000. Elle s'intitule "Why Textbook ElGamal and RSA Encryption Are Insecure" [1].

1.1 ElGamal en bref

ElGamal[5] est un système de chiffrement asymétrique. La sécurité de ce type de chiffrement repose en partie sur le problème du logarithme discret. En effet, dans $\mathbb{Z}_p$ avec p suffisamment grand, l'exponentiation est réalisable relativement rapidement. Au contraire, le logarithme discret est énormément plus difficile et donc plus long.

Voilà en bref le fonctionnement du chiffrement ElGamal :

Paramètres publiques :	p, un grand nombre premier
	g, un générateur de $\mathbb{Z}_p^*$
Initialisation :	générer aléatoirement $x \in \mathbb{Z}_{p-1}$
	$y = g^x \bmod p$
Clé secrète :	$K_s = x$
Clé publique :	$K_p = y$
Message :	M, un élément de $\mathbb{Z}_p^*$
Chiffrement :	génère aléatoirement $r \in \mathbb{Z}_{p-1}$
	$C = \langle u = g^r mod\ p, v = M \cdot y^r \bmod p \rangle$
Déchiffrement :	$M = vu^{-x}$

Une particularité du chiffrement d'ElGamal est qu'il n'est pas déterministe. En effet, pour des paramètres identiques, tels que p, g et la paire de clés, deux chiffrements du même message clair M donneront deux messages chiffrés différents.

1.2 RSA en bref

RSA est un système de chiffrement asymétrique. Il a été inventé par Ronald **R**ivest, Adi **S**hamir et Leonard **A**dleman et publié en 1978[6]. Le nom de ce système cryptographique est en fait la première lettre de chacun de ses créateurs. La sécurité de RSA repose en partie sur le problème de la factorisation. En effet, il est facile de choisir deux grands nombres premiers, q et q, puis d'en créer un troisième $N = p \cdot q$. Par contre, il est très difficile de retrouver p et q à partir de N, en supposant que N soit suffisamment grand.

Voilà en bref le fonctionnement du chiffrement RSA :

Paramètres publiques :	s, un entier
Set up :	générer aléatoirement deux nombres premiers p et q différents de $s/2$ bits $N = pq$ e, un nombre aléatoire tel que $\gcd(e, (p-1)(q-1)) = 1$ $d = e^{-1} \bmod ((p-1)(q-1))$
Clé secrète :	$K_s = (d, N)$
Clé publique :	$K_p = (e, N)$
Message :	M, un élément de $\mathbb{Z}_N^*$
Chiffrement :	$C = M^e \bmod N$
Déchiffrement :	$M = C^d \bmod N$

Chapitre 2

L'attaque théorique

Au cours de ce chapitre, l'attaque théorique sera démontrée. Ceci sera fait de manière à ce que des cryptanalistes de niveau raisonnable puissent la comprendre sans trop de difficultés.

2.1 Le problème des arrondis dans un sous-groupe

ElGamal chiffre des messages dans $\mathbb{Z}_p^*$ pour p premier. Ici, g ne génère pas tout le groupe. Pour accélérer le chiffrement, l'élément g est souvent choisi d'ordre beaucoup plus petit que p. Supposons que g soit un élément de $\mathbb{Z}_p^*$ d'ordre q. Par exemple, p serait de 1024 bits alors que q seulement de 512 bits.

$$g \in \mathbb{Z}_p^*, \text{ ordre}(g) = q \text{ tel que } q \ll p, \text{ on sait alors que } q|p-1 \tag{2.1}$$

La clé secrète est un nombre x tel que

$$1 \leq x < q \tag{2.2}$$

Le chiffrement, le déchiffrement tout comme la clé publique y sont définis de la même manière qu'auparavant.

L'élément g génère donc un sous-groupe G_q de $\mathbb{Z}_p^*$. G_q contient q éléments. Il est important de noter que G_q est très dispersé dans $\mathbb{Z}_p^*$ (car $q \ll p$). En reprenant l'exemple ci-dessus, on voit que G_q a 2^{512} éléments parmi les 2^{1024} de $\mathbb{Z}_p^*$. Donc un rapport de 1 élément G_q pour 2^{512} de $\mathbb{Z}_p^*$.

Assumons maintenant que M est un message court. Plus précisément :

$$\text{length}(M) \ll \log_2(p/q) \tag{2.3}$$

Par exemple, M est un message de 64 bits, typiquement une clé de session.

Pour comprendre l'attaque, il est bénéfique de considérer une petite modification sur ElGamal. Le chiffrement est appliqué de la manière suivante :

$$r \in \mathbb{Z}_q^*, u = g^r \text{ et } v = M + y^r \bmod p \tag{2.4}$$

Autrement dit, le message est "masqué" par l' **addition** de y^r au lieu de la multiplication.

Il est possible de voir cette attaque d'une manière intuitive :

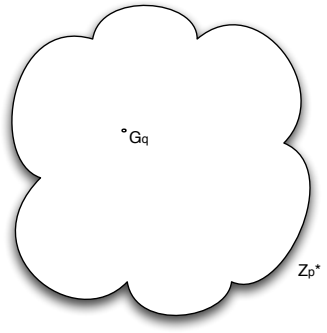

FIG. 2.1 – Vue d'ensemble de G_q dans $\mathbb{Z}_p^*$

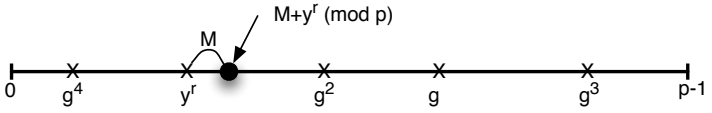

FIG. 2.2 – Visualisation d'un message chiffré dans l'ensemble $\mathbb{Z}_p^*$, les **X** représentent les éléments appartenants au sous-groupe G_q

- $y^r = g^{x \cdot r}$ appartient clairement à G_q, car $g \in G_q$.
- y^r est un élément aléatoire de G_q, car r est aléatoire.
- Supposons que G_q soit distribué uniformément dans $\mathbb{Z}_p$
- L'écart moyen entre deux éléments de G_q est p/q
- Grâce à l'équation (2.3), $M \ll p/q$
- Finalement, v est très proche d'un élément de G_q, l'écart avec l'élément le plus proche donne M.

Par conséquent, avec une très grande probabilité, il y a un unique élément $z \in G_q$ qui est suffisamment proche de v pour en déduire M. D'où :

$$\exists z, \text{ tel que } |v - z| < 2^{64} \tag{2.5}$$

Si z est obtenu, alors la valeur du message M également. Ceci montre d'une manière intuitive la résolution de la version additive du problème des arrondis dans un sous-groupe.

Les additions arrondies dans un sous-groupe : Soit $z \in G_q$ et M un entier tel que $M < 2^m$.
En connaissant $u = z + M \bmod p$, trouver z.
En supposant que G_q soit distribué uniformément dans $\mathbb{Z}_p$, lorsque m est suffisamment petit, z a une unique solution avec une grande probabilité .

Les multiplications arrondies dans un sous-groupe : Soit $z \in G_q$ et M un entier tel que $M < 2^m$.

En connaissant $u = z \cdot M \bmod p$, trouver z.
En supposant que G_q soit distribué uniformément dans $\mathbb{Z}_p$, lorsque m est suffisamment petit, z a une unique solution avec une grande probabilité.

Une solution performante à ce problème impliquerais que plain ElGamal ne serait pas sûre. Il serait intéressant de trouver des solutions en temps $O(\sqrt{M})$ ou $O(\log M)$. Dans la section suivante, une solution est proposée pour résoudre ce problème.

2.2 Algorithmes pour les multiplications arrondies dans un sous-groupe

Soit un élément v de la forme :

$$v = z \cdot M \bmod p \tag{2.6}$$

où z est un élément aléatoire de G_q et $M < 2^m$. L'objectif est de trouver M. Comme auparavant, assumons :

$$m \ll \log_2(p/q) \tag{2.7}$$

On peut par exemple prendre un p de 1024 bits, un q de 512 bits et un M de $m = 64$ bits.

Voici une solution simple du type "Meet-in-the-middle" pour les multiplications arrondies dans un sous-groupe. Une solution plus performante sera également présentée. Elle se rapproche du problème du sac à dos ("Knapsack").

2.2.1 Une attaque du type Meet-in-the-middle

Supposons que M puisse être écrit sous la forme $M = M_1 \cdot M_2$ où $M_1 \leq 2^{m_1}$ et $M_2 \leq 2^{m_2}$ avec $m_1 + m_2 = m$, m étant le nombre de bits de M. Il est maintenant possible d'observer la chose suivante :

$$v = z \cdot M = z \cdot M_1 \cdot M_2 \bmod p$$

$$v/M_2 = z \cdot M_1 \bmod p$$

$$(v/M_2)^q = z^q \cdot M_1^q \bmod p \tag{2.8}$$

Le corollaire du théorème de Lagrange implique que l'ordre d'un élément divise l'ordre du groupe ou du sous-groupe dans lequel il se trouve, ici l'ordre de z divise donc q. L'équation (2.8) s'écrit alors :

$$(v/M_2)^q = M_1^q \bmod p \tag{2.9}$$

Il est maintenant possible de construire une table contenant toutes les valeurs de M_1^q soit $\forall M_1 = 0, ..., 2^{m_1} - 1$. La table a donc 2^{m_1} valeurs. Chaque valeur appartient à $\mathbb{Z}_p$ donc est représentée avec $\log_2(p)$ bits. Ensuite, pour chaque $M_2 = 0, ..., 2^{m_2} - 1$, il suffit de vérifier si $(v/M_2)^q \bmod p$ est présent dans la table. Si oui, la valeur $M = M_1 \cdot M_2$ est un candidat. Bien que M soit unique, il y aura certainement plusieurs couples (M_1, M_2) candidats.

L'algorithme ci-dessus utilise a priori $2^{m_1} + 2^{m_2}$ exponentiations modulaires et $2^{m_1} \log_2(p)$ bits de mémoire. En choisissant différentes valeurs de m_1 et m_2, il est possible d'obtenir différents compromis temps/mémoire.

Pour réaliser cette attaque, il est évident qu'il est nécessaire que le message M de m bits soit découpable en deux parties de m_1 bits et m_2 bits. Des statistiques ont été réalisées pour connaître cette probabilité en fonction du nombre de bits du message ainsi que du nombre de bits utilisés pour remplir chacune des tables, elles se trouvent à la section 4.1.

2.2.2 Réduction à un problème du type sac à dos

Le problème des multiplications arrondies dans un sous-groupe peut être résolu d'une manière plus performante. Il peut en effet être transformé en un problème linéaire. Certaines conditions sont cependant nécessaires :

- p doit être tel que $p - 1 = q \cdot r \cdot s$
- où $s \geq 2^m$ est tel que le logarithme discret dans le sous-groupe de $\mathbb{Z}_p^*$ d'ordre s n'est pas difficile.
- q étant le *gros* facteur premier de $p - 1$ et r le reste.

Pour réaliser cela, il est nécessaire de commencer par la factorisation de $p - 1$ jusqu'à obtenir un s si possible supérieur à 2^m. Cette opération peut être effectuée en utilisant la méthode Pollard $p - 1$. Le résultat sera du type $s = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$. Lorsque $p - 1$ a un facteur premier q tel que tous les facteurs de $q - 1$ soient inférieurs à B, $q - 1$ est B-lisse. Dans ces conditions, le temps nécessaire pour factoriser $p - 1$ sera en $O(B)$. Comme supposer précédemment, $p - 1$ a des petits facteurs $q_i \in s$, par conséquent les $q_i - 1$ auront des facteurs au dessous d'une borne.

Connaissant la factorisation de s, en supposant que s satisfasse les critères ci-dessus, il est possible de réaliser le logarithme discret. Il faut en fait résoudre le problème "Discrete Logarithm with Known Order Fact" (DLKOFP). Ceci peut être réalisé en utilisant la méthode de Pohlig-Hellman. Cette méthode demandera un temps de $O(\sum_{i=1}^{k} e_i \cdot (\log s + \sqrt{p_i}))$, voir [4] page 108.

Soit w un générateur de $\mathbb{Z}_p^*$. Pour tout $x \in \mathbb{Z}_p^*$, x^{qr} appartient au sous-groupe G_s (d'ordre s) généré par w^{qr}. Il est donc maintenant possible de calculer $\log_{w^{qr}} x^{qr}$ car l'on se trouve dans le sous-groupe G_s.

Le problème linéaire considéré est connu sous le nom **"k-table problem"** : Soit k tables d'entiers $T_1, ..., T_k$ et un entier cible n. Le "k-table problem" retourne toutes les expressions possible du type $n = t_1 + t_2 + ... + t_k$ où les $t_i \in T_i$. Le problème général a été étudié par Schroeppel et Shamir[8] car beaucoup de problèmes NP-dur peuvent être réduit à un "k-table problem". Certaines solutions connues vont être appliquées pour k =2,3 et 4.

"k-table problem"

Les solutions du "k-table problem" sont présentées pour k=2 et 3.

Le problème modulaire 2-table Supposons à nouveau que M puisse être écrit sous la forme $M = M_1 \cdot M_2$, avec $0 \leq M_1 \leq 2^{m_1}$ et $0 \leq M_2 \leq 2^{m_2}$. On obtient alors :

$$v = z \cdot M = z \cdot M_1 \cdot M_2 \bmod p$$

$$v^{qr} = z^{qr} \cdot M_1^{qr} \cdot M_2^{qr} = M_1^{qr} \cdot M_2^{qr} \bmod p \tag{2.10}$$

car z appartient à G_q. Elle peut être réécrite sous la forme

$$\log_{w^{qr}}(v^{qr}) = \log_{w^{qr}}(M_1^{qr}) + \log_{w^{qr}}(M_2^{qr}) \bmod s \tag{2.11}$$

Pour résoudre ce problème, il suffit de reconnaître la forme d'une 2-table.

- La table T_1 contiendra les éléments $\log_{w^{qr}}(M_1^{qr}),\ \forall M_1 = 0, ..., 2^{m_1} - 1$.
- La table T_2 contiendra les éléments $\log_{w^{qr}}(M_2^{qr}),\ \forall M_2 = 0, ..., 2^{m_2} - 1$.
- L'entier cible sera $\log_{w^{qr}}(v^{qr})$.

Le problème est donc d'exprimer la cible comme une somme modulaire de $t_1 + t_2$ où $t_1 \in T_1$ et $t_2 \in T_2$. Le nombre de cibles du type $t_1 + t_2$ est $2^{m_1+m_2}$. Il faut s'attendre à avoir peu de solutions lorsque $s \geq 2^{m_1+m_2}$. Le problème implique des additions modulaires. Il peut cependant être vu comme un "2-table problem" ayant 2 cibles, $\log_{w^{qr}}(v^{qr})$ et $\log_{w^{qr}}(v^{qr}) + s$.
La manière classique de résoudre ce problème est décrite à la figure 2.3.

En entrée : La cible n ainsi que les deux tables T_1 et T_2
En sortie : Toutes les solutions du type $n = t_1 + t_2$ avec $t_1 \in T_1$ et $t_2 \in T_2$
Algorithme :
Trier T_1 de manière croissante
Trier T_2 de manière décroissante
Répéter jusqu'à que T_1 ou T_2 soit vide :
Calculer $t = first(T_1) + first(T_2)$
Selon t :
Si $t = n$, afficher la solution trouvée et supprimer $first(T_1)$ et $first(T_2)$.
Si $t < n$, supprimer $first(T_1)$ de T_1.
Si $t > n$, supprimer $first(T_2)$ de T_2.

FIG. 2.3 – Algorithme de résolution du 2-table problem

La méthode va donc donner toutes les solutions du "2-table problem". Il faudra supprimer $2^{min(m_1,m_2)}$ éléments à chacune des deux tables pour que l'une des deux soit vide. Le temps utilisé pour résoudre le problème du sac à dos sera donc de l'ordre de $O(2^{min(m_1,m_2)+1})$. Chaque table T_i possède 2^{m_i} éléments. En constatant que les éléments stockés dans les tables T_1 et T_2 sont les mêmes, le temps utilisé pour les remplir est de l'ordre de $O(2^{\max(m_1,m_2)})$ exponentielles et logarithmes discrets. La mémoire utilisée sera par contre de l'ordre de $O(2^{m_1} + 2^{m_2})$ éléments de $\log_2(p)$ bits.

Le problème modulaire 3-table L'approche précédente peut être étendue à un nombre arbitraire de facteurs. Comme expliqué précédemment, la probabilité de réussite de l'attaque dépend directement de la probabilité de pouvoir découper le message en plusieurs facteurs. Les statistiques de la section 4.1 montrent qu'il est plus difficile de découper un message en 3 facteurs plutôt qu'en 2. Il est toutefois intuitif, que plus le nombre de facteurs est grand, moins la probabilité de réussite sera grande. C'est la raison pour laquelle, ce document s'arrête à trois facteurs.

Supposons que le message puisse être écrit de la manière suivante : $M = M_1 \cdot M_2 \cdot M_3$ où chaque M_i a au plus m_i bits. L'equation 2.11 peut être étendue

à trois facteurs. On obtient alors :

$$\log_{w^{qr}}(v^{qr}) = \sum_{i=1}^{3} \log_{w^{qr}}(M_i^{qr}) \bmod s \tag{2.12}$$

Maintenant, il s'agit toujours de résoudre un problème du type *sac à dos*, mais cette fois avec trois facteurs ! Contrairement aux apparences, cette tâche n'est pas très difficile. Le principe est simple, il suffit de résoudre 2^{m_1} problèmes à deux tables. Pour chacun de ces sous-problèmes, il faut utiliser les tables T_2 et T_3 et comme cible $\log_{w^{qr}}(v^{qr}) - t_1$. De cette manière, le résultat du sous problème sera :

$$\log_{w^{qr}}(v^{qr}) - t_1 = t_2 + t_3 \bmod s \tag{2.13}$$

Qui peut être réécrite de la forme suivante et qui prouve que l'on est bien en train de résoudre le problème voulu :

$$\log_{w^{qr}}(v^{qr}) = t_1 + t_2 + t_3 \bmod s \tag{2.14}$$

L'algorithme est décrit d'une manière globale sur la figure 2.4. Pour résoudre

En entrée : La cible n ainsi que les trois tables T_1, T_2 et T_3
En sortie : Toutes les solutions du type $n = t_1 + t_2 + t_3$ avec $t_i \in T_i \ \forall i = 1..3$
Algorithme :
  Trier T_2 de manière croissante
  Trier T_3 de manière décroissante
  Pour chacun des $t_i \in T_1$
    La sous-cible vaut $n' = n - t_i$
    Résoudre le problème à deux tables, T_2 et T_3, en utilisant comme cible n'.

FIG. 2.4 – Algorithme de résolution du 3-table problem

le problème du sac à dos, l'algorithme de la figure 2.4 utilise, par analogie au problème à deux tables, un temps de l'ordre de $O(2^{m_1} \cdot 2^{\min(m_2,m_3)+1})$. La même constatation que pour le problème à deux tables peut être faite : les éléments stockés dans les tables T_1, T_2 et T_3 sont les mêmes. Le temps utilisé pour remplir ces tables est donc de l'ordre de $O(2^{\max(m_1,m_2,m_3)})$. Il utilisera par contre une mémoire de l'ordre de $O(2^{m_1} + 2^{m_2} + 2^{m_3})$.

Grâce à l'utilisation de trois tables, il est possible de diminuer l'espace mémoire. En effet, avec deux tables, la mémoire utilisée était de l'ordre de $O(2^{m_1} + 2^{m_2})$, soit environ $O(2 \cdot 2^{m/2})$. Maintenant, la mémoire nécessaire est de l'ordre de $O(3 \cdot 2^{m/3})$. Par contre, la complexité **théorique** a augmenté. Il est important de voir que ceci est la complexité théorique et non pratique. Au chapitre traitant de l'implémentation, il sera expliqué pourquoi avec trois tables, le temps de calcul est plus faible qu'avec deux tables. Un point faible de l'utilisation de trois tables est la diminution de la probabilité de succès. Comme toujours, tout est une histoire de compromis. Avec un facteur, l'attaque sur DES, par exemple, est difficilement réalisable. Avec deux, elle devient envisageable, mais avec une probabilité de succès plus faible. Finalement avec trois facteurs, elle est réalisable mais avec une probabilité de succès encore plus faible, voir section 4.5.

2.3 Résumé de l'attaque sur ElGamal présentée ci-dessus

Cette section donne uniquement un bref rappel du déroulement de l'attaque. Certains détails ont par conséquent été omis volontairement, ils sont par contre disponible dans les sections correspondantes.

Il est important de rappeler les différents paramètres connus :

- Les paramètres publiques d'ElGamal p et g.
- La clé publique y
- Le ciphertext $\langle u, v\rangle$ avec $u = g^r$ et $v = M \cdot y^r \bmod p$
- Les caractéristiques du systèmes, comme par exemple le nombre de bits du message M.

Cette méthode s'applique pour un g générant un sous-groupe de $\mathbb{Z}_p^*$ d'ordre q. Il faut donc commencer par s'assurer que cette propriété soit satisfaite.

A l'aide du problème des multiplications arrondies dans un sous-groupe, il est possible de trouver l'unique y^r à partir de v, en considérant que $M \ll (p/q)$. Il suffit ensuite d'en déduire M.

A la section 2.2, plusieurs algorithmes résolvant ce problème ont été étudiés. En premier, une attaque du type Meet-in-the-middle a été montrée. Cette attaque ne sera pas utilisé telle quelle. En effet, elle a ensuite été réduite à un problème du type sac à dos.

Pour résoudre le problème du sac à dos, il faut en premier réduire la complexité des calculs en réduisant l'espace de travail. Il s'agit ici de travailler dans un sous groupe G_s de $\mathbb{Z}_p^*$ d'ordre s avec $s \geq 2^m$. Pour cette raison, il est nécessaire de trouver suffisamment de petits facteurs de $p-1$. Le logarithme discret est supposé pas difficile dans le sous-groupe G_s. Finalement, le résultat était :

$$\log_{w^{qr}}(v^{qr}) = \log_{w^{qr}}(M_1^{qr}) + \log_{w^{qr}}(M_2^{qr}) \bmod s \tag{2.15}$$

La résolution de cette équation n'est rien d'autre qu'un problème à 2-table de la forme $cible = t_1 + t_2 \bmod s$, avec t_i appartenant à la table T_i, voir seciton 2.2.2.

En résolvant ce problème, on obtient tous les couples $\langle t_1, t_2\rangle$ candidats. A partir de ces valeurs, il est possible de retrouver les couples $\langle M_1, M_2\rangle$ candidats en recalculant $\forall M_i = 0..2^{m_i} - 1$ les valeurs $\log_{w^{qr}}(M_i^{qr})$ jusqu'à avoir obtenu toutes les relations $t_i \rightarrow M_i$ voulues.

2.4 Une attaque sur ElGamal utilisant un générateur de $\mathbb{Z}_p^*$

L'attaque sur ElGamal présentée dans les sections précédentes est appliquée à un cas particulier d'ElGamal. Cette attaque suppose que ElGamal utilise un élément $g \in \mathbb{Z}_p^*$ dont l'ordre est beaucoup plus petit que p. En fait, g ne génèrait pas tout le groupe $\mathbb{Z}_p^*$.

Bien que beaucoup d'implémentations d'ElGamal utilisent un tel g, il est important d'étudier le cas où g génère entièrement $\mathbb{Z}_p^*$. Il est clair qu'il n'est pas possible d'utiliser tel quel l'algorithme présenté auparavant car le problème des arrondis dans un sous-groupe n'est plus applicable.

Soient les composants publics suivant : $\langle p, g, y\rangle$ tel que g génère tout le groupe $\mathbb{Z}_p^*$. Supposons un message M de m bits. Le chiffrement avec ElGamal de ce message donne le message chiffré suivant :

$$r \in \mathbb{Z}_p^*, u = g^r \text{ et } v = M \cdot y^r \bmod p \tag{2.16}$$

Supposons que s est un facteur de $p-1$ tel que le logarithme discret n'est pas difficile dans le sous-groupe de $\mathbb{Z}_p^*$ d'ordre s. Par exemple, s pourrait être un entier contenant que de petits facteurs premiers (s est *lisse*). Ce sous groupe est dénommé G_s. Il est maintenant possible de définir :

$$q = (p-1)/s \tag{2.17}$$

Comme auparavant, supposons que $M = M_1 \cdot M_2$ où M_1 et M_2 sont inférieurs à $2^{m/2}$. D'où :

$$\begin{aligned} v &= M_1 \cdot M_2 \cdot y^r mod\ p \\ M_1 \cdot y^r &= v/M_2 mod\ p \\ M_1^q \cdot (y^r)^q &= v^q/M_2^q mod\ p \end{aligned} \tag{2.18}$$

Il n'est pas possible d'utiliser les techniques de la section 2.2 directement car la valeur de y^{rq} est inconnue. y^{rq} appartient à G_s. Par conséquent, il est possible de déterminer y^{rq} en utilisant la clé publique y et $u = g^r$. En effet, supposons que nous connaissons a tel que $y^q = (g^q)^a$. Sachant que le logarithme discret dans G_s n'est pas difficile, il est possible d'en déduire a :

$$a = \log_{g^q}(y^q) \tag{2.19}$$

Grâce à a, y^{rq} est connu et vaut :

$$y^{rq} = g^{a \cdot rq} = (g^r)^{aq} = v^{aq}$$

L'équation 2.18 s'écrit maintenant :

$$M_1^q \cdot v^{aq} = v^q / M_2^q \bmod p \tag{2.20}$$

Les techniques de la section 2.2 peuvent maintenant être utilisées pour obtenir tous les couples $\langle M_1, M_2\rangle$ satisfaisant l'équation ci-dessus.

2.5 Une attaque sur plain RSA du type Meet-in-the-middle

Il est possible de remarquer que le même style d'attaque peut être utilisé contre RSA. Un système RSA chiffre un message $M \in \mathbb{Z}_N$ avec $N = p \cdot q$, p et q étant deux grands nombres premiers différents. Le message chiffré C est déterminé de la manière suivante : $C = M^e \bmod N$ avec e faisant partie de la clé publique, voir section 1.2.

Souvent, pour accélérer le chiffrement, l'exposant e contenu dans la clé publique est choisit beaucoup plus petit que N, par exemple $e = 2^{16} + 1$.

Supposons que le message M de m bits puisse être écrit comme $M = M_1 \cdot M_2$ avec $M_1 \leq 2^{m_1}$ et $M_2 \leq 2^{m_2}$.

Le message chiffré peut alors s'écrire de la manière suivante :

$$C = (M_1 \cdot M_2)^e \bmod N$$

$$\frac{C}{M_2^e} = M_1^e \bmod N \tag{2.21}$$

L'attaque est alors réalisable. Il est au préalable nécessaire de créer une table T_1 contenant les valeurs $M_1^e \bmod N \ \forall M_1 = 0, ..., 2^{m_1}$. Ensuite pour chaque $M_2 = 0, ..., 2^{m_2}$, Il faut vérifier si $\frac{C}{M_2^e}$ est présent dans la table T_1. Chaque collision révèle un couple $\langle M_1, M_2 \rangle$ candidat.

La mémoire utilisée est de l'ordre de 2^{m_1} éléments de taille N. Il est nécessaire de réaliser 2^{m_1} exponentielles pour remplir la table T_1. De plus, il faut réaliser à nouveau 2^{m_2} exponentielles pour la recherche des collisions, les divisons étant négligeables. Le temps total nécessaire est de l'ordre de $O(2^{m_1} + 2^{m_2})$ exponentielles modulaires.

Chapitre 3

Implémentation

Ce chapitre décrit brièvement les solutions choisies pour l'implémentation de l'attaque. Avant de pouvoir réaliser l'attaque, il a fallu implémenter les diverses applications permettant de générer des paramètres, des paires de clés, des messages ainsi que le chiffrement/déchiffrement de ce dernier. Chacune de ces applications sont décrites dans les sections suivantes. Le langage de programmation utilisé est bien évidemment le C.

Comme expliqué dans le chapitre précédent, quelques problèmes typiques à la cryptographie ont d'abord dus être résolus. Ces derniers sont principalement la factorisation, le logarithme discret ainsi que le sac à dos.

3.1 The GNU Multiple Precision Arithmetic Library (GMP)

Les algorithmes de chiffrement montrés dans ce document, ElGamal et RSA, utilisent de grands nombres. Le chiffrement d'ElGamal, par exemple, peut utiliser un groupe $\mathbb{Z}_p^*$ avec un p de 1024 bits.

Les ordinateurs traditionnels utilisent en principe des nombre de 32 ou 64 bits. Pour le moment, il n'est pas possible de travailler directement sur des nombres d'un millier de bits.

La librairie GMP permet de travailler à un niveau supérieur. Elle permet de travailler avec de très grands nombres. Leur taille n'est limitée que par la quantité de mémoire à disposition. Le travail à effectuer dans ce projet est alors possible grâce à cette librairie. De plus, la plupart des fonctions traditionnelles y sont déjà implémentées. Par exemple, les fonctions d'addition, de soustraction, de puissance ou encore de modulo y sont présentes et sont optimisées.

Le travail effectué dans le cadre de ce projet est donc entièrement basé sur la librairie GMP.

3.2 Factorisation

Comme expliqué précédemment, la factorisation utilisera l'algorithme *Pollard* $p-1$. Il faudra donc en premier implémenter cet algorithme. Il a légèrement été modifié de manière à ce qu'il s'arrête après un certain temps. En effet, si

aucun facteur est trouvé, l'algorithme continue sa boucle. La boucle est aussi grande que la taille du groupe. En travaillant dans un très grand groupe, cette boucle devient vite très longue. C'est pourquoi l'algorithme a été modifié : si aucun petit facteur est obtenu, l'algorithme sera stoppé après un certain nombre d'itération. L'algorithme modifiée est décrit à la figure 3.1.

En entrée : Le nombre n dont on cherche un facteur
En sortie : Un facteur de n ou "Erreur"
Algorithme :
Choisir un x aléatoirement dans $\{2, ..., n-1\}$

Si gcd $\neq 1$ alors
Afficher ce gcd et stopper

$i \leftarrow 1$
Tant que $\gcd(x-1, n) = 1$ faire
$x \leftarrow x^i \bmod n$
$i \leftarrow i + 1$
Si $i \geq nbr\_max\_iterations$ **alors**
Stopper

Si $x = 1$ alors
Afficher "Erreur"
Sinon
Afficher ce $\gcd(x-1, n)$ et stopper

FIG. 3.1 – Algorithme *Pollard* $p-1$ modifié

Le problème est que l'algorithme *Pollard* $p-1$ ne donne qu'un facteur du nombre à factoriser et non sa factorisation. Il a donc fallu construire un algorithme permettant de factoriser un nombre n quelconque. L'algorithme retenu est donné à la figure 3.2.

La particularité de cet algorithme est qu'il permet de trouver, si cela est possible, tous les petits facteurs jusqu'à en avoir obtenu suffisamment. Ce *suffisamment* est spécifié en entrée comme une limite minimale.

3.3 Logarithme discret

L'algorithme de *Pohlig-Hellman* permet de décomposer le calcul d'un logarithme discret en plusieurs calculs de petits logarithme discrets. Il demande cependant de connaître la factorisation complète de l'ordre du groupe dans lequel est effectué le logarithme discret. Grâce à la section précédente, il est maintenant possible de factoriser un nombre n à condition qu'il aie des facteurs pas trop grands.

Ce n'est donc pas un hasard si au chapitre précédent, le sous-groupe de $\mathbb{Z}_p^*$ pour lequel nous avions supposer que le logarithme discret n'était pas difficile. En fait, il s'agissait de trouver un groupe, ici G_s, dont la factorisation de l'ordre s est entièrement connue. Il faut bien évidemment que s aie un taille raisonnable. s est choisi comme étant les plus petits facteurs de $p-1$ permettant d'avoir un s supérieur au message M, donc $s \geq 2^m$, m étant le nombre de bits du message.

En entrée : Le nombre à factoriser n et la limite minimale de *petits* facteurs à trouver
En sortie : La décomposition de n en facteurs premiers (jusqu'à atteindre la limite)
Algorithme :

Si n est premier :
Ajouter directement n à la liste des facteurs premiers

Sinon :
Chercher un facteur f à l'aide de l'algorithme *Pollard p-1*

Si *Pollard p-1* réussi
Découper n en deux facteurs f_{small} et f_{big}

Si le petit facteur f_{small} est un composite
Alors le factoriser (entièrement)
Sinon (il est premier)
Alors l'ajouter à la liste des facteurs premiers

Si la limite est atteinte
Ajouter le grand facteur f_{big} à la liste des facteurs premiers (malgré qu'il ne soit pas forcément premier)
Sinon
Factoriser le grand facteur f_{big} (appel récursif)

Si finalement la limite n'a pas été atteinte
Retouner -1(limite minimale pas respectée)
Sinon
Retouner 1 (tout est ok)

Si *Pollard p-1* a atteint la limite maximale du nombre d'itérations
Facteur trop long a rechercher
Retourner -1 (limite minimale pas respectée)

Sinon (*Pollard p-1* a échoué sur ce facteur)
Alors ajouter n à la liste des facteurs premiers
Retourner 0 (erreur)

FIG. 3.2 – Algorithme permettant de factoriser un nombre n

L'algorithme de *Pohlig-Hellman* a été utilisé tel qu'il est décrit dans le cours du Professeur Vaudenay[2].

Cet algorithme permet uniquement la décomposition en plusieurs logarithmes discrets ainsi que la déduction du résultat final en utilisant le *Théorème du Reste Chinois*[2]. Il a donc fallu implémenter également une fonction permettant de calculer un logarithme dans un sous-groupe d'ordre premier. Cette tâche est accomplie par une recherche exhaustive pour les éléments appartenant à un petit sous-groupe. Si le sous-groupe est trop grand, la recherche peut s'avérer très longue. Par conséquent, pour cette taille de sous-groupes, il est préférable d'utiliser l'algorithme *Baby steps-Giant steps*, comme décrit dans [2].

3.4 Résolution du problème de type sac à dos

La résolution d'un tel problème demande énormément de mémoire. Par exemple, les clé de DES utilisent 56 bits. Une attaque en utilisant deux tables nécessite donc 2 tables de 2^{28} éléments soit 2^{29} éléments en mémoire. Chacun de ces éléments est composé d'environs 1000 bits. Dans la plus part des cas, ElGamal est utilisé avec 512 ou 1024 bits. Par conséquent, chaque élément occupe autant de place que 32 entiers de 32 bits, soit 2^5 fois la taille d'un entier conventionnel. Finalement, la mémoire occupée est de l'ordre de $2^{29} \cdot 2^5 \cdot 4 = 2^{36}$ bytes, cela représente tout de même 64 GB !

Avec cette petite démonstration, il est clair qu'il sera très difficile d'attaquer le chiffrement DES en utilisant 2 tables.

3.4.1 Résolution en utilisant deux tables

La résolution du problème s'occupe également de l'allocation mémoire, du remplissage des tables, etc. L'algorithme utilisé pour résoudre ce problème est exactement celui décrit à la figure 2.3.

3.4.2 Résolution en utilisant trois tables

Comme pour le problème précédent, la résolution de ce problème prend également en compte, l'allocation mémoire, le remplissage de table et d'autres opérations du même style. L'algorithme utilisé pour résoudre ce problème est celui décrit à la figure 2.4. Comme expliqué dans la section correspondante, cet algorithme résolvant le problème à trois tables n'est rien d'autres que la résolution de 2^{m_1} problèmes à deux tables, 2^{m_1} étant le nombre d'éléments présents dans la première table.

3.5 Les applications concernants ElGamal

Afin de pouvoir réaliser des attaques, il a fallu tout d'abord implémenter une série d'applications. Ces applications sont appelées par le terminal. Elles dialoguent entre elles par des fichiers. Ces derniers sont utilisés de manière à prouver qu'une attaque est réalisable sans informations à priori. Par exemple, avant de lancer l'attaque, il est possible de détruire la clé privée ainsi que le message original.

Les applications sont décrites dans les paragraphes suivants. Ceci permettra de comprendre le fonctionnement global de chacune d'elle ainsi que de voir l'utilité des fichiers utilisés comme moyen de communication.

3.5.1 Librairie des fonctions du chiffrement d'ElGamal

Ce premier élément n'est pas une application réelle, mais plutôt une sorte de librairie. Elle contient une série de fonctions propre au chiffrement d'ElGamal. Entre autre, les fonctions de génération de paramètres, de génération de clés, de chiffrement et de déchiffrement y sont présentes. Celles-ci seront donc appelées lorsqu'elles seront nécessaires.

3.5.2 Génération des paramètres : elgamal_parametres

Cette application s'occupe de générer les paramètres publics d'ElGamal ainsi qu'une paire de clés. Grâce à la librairie qui vient d'être définie, il suffit d'appeler la génération des paramètres pour obtenir p et g.

- Le premier paramètre permet de déterminer le groupe dans lequel les opérations seront effectuée, ici $\mathbb{Z}_p^*$. Comme désiré pour l'attaque, il faut que p soit premier et aie n bits. De plus, il est nécessaire que l'ordre du groupe, ici $p-1$, aie au moins un grand facteur de q bits ainsi que suffisamment de petits facteurs pour que l'on obtienne un sous-groupe supérieur au nombre de bits m du message.
- Le second paramètre est le générateur g du sous-groupe d'ordre q, G_q.

Ces paramètres sont finalement stocké dans le fichier *parametres.elgamal*. Ils pourront donc être récupérés ultérieurement par n'importe quelle application.

Finalement, une paire de clés est générée. Cette opération est réalisée de manière conventionnelle. C'est-à-dire que la clé privée x est choisie aléatoirement dans $\mathbb{Z}_{p-1}$ et la clé publique y est déterminée comme étant $g^x \bmod p$. Les deux clés sont stockées dans deux fichiers différents. Ceci permet entre autre de supprimer la clé privée avant de lancer l'attaque et ainsi de démontrer que cette clé n'a pas été utilisée pour retrouver le message. La clé privée est stockée dans le fichier *cle_privee.elgamal* et la clé publique est stockée dans le fichier *cle_publique.elgamal*.

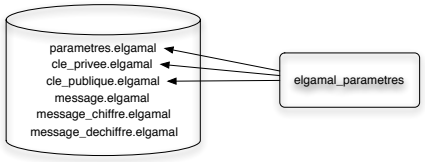

FIG. 3.3 – ElGamal : Génération des paramètres

3.5.3 Génération d'un message : elgamal_message

Cette application est appelée pour générer un message. Le message est choisi aléatoirement. La seule contrainte est qu'il soit composé d'exactement m bits.

La factorisation du message est également affichée afin de voir la forme générale de sa décomposition en facteurs premiers. Finalement le message est stocké dans le fichier *message.elgamal*.

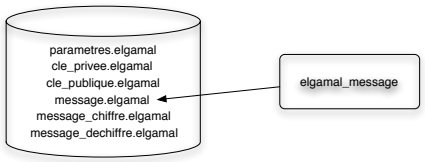

FIG. 3.4 – ElGamal : Génération d'un message

3.5.4 Savoir si le message est attaquable : elgamal_it_is_splittable

Comme expliqué auparavant, il est possible de savoir à l'avance si l'attaque fonctionnera ou non. Le critère est que le message puisse être séparé en deux facteurs de m_1 et m_2 bits ou trois facteurs de m_1, m_2 et m_3 bits selon le type d'attaque choisie. Cette application cherche alors toutes les solutions possibles de découpes du message à partir de sa décomposition en facteurs premiers. Le message est lu depuis le fichier *message.elgamal* Si une découpe est réalisable avec les paramètres de l'attaque alors le message sera attaquable. Au contraire, si aucune découpe satisfaisant les critères n'a été trouvée, alors le message ne sera pas attaquable.

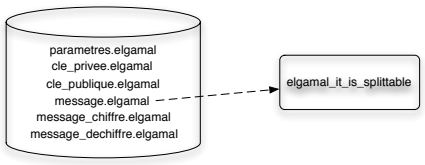

FIG. 3.5 – ElGamal : Savoir si le message est attaquable

3.5.5 Chiffrement du message : elgamal_chiffre

Cette application lit le message depuis le fichier *message.elgamal* ainsi que la clé publique depuis le fichier *cle_publique.elgamal*. Ensuite, ce message est chiffré en appelant la fonction correspondante de la librairie ElGamal décrite ci-dessus. Finalement, le message chiffré, qui se compose de u et de v, est enregistré dans le fichier *message_chiffre.elgamal*.

3.5.6 Déchiffrement du message chiffré : elgamal_dechiffre

Cette application lit le message chiffré depuis le fichier *message_chiffre.elgamal* ainsi que la clé privée depuis le fichier *cle_privee.elgamal*. Ensuite, le message

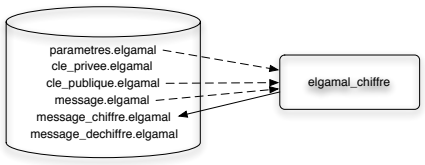

FIG. 3.6 – ElGamal : Chiffrement du message

chiffré est déchiffré en appelant la fonction correspondante de la librairie ElGamal décrite ci-dessus. Finalement, le message déchiffré M' est enregistré dans le fichier *message_dechiffre.elgamal*.

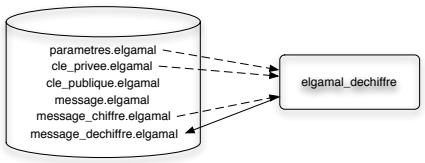

FIG. 3.7 – ElGamal : Déchiffrement du message chiffré

3.6 L'attaque sur le chiffrement d'ElGamal

3.6.1 L'application : elgamal_attaque

Cette application lit le message chiffré depuis le fichier *message_chiffre.elgamal* ainsi que les paramètres depuis le fichier *parametres.elgamal*. Elle s'occupe d'attaquer le message chiffré. Il en résulte une liste de candidats susceptibles d'être le message original. Cette liste est souvent, pour ne pas dire presque toujours, très restreinte. La plupart du temps, seul le message réel est présent.

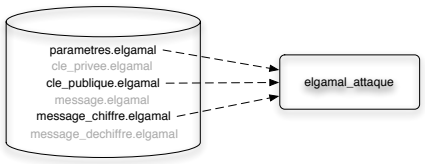

FIG. 3.8 – ElGamal : Attaque à partir du message chiffré

3.6.2 Un aperçu des opérations réalisées

L'attaque étant la partie importante, il est préférable de se rendre compte des différentes étapes réalisées. La figure 3.9 décrit la séquence d'opérations

effectuées.

Algorithme :

1. Lecture des paramètres p et g depuis le fichier *parametres.elgamal*
2. Factoriser l'ordre du groupe, donc $p-1$, jusqu'à obtenir suffisamment de petits facteurs
3. Extraire s, les petits facteurs de $p-1$ tels que $s \geq 2^m$
4. Déduire qr, $p-1 = q \cdot r \cdot s$
5. Chercher un w générateur de G_s
6. Remplir les tables avec $\log_{w^{qr}}(M^{qr}), \forall M = 0, ..., 2^{\max(m_1,m_2,m_3)} - 1$
7. Trier les tables

---

8. Lecture du message chiffré $\langle u, v \rangle$ depuis le fichier *message_chiffre.elgamal*
9. Déterminer la cible (utilisée pour résoudre le problème du sac à dos)
10. Résoudre le problème du sac à dos
11. Rechercher les M_i correspondants aux $\log_{w^{qr}}(M_i^{qr})$
12. Afficher les candidats du message

FIG. 3.9 – L'attaque sur le chiffrement d'ElGamal

Il est possible de distinguer deux parties de l'attaque :
- La première partie, constituée des étapes 1 à 7, est propre aux paramètres publiques. En effet, cette phase est le **prétraitement**. Elle peut être utilisée pour plusieurs messages, la seule condition est que les paramètres publiques p et g n'aient pas changés. En principe, ceux-ci sont contenus dans la clé publique et sont propres aux destinataire du message.
- La seconde partie doit être recalculée pour chaque nouveau message chiffré. En effet, la cible du problème du sac à dos dépend directement du message chiffré. Cette phase est souvent appelée **récupération du message**.

3.6.3 L'attaque et sa complexité

Cette section explique plus en détails les moyens utilisés pour réaliser l'attaque sur ElGamal. En parallèle, l'analyse de la complexité est détaillée.

La phase de prétraitement est réalisée de la manière suivante :

1. La lecture des paramètres est un simple accès à un fichier. p et g, qui sont en fait publiques et facilement accessible, sont donc extraits de *parametres.elgamal*.
2. La première étape importante de l'attaque est la factorisation de $p-1$. Cette opération est réalisée en utilisant l'algorithme décrit à la section 3.2. $p-1$ n'est pas entièrement factorisé. Comme expliqué dans l'étape suivante, seule une petite partie est nécessaire.
3. L'extraction de s est maintenant relativement facile. Il suffit de prendre les petits facteurs de la factorisation de $p-1$ jusqu'à en avoir suffisamment. Il suffit que s soit supérieur à la taille du message, donc supérieur à 2^m.
4. Maintenant que s est connu, qr peut en être déduit facilement en sachant que $p-1 = q \cdot r \cdot s$. Il suffit donc de déterminer qr comme étant $\frac{p-1}{s}$.

5. Pour trouver un générateur de G_s, la méthode est simple. Il suffit de tirer au hasard des nombres, $w \in \mathbb{Z}_p^*$, jusqu'à ce que w^{qr} soit d'ordre s Sachant que factorisation de s est connue et que w^{qr} appartient au sous-groupe G_s, l'ordre de cet élément est donc forcément un facteur de s. Il suffit donc de vérifier s'il n'existe pas d'autre facteurs f de s tel que $(w^{qr})^f \equiv 1 \bmod p$.
6. L'une des étapes les plus coûteuses en temps est le remplissage des tables. Il s'agit ici de remplir les deux, voir trois tables selon le type de l'attaque. Cette opération est réalisée en effectuant une unique fois le calcul de $\log_{w^{qr}}(M^{qr})$ pour tous les $M = 0, ..., 2^{\max(m_1,m_2,m_3)} - 1$. Les résultats sont stockés dans les deux ou trois tables T_i simultanément. La seule condition est que $M < 2^{m_i}$, donc chaque table a finalement une taille indépendante des autres. Comme nous l'avions vu, cette étape requiert un temps en $O(2^{\max(m_1,m_2,m_3)})$ exponentielles et logarithmes discrets.
7. L'étape suivante est le tri des tables. Cette opération est, d'après les mesures réalisées, pratiquement négligeable. En principe, même sur de très grandes tables, le tri nécessite au maximum une dizaine de secondes. Ce temps est obtenu grâce à la librairie *quick sort* que l'on trouve dans les systèmes sous le nom *qsort*. La seule chose à réaliser pour l'utiliser est de fournir une fonction de critère permettant de déterminer le classement des éléments.

La phase de récupération du message est, quand à elle, réalisée selon les lignes ci-dessous :

8. La lecture du message chiffré est faite de la même manière que la lecture des paramètres publiques.
9. Pour pouvoir résoudre le problème du sac à dos, il est crucial de trouver la cible. Comme expliqué théoriquement, la cible est simplement $\log_{w^{qr}}(v^{qr})$.
10. La prochaine étape est la résolution du problème du sac à dos. Il s'agit donc de trouver les éléments $t_i \in \text{table}T_i$ tels que $cible = \sum t_i$. Ici, il est important de distinguer une attaque à deux tables, d'une attaque à trois tables :
    - Dans le cas d'une attaque à deux tables, la complexité de la résolution du sac à dos est faible. Il suffit de résoudre une fois un énorme sac à dos de deux tables. Comme vu précédemment, cette opération nécessite un temps en $O(2 \cdot 2^{\min(m_1,m_2)})$ additions.
    - Dans le cas d'une attaque à trois tables, la complexité de la résolution du sac à dos est beaucoup plus grande que dans le cas précédent. En effet, il s'agit ici de résoudre 2^{m_1} sac à dos de deux tables contenant 2^{m_2} et 2^{m_3} éléments. Le temps nécessaire pour réaliser ceci est en $O(2^{m_1} \cdot 2 \cdot 2^{\min(m_2,m_3)})$.
11. Finalement, il reste à retrouver les valeur des M_i à partir des éléments candidats, qui sont de la forme $\log_{w^{qr}}(M_i^{qr})$. Il est donc nécessaire de recalculer $\log_{w^{qr}}(M^{qr})$ pour tous les $M = 0, ..., 2^{\max(m_1,m_2,m_3)} - 1$. Le temps utilisé est donc le même que lors du remplissage des tables. La seule différence est qu'il est possible de s'arrêter lorsque tous les éléments candidats stockés dans la liste des résultats ont été retrouvés.

La complexité totale d'une attaque à 2 tables est finalement en $O(2 \cdot 2^{\max(m_1,m_2)})$ exponentielles modulaires et logarithmes discrets et $O(2 \cdot 2^{\min(m_1,m_2)})$ additions.

La mémoire utilisée est en $O(2^{m_1} + 2^{m_2})$ éléments appartenant au groupe $\mathbb{Z}_p^*$, donc de $\log_2(p)$ bits.

La complexité totale d'une attaque à 3 tables est en $O(2 \cdot 2^{\max(m_1,m_2,m_3)})$ exponentielles modulaires et logarithmes discrets et $O(2^{m_1} \cdot 2 \cdot 2^{\min(m_2,m_3)})$ additions. La mémoire utilisée est en $O(2^{m_1} + 2^{m_2} + 2^{m_3})$ éléments appartenant au groupe $\mathbb{Z}_p^*$, donc de $\log_2(p)$ bits.

3.7 Les applications concernants RSA

Comme pour le chiffrement d'ElGamal, il a fallu tout d'abord implémenter une série d'applications. Elles dialoguent également entre elles par des fichiers. Ces applications sont décrites dans les paragraphes suivants.

3.7.1 Librairie des fonctions du chiffrement d'ElGamal

Comme pour le chiffrement d'ElGamal, cette librairie contient une série de fonctions propre au chiffrement de RSA. Entre autre, les fonctions de génération de paramètres, de chiffrement et de déchiffrement y sont présentes.

3.7.2 Génération des paramètres : rsa_parametres

Cette application génère les paramètres utiles au chiffrement RSA. Ceci revient à créer la clé publique (e, N), stockée dans le fichier *cle_publique.rsa*, ainsi que la clé privée (d, N), stockée dans le fichier *cle_privee.rsa*.

Pour réaliser cela, il suffit de choisir deux nombres premiers différents de $n/2$ bits, p et q. Grâce à eux, il est possible de déterminer $N = p \cdot q$ ainsi que l'ordre du groupe $\mathbb{Z}_N^*$ qui est $\phi(N) = (p-1)(q-1)$. e est choisit aléatoirement tel que $\gcd(e, \phi(N)) = 1$. Finalement, d est déterminé comme étant $e^{-1} \bmod \phi(N)$. Les deux nombres p et q sont finalement détruits.

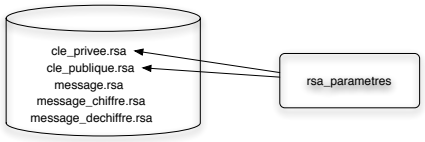

FIG. 3.10 – RSA : Génération des paramètres

3.7.3 Génération d'un message : rsa_message

Comme dans le cas d'ElGamal, cette application est appelée pour générer aléatoirement un message d'exactement m bits. Le message est stocké dans le fichier *message.rsa*.

3.7.4 Savoir si le message est attaquable : rsa_it_is_splittable

Comme pour le chiffrement d'ElGamal, il est possible de savoir à l'avance si l'attaque fonctionnera ou non. Le critère est le même que pour ElGamal, soit

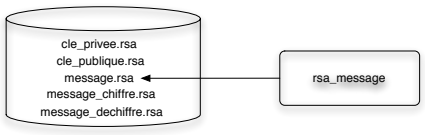

FIG. 3.11 – RSA : Génération d'un message

de savoir si le message est découpable en deux ou non. Le message est lu depuis le fichier *message.rsa*.

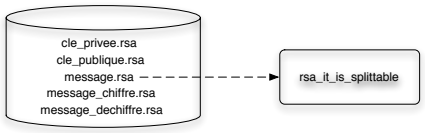

FIG. 3.12 – RSA : Savoir si le message est attaquable

3.7.5 Chiffrement du message : rsa_chiffre

Cette application lit le message depuis le fichier *message.rsa* ainsi que la clé publique depuis le fichier *cle_publique.rsa*. Ensuite, ce message est chiffré en appelant la fonction correspondante de la librairie RSA décrite ci-dessus. Finalement, le message chiffré est enregistré dans le fichier *message_chiffre.rsa*.

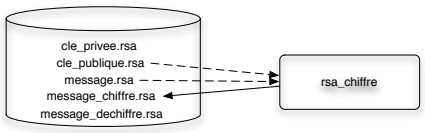

FIG. 3.13 – RSA : Chiffrement du message

3.7.6 Déchiffrement du message chiffré : rsa_dechiffre

Cette application lit le message chiffré depuis le fichier *message_chiffre.rsa* ainsi que la clé privée depuis le fichier *cle_privee.rsa*. Ensuite, le message chiffré est déchiffré en appelant la fonction correspondante de la librairie RSA décrite ci-dessus. Finalement, le message déchiffré M' est enregistré dans le fichier *message_dechiffre.rsa*.

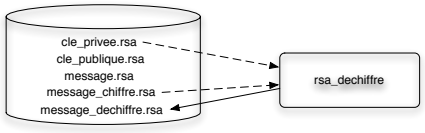

FIG. 3.14 – RSA : Déchiffrement du message chiffré

3.8 L'attaque sur le chiffrement de RSA

3.8.1 L'application : rsa_attaque

Cette application lit le message chiffré depuis le fichier *message_chiffre.rsa* ainsi que la clé publique depuis le fichier *cle_publique.rsa*. Elle s'occupe d'attaquer le message chiffré. Il en résulte une liste de candidats susceptibles d'être le message original. Cette liste est souvent, pour ne pas dire presque toujours, très restreinte. La plupart du temps, seul le message réel est présent.

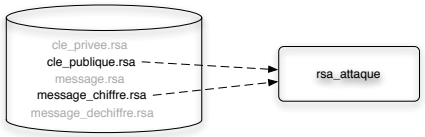

FIG. 3.15 – RSA : Attaque à partir du message chiffré

3.8.2 Un aperçu des opérations réalisées

L'attaque étant la partie importante, il est préférable de se rendre compte des différentes étapes réalisées. La figure 3.16 décrit la séquence d'opérations effectuées.

Algorithme :

1. Lecture de la clé publique, e et N depuis le fichier *cle_publique.rsa*
2. Créer la table T_1
   Pour chaque $M_1 = 0, ..., 2^{m_1} - 1$ calculer $M_1^e \bmod N$
   Trier la table

---

3. Lecture du message chiffré C depuis le fichier *message_chiffre.rsa*
4. Pour chaque $M_2 = 0, ..., 2^{m_2} - 1$: Vérifier si $\frac{C}{M_2^e} \bmod N$ appartient à T_1
   Chaque collision révèle un couple candidat pour M
   Mémoriser $M_1^e \bmod N$ et M_2.
5. Pour chaque couple candidat, rechercher M_1 en fonction de $M_1^e \bmod N$
6. Afficher les candidats de M

FIG. 3.16 – L'attaque sur le chiffrement de RSA

Comme pour l'attaque contre ElGamal, il est possible de distinguer deux parties :

- La première partie, constituée des étapes 1 et 2, est la phase **prétraitement**.
- La seconde partie, phase de **récupération du message**, doit être recalculée pour chaque nouveau message chiffré. Elle est constituée des étape 3 à 6.

3.8.3 L'attaque et sa complexité

Comme pour le cas d'ElGamal, l'analyse de l'attaque sur RSA est expliquée plus en détails dans cette section en essayant de traiter en parallèle la complexité.

L'attaque commence bien évidemment par la phase de prétraitement :

1. Il faut bien évidemment commencer par lire les paramètres publiques e et N. Ceux-ci sont stockés dans la clé publique. Une simple lecture du fichier *cle_publique.rsa* est effectuée.
2. La première étape importante de l'attaque est le remplissage de la table T_1. Il s'agit de calculer pour chaque $M_i \in [0..2^{m_1} - 1]$ sont correspondant $M_i^e \bmod N$ et de le stocker dans la table. Cette opération demande 2^{m_1} exponentielles modulaires. Cette table doit ensuite être triée de manière à retrouver facilement un élément. D'après les mesures réalisées, le tri est très peu coûteux. Généralement, cette opération est réalisée en moins d'une seconde. Elle est donc négligeable.

La phase de récupération du message est, quand à elle, réalisée comme décrit ci-dessous :

3. Pour retrouver le message clair, il faut commencer par lire le message chiffré. Dans notre cas, il faut effectuer une lecture du fichier *message_chiffre.rsa*.
4. La seconde étape importante de l'attaque est la recherche de collision(s). Pour chaque M_2, cette étape commence par calculer $C/M_2^e \bmod N$. Ensuite, une recherche dans la table T_1 est effectuée pour rechercher une éventuelle collision. Si c'est le cas, le couple $\langle M_1, M_2 \rangle$ est un candidat du message. Il est important de rappeler que M_1 n'est pas connu à ce moment, mais uniquement la valeur stockée dans la table soit $M_1^e \bmod N$. Cette dernière valeur ainsi que M_2 sont stockés dans la liste des couples candidats.
5. Finalement, il faut retrouver les valeur des M_i à partir des valeurs des $M_i^e \bmod N$. Il est donc à nouveau nécessaire de réaliser 2^{m_1} exponentielles modulaires.

La complexité totale de cette attaque est finalement en $O(2 \cdot 2^{m_1} + 2^{m_2})$ exponentielles modulaires. La mémoire utilisée est en $O(2^{m_1})$ éléments du groupe $\mathbb{Z}_N^*$, soit de $\log_2(N)$ bits.

Chapitre 4

Résultats

Tous les calculs effectués dans ce chapitre ont été réalisés sur la machine *Cluster64* du LASEC. Elle possède les caractéristiques suivantes :

Processeur	AMD Athlon 64 3800+
Mémoire RAM	4GB
Système d'exploitation	Linux SuSe, noyau 2.6

FIG. 4.1 – Caractéristiques de l'ordinateur utilisé : *Cluster64*

4.1 Probabilité de réussite d'une attaque sur ElGamal ou sur RSA

Ici, on suppose que toutes les conditions pour que l'attaque aie lieu soient satisfaites. Par exemple, les paramètres doivent être générés de la bonne manière, tels que p aie une partie lisse suffisamment grande. Malgré cela, il se peut que l'attaque sur un message chiffré réussisse alors que sur un autre message chiffré elle échoue. On a vu auparavant que l'attaque est du type sac à dos. C'est-à-dire qu'il y aura autant de solutions à notre attaque que de solutions au problème de sac à dos. Les solutions à ce problème sont de la forme :

$$cible = t_1 + t_2 + ... + t_k \bmod s$$

avec $t_i \in \text{table}T_i$, chaque table T_i contient $\log_{w^{qr}}(M_i^{qr})$ $\forall M_i = 0, ..., 2^{m_1} - 1$
Il y aura des résultats uniquement si au moins une solution à ce problème existe. Autrement dit, le message doit pouvoir s'écrire de la manière suivante :

$$M = M_1 \cdot M_2 \cdot ... \cdot M_k$$

En résumé, pour qu'un message (chiffré) soit attaquable, il est nécessaire et suffisant qu'il soit découpable en un certain nombre de facteurs, déterminé par le nombre de tables. Chaque facteur doit cependant avoir une taille limitée (exprimée en nombre de bits).

Pour connaître la probabilité qu'un message M soit découpable en 2, respectivement 3, facteurs de maximum m_1, m_2 et éventuellement m_3 bits, il a fallu

faire des statistiques. La figure 4.2 présente une méthode pour déterminer si un message est découpable ou non. Cette méthode a ensuite été réutilisée pour effectuer des statistiques sur un grands nombre de messages choisis aléatoirement.

En entrée : Le message M, les paramètres m_1 et m_2 bits.
En sortie : Découpable ou non
Algorithme :
- Factoriser M
- Pour toutes les répartitions possibles des facteurs premiers entre les 2 parties de M_i :
  - Si cette répartition est une solution, c'est-à-dire que le premier facteur a au maximum m_1 bits et le second a au maximum m_2 bits, alors
    - Afficher : découpable.
- Si aucune solution n'a été trouvée alors :
  - Afficher : non découpable

FIG. 4.2 – Algorithme permettant de déterminer si un message M est découpable en m_1 et m_2 bits

Grâce à cette méthode, on obtient les résultats suivants :
- Un message de m bits a une certaine probabilité d'être découpable en deux facteurs de m_1 et m_2 bits. Le tableau de la figure 4.3 montre différentes probabilités de découpage en 2 pour différents paramètres.
- Un message de m bits a également une certaine probabilité d'être découpable en trois facteurs de m_1, m_2 et m_3 bits. Le tableau de la figure 4.4 montre différentes probabilités de découpage en 3 pour différents paramètres.

4.2 Influence de la taille du groupe

La taille du groupe est un facteur important pour une attaque. En effet, le temps d'une opération varie énormément en fonction de la taille de ses paramètres. Par exemple, l'exponentielle d'un nombre x exposant e modulo p, avec x et p d'environ n bits est en $0(n^2 \cdot \log e)$. L'attaque utilise une énorme quantité d'exponentielles. Le temps sera donc fortement influencé par le groupe dans lequel les opérations sont effectuées, ici $\mathbb{Z}_p^*$. En résumé, la taille du groupe, définie par p est un facteur important.

Le tableau de la figure 4.5 met en évidence le rapport entre p et le temps d'exécution de l'attaque sur le chiffrement d'ElGamal. Les mesures ont toutes été effectuées sur le même message. Il est composé de 24 bits. Pour les mesures des temps d'attaque, ce message sera découpé en deux parties de 12 bits ou en trois partie de 8,8 et 9 bits selon si l'attaque utilise respectivement 2 tables ou 3 tables.

Un tableau comme celui-ci ne parle pas beaucoup. Pour mieux visualiser l'ensemble, ces données sont représentées sur le graphe de la figure 4.6.

m	m_1	m_2	P(découpable)
40	20	20	11%
	21	21	32%
	22	22	38%
	20	22	36%
	20	25	53%
56	28	28	9%
	29	29	27%
	30	30	33%
	28	30	33%
64	32	32	10%
	33	33	23%
	34	34	32%
	30	36	35%

FIG. 4.3 – Tableau présentant la probabilité qu'un message aie d'être découpable en deux parties

m	m_1	m_2	m_3	P(découpable)
40	13	13	14	1%
	14	14	14	4%
	13	14	15	7%
56	18	19	19	1%
	19	19	19	3%
	20	20	20	5%
64	22	22	22	3%
	23	23	23	5%
	24	24	24	9%
	25	25	25	12%

FIG. 4.4 – Tableau présentant la probabilité qu'un message aie d'être découpable en trois parties

n	q	temps d'attaque avec 2 tables [s]	temps d'attaque avec 3 tables[s]
64	20	0.65	0.09
128	40	1.5	0.21
256	80	2.9	0.43
512	160	9.0	1.2
1024	320	38	5.0

FIG. 4.5 – Tableau présentant le temps des attaques sur ElGamal par rapport à la taille du groupe $\mathbb{Z}_p^*$, p ayant n bits donc un gros facteur de q bits

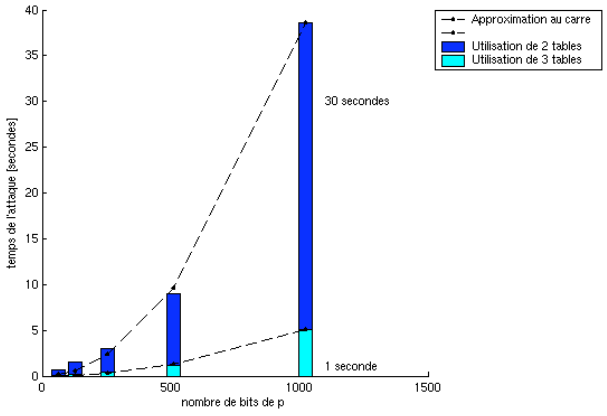

FIG. 4.6 – Graphique montrant le temps des attaques sur ElGamal en fonction de la taille du groupe

4.3 Influence de la taille du message

Afin de bien mettre en évidence l'important rôle que joue la taille du message, une série d'attaques a été réalisée. Chaque attaque a été effectuée avec une taille de message différente. Deplus, pour chaque taille, le message a été retrouvé à l'aide de deux, puis de trois tables. Ainsi il est possible de comparer la différence de temps entre une attaque à deux ou trois tables.

Finalement les probabilités de réussites ont été calculées pour chacune de ces attaques. On peut ainsi se rendre compte de l'importance du nombre de tables. Comme expliqué précédemment, il est évident que plus il y a de tables, moins de chance aura le message d'être découpable, plus la probabilité de réussite de l'attaque sera faible.

Le tableau de la figure 4.7 présente les résultats pratiques des attaques sur ElGamal alors que celui de la figure 4.8 présente les résultats pratiques des attaques sur RSA. Pour les attaques à deux tables, le message de m bits a été découpé en deux parties de $m/2$ bits. Pour les attaques à trois tables, le message de m bits a été découpé en trois parties de $m/3$ bits.

Des tableaux comme ceux-ci ne parlent pas beaucoup. Pour mieux visualiser l'ensemble, ces données sont représentées sur le graphe de la figure 4.9.

m	temps d'attaque avec 2 tables	P(réussite) avec 2 tables	temps d'attaque avec 3 tables	P(réussite) avec 3 tables
6	8 ms	14%	-	0%
12	80ms	14%	20ms	1.5%
18	800ms	14%	50ms	1.0%
24	7,8s	13%	500ms	1.0%
30	1m 30s	12%	3,2s	1.0%
36	13m	10%	16s	0.5%
42	2h	10%	2m50s	0.5%
48	22h	9%	30m3s	0.5%
56	*long...*	9%	16h	0.5%

FIG. 4.7 – Tableau présentant le temps des attaques sur ElGamal par rapport à la taille du message, ici de m bits

m	temps d'attaque	P(réussite)
18	20ms	14%
24	270ms	13%
30	2.2s	12%
36	18s	10%
48	21m	9%
56	19h	9%

FIG. 4.8 – Tableau présentant le temps des attaques sur RSA par rapport à la taille du message, ici de m bits (2 tables)

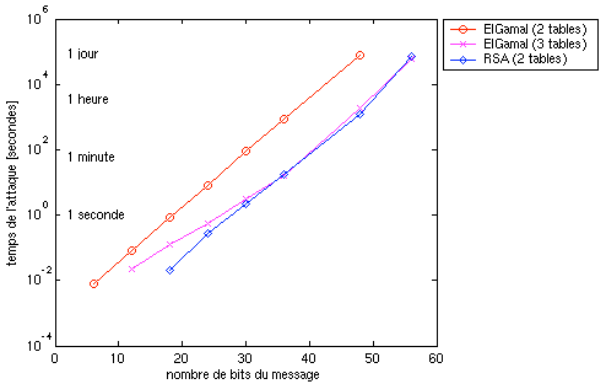

FIG. 4.9 – Graphique montrant le temps des attaques en fonction de la taille du message

4.4 Etude de l'attaque sur ElGamal

Les différentes étapes d'une attaque sur ElGamal ont été décrites à la section 3.6. Afin de connaître le temps nécessaire à chacune, l'heure est affichée avant d'exécuter les opérations d'une phase important de l'attaque. De cette manière, il est possible de repérer les étapes nécessitant beaucoup de temps. Sur la figure 4.10, l'exécution d'une attaque sur un message de 42 bits en utilisant deux tables est décortiquée tandis que sur la figure 4.11 l'attaque utilise trois tables.

1.	Lecture des paramètres p et g, *parametres.elgamal*	< 1s
2.	Factoriser l'ordre du groupe $(p-1)$	≈ 1s
3.	Extraire s, les petits facteurs de $p-1$ tels que $s \geq 2^m$	< 1s
4.	Déduire qr, $p-1 = q \cdot r \cdot s$	< 1s
5.	Chercher w générateur de G_s	< 1s
6.	Remplir les tables $\rightarrow$ ($O(2^{21})$ logarithmes discrets)	62m
7.	Trier les tables	7s
8.	Lecture du message chiffré $\langle u, v \rangle$, *message.elgamal*	< 1s
9.	Déterminer la cible	< 1s
10.	Résoudre le problème du sac à dos (2 tables)	≈ 1s
11.	Rechercher les M_i correspondants aux $\log_{w^{qr}}(M_i^{qr})$	55m
12.	Afficher les résultats (candidats du message)	< 1s
	Au total :	1h57

FIG. 4.10 – Tableau montrant le temps passé par l'attaque dans chaque phase pour la recherche d'un message de 42 bits en utilisant deux tables

1.	Lecture des paramètres p et g, *parametres.elgamal*	< 1s
2.	Factoriser l'ordre du groupe $(p-1)$	≈ 1s
3.	Extraire s, les petits facteurs de $p-1$ tels que $s \geq 2^m$	< 1s
4.	Déduire qr, $p-1 = q \cdot r \cdot s$	< 1s
5.	Chercher w générateur de G_s	< 1s
6.	Remplir les tables $\rightarrow$ ($O(2^{14})$ logarithmes discrets)	43s
7.	Trier les tables	< 1s
8.	Lecture du message chiffré $\langle u, v \rangle$, *message.elgamal*	< 1s
9.	Déterminer la cible	< 1s
10.	Résoudre le problème du sac à dos (3 tables) $\rightarrow$ ($O(2^{14})$ résolutions de problèmes à 2 tables)	84s
11.	Rechercher les M_i correspondants aux $\log_{w^{qr}}(M_i^{qr})$	42s
12.	Afficher les résultats (candidats du message)	< 1s
	Au total :	2m50s

FIG. 4.11 – Tableau montrant le temps passé par l'attaque dans chaque phase pour la recherche d'un message de 42 bits en utilisant trois tables

Grâce à ces deux tables, il est possible d'identifier rapidement les étapes coûteuses de l'attaque.

Dans un attaque à deux tables, seuls le calcul des exponentielles et logarithmes discrets combinés est long. Ceci touche donc uniquement les étapes 6. et 11. de la figure 4.10. Il est important de remarquer que la résolution du problème du sac à dos à deux tables est, dans ce cas, extrêmement faible.

En ce qui concerne les attaques à trois tables, il est possible de remarquer l'inverse. En effet, la partie la plus longue est la résolution du problème du sac à dos à trois tables. Ceci est compréhensible, car il faut se rappeler que cette opération est réalisée en résolvant $O(2^{m_1})$ problèmes à deux tables, soit ici $O(2^{14})$. La partie du remplissage des tables est tout de même toujours relativement longue.

En comparant ces deux attaques, il est clair qu'au niveau temps, l'attaque à trois tables est plus performante. Au niveau mémoire, l'attaque à trois tables est également meilleure, $O(3 \cdot 2^{14})$ contre $O(2 \cdot 2^{21})$. Comme expliqué précédemment, l'énorme inconvénient d'une attaque à trois tables est la probabilité de réussite. Pour ce cas, en effet, l'attaque à trois tables à moins d'un pourcent de chance de réussite, tandis que l'attaque à deux tables en a aux alentours de dix.

4.5 Qu'en est-il de simple DES ?

Au début de ce projet, l'objectif, ou la motivation, fixé était de pourvoir retrouver une clé DES, donc de 56 bits, transmise en utilisant la version "textbook" d'ElGamal ou de RSA. Ceci ne fut pas une chose facile, mais cela a quand même été réalisé.

Supposons, que l'attaque est réalisée sur la communication effectuée à l'aide du chiffrement symétrique DES. Il existe beacuoup d'attaques sur DES, mais pour ne pas entrer dans les détails et sortir du cadre de ce travail, supposons que l'attaqueur soit un débutant et qu'il ne connaisse pas d'attaque spécifique à DES. Il lui faudrait alors au moins une paire (message clair, message chiffré) pour tenter de retrouver la clé. Même avec ces informations, qui de plus ne sont pas facile à obtenir, il faudrait tester chaque clé une à une. Ceci impose d'effectuer en moyenne $O(2^{55})$ chiffrements DES. Pour le moment, une attaque d'une telle complexité est très longue.

Supposons maintenant que l'attaque ne soit plus réalisée sur la communication elle-même, mais sur la transmission de la clé. Supposons de plus que la clé soit transmise comme souhaité dans ce travail, c'est-à-dire en chiffrant la clé directement avec la version "textbook" d'ElGamal. L'attaque peut alors être réalisée avec les techniques présentées dans ce document. Comme vu auparavant, il suffit d'obtenir la clé publique du destinataire, qui en principe est, comme son nom l'indique, publique, ainsi que le message chiffré, qui lui peut-être obtenu en écoutant le réseau. Supposons que l'attaque soit effectuée en utilisant trois tables avec comme paramètres $m_1 = 18$, $m_2 = m_3 = 19$. L'attaque nécessite alors un temps en $O(2 \cdot 2^{19})$ exponentielles modulaires et logarithmes discrets et $O(2^{18} \cdot 2 \cdot 2^{19}) \equiv O(2^{38})$ additions. Le tout représente environ 16 heures sur la machine *cluster64*. Elle utilise une mémoire en $O(2^{18} + 2^{19} + 2^{19})$ éléments de $\log_2(p) = 512$ bits, ce qui représente environ 80 MB, soit 2% de la mémoire disponible sur *cluster64*.

Cette attaque paraît maintenant très avantageuse. Elle a cependant un point faible : la probabilité de réussite. En effet, pour que cette attaque réussisse, il est nécessaire (et suffisant) que le message clair soit découpable en trois parties de maximum m_i bits. La probabilité de réussite avec les paramètres ci-dessus est d'environ 1%. Cette valeur peut cependant être accrue en augmentant la taille des tables.

Chapitre 5

Conclusion

Dans ce document, il est démontré que les versions théoriques des chiffrements d'ElGamal et de RSA ne sont pas sûres dans certains cas. Ici, il s'agissait de prouver que le chiffrement d'un petit message n'est pas une bonne chose lorsque l'algorithme à clé publique n'est pas utilisé correctement. En effet, uniquement à partir du message chiffré et de la clé publique, il est possible de retrouver le message original. Les attaques présentées ont été illustrées en les appliquant au cas de la transmission de clés symétriques en utilisant des moyens de cryptographie asymétriques. Ce type de messages a l'avantage d'être souvent court et de plus d'avoir une taille connue.

Les attaques de ce document font un compromis entre le temps d'exécution et la mémoire nécessaire. Pour un message de m bits, il est souvent possible de le retrouver en un temps évoluant approximativement en $O(2^{m/2})$, voir $O(2^{m/3})$, pour une attaque de 2, respectivement 3, tables. La mémoire utilisée serait alors en $O(2^{m/2})$ et $O(2^{m/3})$ respectivement. L'inconvénient majeur d'une attaque utilisant 3 tables est la faible probabilité de réussite.

Une série d'attaques a été réalisée afin de montrer le temps d'exécution pour différentes longueurs de messages. L'objectif fixé au début de ce projet était de pouvoir retrouver une clé de 56 bits afin de simuler une attaque sur DES. L'attaque sur le chiffrement d'ElGamal a permis de retrouver une clé DES en environ 16 heures de calculs. Cette attaque était basée sur la découpe du message en trois parties, donc utilisait trois tables. La probabilité de réussite est cependant le point faible de la méthode, dans ce cas de l'ordre de 1%. L'attaque sur le chiffrement de RSA a permis de retrouver une clé DES en 19h. La probabilité de réussite est nettement plus grande que la précédente, de l'ordre de 8%. Cette valeur s'explique par le fait que l'attaque sur RSA utilise seulement deux tables.

Maintenant, il est possible de mettre en évidence l'importance du pré-traitement et donc des standards de chiffrements. Grâce à eux, ce type d'attaques n'a aucun effet. Lors de la conception d'un système de communication, il est donc recommandé de ne pas utiliser les versions théoriques de ces chiffrements, mais plutôt d'utiliser un standard comme par exemple RSA-OAEP[7].

Bibliographie

[1] D. Boneh, A. Joux, P. Q. Nguyen : Why textbook ElGamal and RSA encryption are insecure, ASIACRYPT 2000

[2] Serge Vaudenay : Communication Security : An Introduction to Cryptography

[3] Douglas Stinson, traduction de Serge Vaudenay : Cryptographie, Théorie et pratique

[4] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone : Handbook of applied cryptography

[5] T. ElGamal : A public key cryptosystem and a signature scheme based on the discrete logarithm, IEEE Trans. on Information Theory, 31(4) :469-472, 1985

[6] R. L. Rivest., A. Shamir, L-M- Adleman : A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, 21(2) :120-126, 1978

[7] PKCS1 : Public Key Cryptography Standard No. 1 Version 2.0, RSA Labs

[8] R.Schroeppel, A. Shamir : A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems, SIAM J-Comput., 10(3) :456-464, 1981