Cours 5-6: Bases de la cryptographie: Confidentialité asymétrique .

Michaël Quisquater (Maître de Conférences,UVSQ)

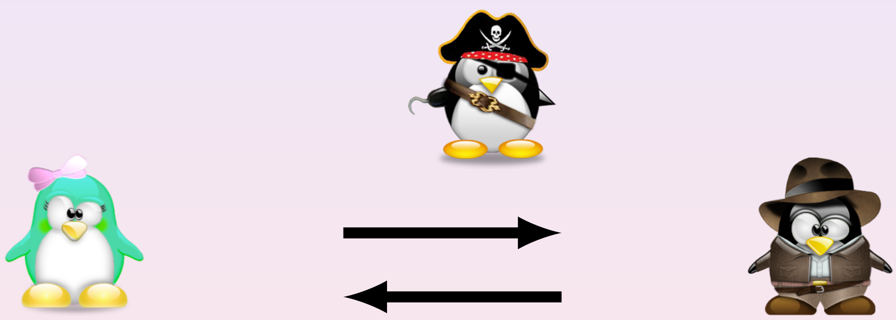
Première partie I

Motivation à la cryptographie asymétrique (ou à clé publique)

Plan de la première partie

Confidentialité

S'assurer du caractère secret de l'information



Stockage de données sécurisé

Cryptographie au sein d'un réseau ?

Dans le cas de la confidentialité par exemple, les solutions basées sur la cryptographie d'hier nécessitent que chacun dispose de $n-1$ clés (groupe de n personnes).

Cela signifie $C_n^2 = (n-1)n/2$ paires de clés différentes.

Difficulté de gestion ! !

Cryptographie au sein d'un réseau ? (suite)

On va donc devoir considérer un cadre solution additionnel à celui de la cryptographie d'hier :

Hypothèses :

- De nombreux intervenants en présence d'un pirate.
- Les intervenants ne peuvent pas toujours se rencontrer avant l'ensemble des échanges afin de se mettre d'accord sur un ensemble de paramètres.

Deuxième partie II

Notion d'algorithme, de problème difficile et de fonction à sens unique

Plan de la seconde partie

Première stratégie

Whitfield Diffie est le premier à avoir compris les problèmes que posaient la gestion de clé. Il a proposé en collaboration avec Martin Hellman diverse tratégie pour pallier à ce problème ("New direction in cryptography", 1976)

Permettre l'échange de clé

Leur solution est basée sur la notion de fonction à sens unique.

Notion de fonction à sens unique

Fonction à sens unique : Pour une relation $y = f(x)$, calculer y est "facile", mais retrouver une preimage de y est "difficile".

Où trouver les outils mathématiques pour contruire de telles fonctions ?

Théorie des nombre, inutile ?

"I have never done anything 'useful'. No discovery of mine has made, or is likely to make, directly or indirectly, for good or ill, the least difference to the amenity of the world."

"Je n'ai jamais rien fait 'd'utile'. Aucune de mes découvertes a apporté, ou probablement apportera, directement ou indirectement, le moindre bénéfice au monde, quoi qu'il arrive."

Issu du livre de G.H. Hardy (théoricien des nombres d'Oxford) intitulé "A Mathematician's Apology" (1940).

Candidat 1 : la multiplication/factorisation

Exemple1 : la multiplication/factorisation

- $(p, q) \mapsto n = p \cdot q$ est "facile" à calculer (quadratique en la taille des nombres)

Par contre,

- $n = p \cdot q \mapsto (p, q)$ est "difficile" (p, q premiers).

Candidat 2 : exponentiation/logarithme

Exemple2 : exponentiation/logarithme

- $\mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^* : x \mapsto \alpha^x \bmod p$ est "facile" (voir plus loin) à calculer (α générateur)

Par contre,

- $\mathbb{Z}_p^* \rightarrow \mathbb{Z}_{p-1} : y = \alpha^x \bmod p \mapsto x$ est "difficile" à calculer

Comment construire une fonction à sens unique

Qu'appelle-t'on "problème facile et difficile ?"

$\longrightarrow$ Notion de problème
$\longrightarrow$ Notion d'algorithme
$\longrightarrow$ Notion de complexité d'un algorithme
$\longrightarrow$ Notion de comparaison de complexités d'algorithmes ?

Qu'est-ce qu'un problème ?

Définition

Un problème Π est une question à plusieurs paramètres dont les valeurs ne sont pas spécifiées.

Exemple : Quelle est la valeur du produit scalaire de deux vecteurs à coefficients dans $\mathbb{Z}_2$.

Qu'est-ce qu'un algorithme ?

Définition

(informelle) Assemblage d'instructions à suivre pour obtenir l'exécution d'une tâche donnée.

Remarques : Le mot "algorithme" vient du nom du mathématicien iranien : Al kharezmi (820 ap. J-C). Ce mathématicien publia un traité d'arithmétique qui permit la diffusion en Occident de règles de calcul sur la représentation décimale des nombres (découvert par les Indiens).

Qu'est-ce qu'un algorithme ? (suite)

Exemples

- recettes de cuisine
- partitions musicales
- Algorithme d'Euclide $\longrightarrow$ pgcd (3ème s. av. J-C)
- Algorithme d'Archimède $\longrightarrow$ approximation de π (3ème s. av. J-C)
- Algorithme de Diophante d'Alexandrie (3ème s. ap. J-C) $\longrightarrow$ résoudre des équations en nombres entiers (exemple relation de Bézout)
- Algorithme de multiplication de matrices de $\mathbb{R}^{n\times n}$.

Qu'est-ce qu'un algorithme ?

Définition

On dit qu'un algorithme résoud un problème Π si pour chaque valeurs des paramètres de Π l'algorithme renvoie la solution à la question posée si il y en a une et dit qu'elle n'en existe pas sinon.

Exemple : Equation diophantienne. L'algorithme d'Euclide étendu permet de dire si $ax + by = c$ $(x, y, a, b, c \in \mathbb{Z})$ ne possède pas de solution ou bien la trouver.

Qu'est que que la complexité d'un algorithme ?

Exemple : Produit scalaire de vecteur de $\mathbb{Z}_2^n$.

$$A = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \quad B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Le produit scalaire de ces vecteurs $c = A^T \cdot B$ sévalue par la formule $\sum_{k=1}^{n} a_k b_k$.

Nombre de bits pour représenter les données du problème (A et B) : $2n$.
Nombre d'opérations pours calculer ce produit scalaire : n produits et $n-1$ additions $\rightarrow 2n-1$ opérations.

Qu'est que que la complexité d'un algorithme ? (suite)

Remarques :

- La constante n'a pas de sens (dépend de l'unité choisie)
- 1 est très petit par rapport à n

$\rightarrow$ introduction d'un concept qui permet de capturer "l'essentiel" de la complexité d'un algorithme.

Qu'est que que la complexité d'un algorithme ? (suite)

Définition

Soit $g : \mathbb{N} \rightarrow \mathbb{R}$ une application. On note $\mathcal{O}(g(n))$ l'ensemble $\{f : \mathbb{N} \rightarrow \mathbb{R} \mid$ il existe $c, n_0 \in \mathbb{N}$ tels que

$$0 \leq f(n) \leq c \cdot g(n)$$

pour tout $n \geq n_0\}$.

Exemple : Soit $f(n) = 2 \cdot n - 1$, $n \in \mathbb{N}$. $f \in \mathcal{O}(n)$. Par conséquent, la complexité de l'algorithme utilisé précédemment est en $\mathcal{O}(n)$.

Qu'est que que la complexité d'un algorithme ? (suite)

Définition

Considérons un problème Π paramétrisé par $i \in I$ (ce sont les données du problème). Notons $n \in \mathbb{N}$ le nombre de bits nécessaires à la représentation des données i.

Si le nombre d'opérations d'un algorithme pour résoudre chaque instance de ce de problème est en $\mathcal{O}(n^t)$ où $t \in \mathbb{R}_0^+$ (resp. $\mathcal{O}(\exp(\alpha n))$, $\alpha > 0$) alors cet algorithme est dit polynomial (resp. exponentiel).

Qu'est que que la complexité d'un algorithme ? (suite)

Remarques :

- On choisira en pratique le plus petit t ou α possible pour décrire la complexité de l'algorithme.
- Si l'exposant $t = 1$ on dit que l'algorithme est linéaire.
- Si la complexité n'est pas polynomiale, on dit que l'algorithme est super-polynomiale
- Si la complexité $f(n)$ n'est pas polynomiale ($f(n) \notin \mathcal{O}(n^t)$) mais est également en dessous de l'exponentiel ($exp(\alpha \cdot n) \notin \mathcal{O}(f(n))$), on dit que l'algorithme est sous-exponentiel.

Comment comparer la complexité d'algorithmes ?

Complexité exacte d'un algorithme ? : souvent inconnue ! (on connait juste un algorithme mais on ne sais pas si c'est le meilleur).

Idée : Créer des classes d'algorithme dont les complexités sont considérées comme équivalentes.

$\longrightarrow$ Notion de réduction.

Comment comparer la complexité d'algorithmes ?

Définition

Soit Π_1 et Π_2 deux problèmes.

- Π_1 *est dit polynonialement réductible à* Π_2 *s'il existe un algorithme résolvant* Π_1 *qui utilise comme sous-routine un algorithme B résolvant* Π_2 *et qui fonctionne en temps polynomial si B fonctionne en temps polynomial. On note alors* $\Pi_1 \leq_P \Pi_2$.
- Π_1 *et* Π_2 *sont dits calculatoirement équivalent si* $\Pi_1 \leq_P \Pi_2$ *et* $\Pi_2 \leq_P \Pi_1$.

Comment comparer la complexité d'algorithmes ?

Remarques :

- On suppose que B est polynomial même si c'est peut-être pas le cas. Le but est de modéliser le fait que la "transformation" soit polynomiale.
- On remarque tous les problèmes polynomiaux sont calculatoirement équivalents. Ces problèmes sont considérés comme faciles.
- Les problèmes "NP-Complets" sont tous calculatoirement équivalents mais il n'existe pour aucun d'eux un algorithme polynomial connu. Les problèmes pour lesquels on ne connait pas d'algorithmes polynomiaux sont considérés comme difficiles.

Notion de fonction à sens unique

On peut réexprimer la définition de fonction à sens unique en des termes plus précis :

Définition

Une fonction $f : X \rightarrow Y$ est une fonction à sens unique si il existe un algorithme polynomial pour l'évaluation de f mais il n'existe pas de d'algorithme polynomial (probabiliste...) pour calculer une preimage de f.

Troisième partie III

Echange de clé et algorithme de chiffrement asymétrique El Gamal

Plan de la troisième partie

Notion de fonction à sens unique : candidat2

Reprenons notre second candidat que nous généralisons un peu :
Soit $(G, \cdot)$ un groupe cyclique, $g \in G$ un générateur. Dénotons E par $E = \{0, \cdots, |G| - 1\}$.
La fonction exponentiation est définie par :

$$exp_g : E \to G : x \mapsto g^x .$$

La fonction logarithme discret est l'inverse de cette application et est définie par

$$Log_g : G \to E : g^x \mapsto x$$

Notion de fonction à sens unique : candidat2

Pour que cela soit un bon candidat, il faut montrer que la fonction exp_g peut être évaluée en temps polynomial et que la fonction Log_g ne peut pas être évaluée en temps polynomial.

Il s'agit du cas général. Bien sûr, il y a autant de candidats "exponentiation/logarithme discrets" possibles, qu'il y a de groupes cycliques. Ex. $\mathbb{Z}_p^*$, $\mathbb{F}_q^*$ (où $\mathbb{F}_q$ est un corps fini à q éléments), groupe sur les courbes elliptiques etc...

Nous allons d'abord montrer que l'évaluation de la fonction exp_g est est un problème polynomial si l'opération de groupe est un problème polynomial.

Exponentiation rapide : tentative

Considérons un ensemble G muni d'une opération binaire multiplicative $\cdot$, un naturel $d \in \mathbb{N}$ et $x \in G$.

Calcul de $x^d = x \cdot x \cdot x \cdots x$

Méthode élémentaire : $d - 1$ multiplications

Remarque : en cryptographie, d est très grand et donc ce n'est pas praticable.

Tentatives d'amélioration

Calcul de 2^6.

Par la méthode élémentaire : 5 multiplications.

Idée : $6 = 2 \cdot 3$. Par conséquent, $2^6 = (2^3)^2$.

$\rightarrow$ 2 multiplications + 1 multiplication (carré) : 3 opérations au total.

Remarque : Notons que cette méthode fonctionne car 6 est factorisable en le produit de 2 et de 3.

Amélioration : Première méthode (intuition)

Calcul de 2^5.

Méthode de base : 4 multiplications et la méthode précédente ne fonctionne plus.

Idée : décomposition binaire de l'exposant : $5 = 2^2 + 0 \cdot 2^1 + 2^0$.

$$2^5 = 2^{2^2} \cdot 2^{2^0}.$$

Il suffit de calculer $2^{2^0} = 2$, $2^{2^1} = 2^2$ et $2^{2^2} = (2^2)^2$.

$\rightarrow$ 2 multiplications

Ensuite multiplier 2^{2^0} et 2^{2^2}

$\rightarrow$ 3 multiplications au total au lieu de 4.

Amélioration : Première méthode (formalisation)

bits faibles → forts

Décomposer l'exposant en binaire :

$$d = d_k 2^k + d_{k-1} 2^{k-1} + \cdots + d_1 2 + d_0$$

A chaque étape, multiplication éventuelle par une puissance x^{2^i}

"*square-and-multiply*"

$$x^d = (x^{2^k})^{d_k} \cdot (x^{2^{k-1}})^{d_{k-1}} \cdots (x^{2^1})^{d_1} \cdot (x^{2^0})^{d_0}$$

Remarque : la séquence des puissances x^{2^i} peut être calculée efficacement car $x^{2^i} = (x^{2^{i-1}})^2$.

"Square-and-multiply" : bits faibles → bits forts

Algorithme 1 : "Square-and-multiply"

Données : $x \in G$, $d \in \mathbb{N}$.
Résultat : x^d
$d = \sum_{i=0}^{k} d_i \cdot 2^i$ (avec $d_k = 1$), $temp := 1$. $puiss = x$
pour $i = 0, \ldots, k$ **faire**
  si $d_i = 1$ **alors**
    $temp := temp \cdot puiss$
  fin
  $puiss := puiss^2$
fin
retourner $temp$

Premirère méthode (exemple)

Calculons $[7 + 11\mathbb{Z}]^5$ ou de façon équivalente $7^5 \bmod 11$.

Initialisation : $5 = 2^2 + 2^0$ ($d_2 = 1$, $d_1 = 0$ $d_0 = 1$). $k = 2$
$x = 7$, $temp = 1$ et $puiss = 7$.

itération 0 : $d_0 = 1$ $temp = temp \cdot puiss = 1 \cdot 7 \bmod 11 = 7$,
$puiss = puiss^2 \bmod 11 = 7^2 \bmod 11 = 5$

itération 1 : $d_1 = 0$ $-$,
$puiss = puiss^2 \bmod 11 = 5^2 \bmod 11 = 3$

itération 2 : $d_2 = 1$ $temp = temp \cdot puiss = 7 \cdot 3 \bmod 11 = 10$,
$puiss = puiss^2 \bmod 11 = 3^2 \bmod 11 = 9$

Conclusion : $[7 + 11\mathbb{Z}]^5 = [10 + 11\mathbb{Z}]$

Fonction à sens unique : Exponentiation/logarithme
Protocole d'échange de clé : Diffie-Hellman
Algorithme de chiffrement asymétrique
Algorithme de chiffrement El Gamal
Introduction
Exponentiation rapide et complexité
Logarithme discret et sa complexité
Logarithme discret : calcul d'indices

Amélioration : Seconde méthode (intuition)

Calcul de 11^{13}.

Méthode élementaire : 12 multiplications.

Idée : Séquence de divisions Euclidiennes (division par 2)

$$13 = 2 \cdot 6 + 1, 6 = 2 \cdot 3 + 0, 3 = 2 \cdot 1 + 1$$

On calcule $11^3 = (11)^2 \cdot 11^1$, $11^6 = (11^3)^2$ et $11^{13} = (11^6)^2 \cdot 11$.

$\rightarrow$ 5 multiplications.

Seconde méthode (formalisation)

bits forts $\longrightarrow$ faibles

Décomposer l'exposant en binaire :

$$d = d_k 2^k + d_{k-1} 2^{k-1} + \cdots + d_1 2 + d_0$$

$$x^d = ((((x^{d_k})^2 \cdot x^{d_{k-1}})^2 \cdot x^{d_{k-2}} \ldots)^2 \cdot x^{d_1})^2 \cdot x^{d_0}$$

A chaque étape, élévation au carré et éventuellement multiplication par x.

"*square-and-multiply*"

"Square-and-multiply" : bits forts $\longrightarrow$ bits faibles

Algorithme 2 : "Square-and-multiply"

Données : $x \in G$, $d \in \mathbb{N}$.
Résultat : x^d
$d = \sum_{i=0}^{k} d_i \cdot 2^i$ (avec $d_k = 1$), $temp := 1$.
pour $i = k, \ldots, 0$ **faire**
  $temp := temp^2$
  si $d_i = 1$ **alors**
    $temp := temp \cdot x$
  fin
fin
retourner *temp*

Seconde méthode (exemple)

Calculons $[7 + 11\mathbb{Z}]^5$ ou de façon équivalente 7^5 mod 11.

Initialisation : $5 = 2^2 + 2^0$ ($d_2 = 1$, $d_1 = 0$ $d_0 = 1$). $k = 2$.
$x = 7$ et $temp = 1$.

itération 2 : $temp = temp^2 \bmod 11 = 1$
$d_2 = 1$ $temp = temp \cdot x = 1 \cdot 7 \bmod 11 = 7$,

itération 1 : $temp = temp^2 \bmod 11 = 7^2 \bmod 11 = 5$
$d_1 = 0$ $-$,

itération 0 : $temp = temp^2 \bmod 11 = 5^2 \bmod 11 = 3$
$d_0 = 1$ $temp = temp \cdot x = 3 \cdot 7 \bmod 11 = 10$,

Conclusion : $[7 + 11\mathbb{Z}]^5 = [10 + 11\mathbb{Z}]$

Complexité des méthodes d'exponentiation rapide

Les deux méthodes nécessitent de réaliser au plus $2 \cdot log_2\ d$ ($log_2\ d$ est la taille de d) opérations dans G.

Conclusion : Si l'opération de G est polynomiale (en la taille des éléments de G), cela signifie que l'exponentiation est une opération polynomiale.

Complexité du logarithme discret

Il reste à montrer l'évaluation du logarithme discret n'est pas un problème polynomial.

Exprimons le problème formellement :

DLG (Logarithme discret généralisé)

Etant donné un groupe cyclique $(G, \cdot)$, un générateur $g \in G$ et $\beta \in G$, calculer $a \in \{0, \cdots, |G| - 1\}$ tel que $g^a = \beta$.

Complexité du logarithme discret

Remarque :

- Il s'agit du cas général. Bien sûr, il y a autant de problèmes de logarithme discret particularisés à un groupe cyclique, qu'il de groupes cycliques. Ex. $\mathbb{Z}_p^*$ (p premier), $\mathbb{F}_q^*$ (où $\mathbb{F}_q$ est un corps fini à q éléments), groupe sur les courbes elliptiques etc...

Logarithme discret : calcul d'indices (suite)

Nous allons décrire un algorithme permettant de résoudre le logarithme discret dans le cas où $G = \mathbb{Z}_p^*$

DG sur $\mathbb{Z}_p^*$

Etant donné $\mathbb{Z}_p^*$ (p premier) , un générateur $\alpha \in \mathbb{Z}_p^*$, $\beta \in \mathbb{Z}_p^*$, calculer $y \in \{0, \cdots, p-2\}$ tel que $\alpha^y = \beta \bmod p$.

Fonction à sens unique : Exponentiation/logarithme
Protocole d'échange de clé : Diffie-Hellman
Algorithme de chiffrement asymétrique
Algorithme de chiffrement El Gamal
Introduction
Exponentiation rapide et complexité
Logarithme discret et sa complexité
Logarithme discret : calcul d'indices

Logarithme discret : calcul d'indices (suite)

Algorithme : Calcul d'indices sur $\mathbb{Z}_p^*$.

Principe : Construire un sous-ensemble d'éléments de $\mathbb{Z}_p^*$ pour lesquels on connait les logarithmes discrets. On exprime ensuite le calcul du logarithme discret particulier que l'on souhaite résoudre en fonction des logarithmes discrets que l'on connaît.

Calcul d'indices : Phase de précalcul

Phase de précalcul :

1. Soit B une base de facteurs. En pratique, il s'agit de l'ensemble des K plus petits nombres premiers.
2. Choisir au hasard $K + L$ éléments $y_i \in_R \{1, \cdots, p-1\}$, $1 \leq i \leq K + L$ où L est un paramètre d'efficacité.

Calcul d'indices : Phase de précalcul (suite)

Phase de précalcul (suite) :

3. Factoriser les nombres α^{y_i} mod p, i.e.

$$\alpha^{y_i} \bmod p = \prod_{j=1}^{K} p_j^{e_{j,i}} \quad (1)$$

avec $e_{i,j} \in \mathbb{N}$ et $p_i \in B$. Si α^{y_i} mod p ne se factorise pas dans la base, il faut choisir un autre y_i.

4. Notons que les équations (1) sont équivalentes à

$$y_i = \sum_{j=1}^{K} e_{j,i} log_\alpha p_j \bmod p - 1 \quad (2)$$

Par conséquent, on peut résoudre le système (si il est déterminé) et obtenir les valeurs de $log_\alpha p_j$.

Calcul d'indices : Phase de calcul (suite)

On suppose que l'on a $log_\alpha p_j$, $j = 1 \cdots K$. On cherche $log_\alpha \beta$.

Phase de calcul

1. Tirer au hasard $y \in_R \{0, 1, \cdots, p-2\}$.
2. Essayer de factoriser $\beta \cdot \alpha^y \bmod p$ dans la base B, i.e.
$$\beta \cdot \alpha^y \bmod p = \prod_{i=1}^{K} p_i^{c_i} \quad (3)$$
avec $c_i \in \mathbb{N}$. Si ce n'est pas le cas, choisir un autre y.
3. Notons que (3) est équivalent à
$$log_\alpha \beta + y = \sum_{i=1}^{K} c_i \cdot log_\alpha p_i \bmod p - 1 \,.$$
4. Finalement, $log_\alpha \beta = -y + \sum_{i=1}^{K} c_i \cdot log_\alpha p_i \bmod p - 1$.

Fonction à sens unique : Exponentiation/logarithme
Protocole d'échange de clé : Diffie-Hellman
Algorithme de chiffrement asymétrique
Algorithme de chiffrement El Gamal
Introduction
Exponentiation rapide et complexité
Logarithme discret et sa complexité
Logarithme discret : calcul d'indices

Logarithme discret : calcul d'indices (suite) : Remarques

- En pratique, K est déterminé de façon optimale par rapport aux propriétés statistiques des facteurs d'un nombre. Au plus K est grand au plus la complexité de l'attaque est importante mais au plus le taux de succès est important (non détaillé ici).
- En pratique, L de l'ordre de 10.
- Avec un choix optimal de K :
  1. complexité (phase de précalcul) : $\mathcal{O}(e^{1+\mathcal{O}(1)(lnp)^{1/2}.(lnlnp)^{1/2}})$
  2. complexité (phase de calcul) : $\mathcal{O}(e^{1+\mathcal{O}(1)(lnp)^{1/2}.(lnlnp)^{1/2}})$

  Une variante de la méthode permet de calculer des logarithmes discrets de nombres de 100 décimaux (350 bits).

Complexité du logarithme discret (suite)

Attention

A l'heure actuelle, nous ne connaissons pas la complexité exacte du problème du logarithme discret sur $\mathbb{Z}_p^*$. On sait juste que le meilleur algorithme connu est super-polynomial mais sous-exponentiel ($\mathcal{O}(exp(c \cdot k^{1/3} \cdot (lnk)^{2/3}))$ où k est la taille de p).

La démarche consiste donc à considérer ce problème comme difficile (sans preuve de cela) et à tenter soit de le résoudre (trouver un algorithme polynomial) soit de le lier à d'autres problème connus.

Echange des clés (Protocole Diffie-Hellman) : primitive

α générateur
p premier

$y_1 = \alpha^x \bmod p$ $\longrightarrow$

$x \in_R \mathbb{Z}_{p-1}$ $\longleftarrow$

$K = (y_2)^x \bmod p$

$y_2 = \alpha^y \bmod p$

$y \in_R \mathbb{Z}_{p-1}$

$K = (y_1)^y \bmod p$

Sécurité de la primitive du protocole de Diffie-Hellman

La sécurité du protocole de Diffie-Hellman semble être basée sur la difficulté du logarithme discret.

Effectuer le protocole revient à effectuer des exponentiations (opération polynomiale)

Une façon triviale de retrouver clé commune revient à calculer x et y à partir des données publiques $\alpha^x \bmod p$ et $\alpha^y \bmod p$ respectivement. Il s'agit du problème du logarithme discret.

Sécurité du protocole de Diffie-Hellman

Plus précisément : Retrouver la clé commune à partir des données publiques revient à résoudre le problème suivant :

DHG (Diffie-Hellman généralisé)

Etant donné un groupe cyclique $(G, \cdot)$, un générateur $g \in G$, g^a et g^b $(a, b \in \mathbb{N})$, calculer $g^{a \cdot b}$.

Sécurité du protocole de Diffie-Hellman (suite)

Remarques :

- Il s'agit du cas général. Bien sûr, il y a autant de problèmes de Diffie-Hellman particularisé à un groupe cyclique, qu'il de groupes cycliques. Ex. $\mathbb{Z}_p^*$, $\mathbb{F}_q^*$ (où $\mathbb{F}_q$ est un corps fini à q éléments), groupe sur les courbes elliptiques etc...
- La complexité du problème DHG et de tout les cas particuliers n'est pas connues

Sécurité du protocole de Diffie-Hellman (suite)

La complexité du problème DHG et de tous les cas particuliers n'est pas connues

Par contre,

DH(G) est polynomialement réductible à DL(H)

Remarque : Idéalement, on aimerait prouver le contraire également de sorte que les DH et DL soient équivalents (problème ouvert !).

Echange des clés (suite)

Attaque : on ne sait pas avec qui on échange la clé !
$\Rightarrow$ Attaque Man-in-the-middle

$y_1 = \alpha^x \bmod p$

$x \in_R \mathbb{Z}_{p-1}$

$K_1 = (y_3)^x \bmod p$

$y_2 = \alpha^y \bmod p$

$y \in_R \mathbb{Z}_{p-1}$

$K_2 = (y_3)^y \bmod p$

Echange des clés (suite)

L'attaque meet-in-the-middle montre que l'on peut attaquer le protocole de Diffie-Hellman sans attaquer le problème DH(G). Cela montre en particulier que le protocole devra être amélioré.

Il manque par conséquent un outil :

- Soit un outil qui permet l'authentification des personnes (voir "cryptographie industrielle")
- Soit une toute autre idée

Stratégie pour apporter une solution au problème

Rendre le mécanisme asymétrique

Chiffrer doit être facile et réalisable par tout le monde et déchiffrer doit être difficile sauf si l'on dispose d'une information complémentaire.

Cette idée a été émise en 1976 par Whitfield Diffie et Martin Hellman ("New direction in cryptography"). Merkle est aussi à l'orgine de l'idée (de façon indépendante)

Concept : fonction à sens unique et à trappe

Définition

Une fonction à sens unique avec trappe est une fonction $f : X \mapsto Y$ telle que l'évaluation soit en complexité polynomial et

- *calculer une preimage de f n'est pas polynomial (probabiliste)*
- *calculer une preimage de f est polynomial si l'on dispose d'une information complémentaire, appelée trappe*

Fonction à sens unique et fonction à trappe (suite)

L'article "New direction in cryptography" de Whitfield Diffie et Martin Hellman s'arrête ici. Il n'avait aucune idée de comment construire de telle fonctions avec trappe.

Etant donné qu'une fonction à sens unique se comporte comme une fonction à sens unique si on ne connaît pas la trappe, une idée est de construire une fonction à trappe à partir d'une fonction à sens unique.

Cryptographie à clé publique : encore un peu d'histoire

R. Rivest, A. Shamir et L. Adelman ont cherché pendant un an une telle fonction.

$\Rightarrow$ publient la primitive RSA en 1977.

Notons que Cocks (Government Communications Headquarters, GCHQ, Royaume-Uni) a trouvé les bases du RSA en 1974 et Williamson retrouva le protocole DH en essayant de casser l'algorithme de Cocks.

Dans un but pédagogique, nous allons faire un saut dans l'histoire et voir l'algorithme de chiffrement asymétrique El Gamal (1984) avant le RSA.

Algorithme de chiffrement El Gamal

Idée :

1. Partager, via le protocole de Diffie-Hellman, une clé aléatoire à usage unique appartenant à l'espace $\mathbb{Z}_p^*$ (p premier)
2. Utiliser cette clé pour masquer un message de l'espace $\mathbb{Z}_p^*$ via la technique du masque jetable (Vernam).

Algorithme de chiffrement El Gamal

$r \in_R \mathbb{Z}_{p-1}$

$m \in \mathbb{Z}_p^*$

α générateur

p premier

$y_2 = \alpha^y \bmod p$

$m = c \cdot ({y_1}^y)^{-1} \bmod p$

(c, y_1)

$y \in_R \mathbb{Z}_{p-1}$

$y_1 = \alpha^r \bmod p$

$c = m \cdot {y_2}^r \bmod p$

Chiffrement El Gamal

Remarques :

- La clé publique de Bob est (α, p, y_2). Sa clé secrète est y.
- Il s'agit d'un schéma probabiliste (différent de la primitive de chiffrement RSA) ; Alice doit changer de r à chaque message.
- La sécurité est basé sur la difficulté du problème DL.
- le système de chiffrement est malléable : Si (c, y_1) est le chiffré de m, $(\alpha \cdot c, y_1)$ est le chiffré de $\alpha \cdot m$.
- Ce schéma n'utilise pas une permutation à sens unique avec trappe mais le protocole de de DH.
- La taille du chiffré est le double de la taille du message.

Chiffrement El Gamal

Théorème

Inverser El Gamal (trouver m à partir de (c, y_1, y_2)) est calculatoirement équivalent à DH.

Preuve :

- Inverser El Gamal est réductible à DH. Si on peut résoudre DH. On peut retrouver y_2^r à partir de y_1 et y_2. On peut donc retrouver $m = c \cdot (y_2^r)^{-1} \bmod p$.
- DH est réductible à inverser El Gamal. Si on peut retrouver m à partir de $(m \cdot \alpha^{y \cdot r} \bmod p, \alpha^y, \alpha^r)$, il suffit d'appliquer cet algorithme à (R, α^y, α^r) ce qui nous donnera $m = R \cdot (\alpha^{y \cdot r})^{-1}$. Par conséquent, $\alpha^{y \cdot r} = R \cdot m^{-1} \bmod p$.

Quatrième partie IV

Primitive RSA et algorithme de chiffrement

Plan de la quatrième partie

Fonction à sens unique : candidat 2

Reprenons notre premier candidat : multiplication/factorisation.

La fonction *Mult* est une fonction qui associe à un ensemble de nombres premiers leur produit. Cette fonction est de complexité polynomiale étant donné que la multiplication est une opération en $\mathcal{O}(n^2)$, n la taille des nombres à multiplier.

Fonction à sens unique : candidat 2 (suite)

Le fonction inverse est le problème de la factorisation

IFP (Problème de la factorisation entière)

Etant donné un naturel $n \in \mathbb{N} \setminus \{0, 1\}$, trouver les p_i's et les α_i's tels que $n = \prod_i p_i^{\alpha_i}$

Complexité de la factorisation

Méthode de factorisation de Fermat :

"Tout nombre impair se décompose en la différence de deux carrés."

En effet, $n = p \cdot q = (\frac{p+q}{2})^2 - (\frac{p-q}{2})^2$.

L'algorithme consistera donc à essayer différentes valeurs b jusqu'à ce que $n + b^2$ soit un carré que l'on note a^2.
L'algorithme fournit alors une factorisation non-triviale de

$$n = (a+b)(a-b)\,.$$

En général : pas efficace ! Cependant, si les nombres p et q sont trop proches, la méthode devient efficace car il ne faudra pas essayer beaucoup de "b" !

Complexité de la factorisation (suite)

Méthode de factorisation de Dixon :

La méthode de Dixon est une relaxation de la méthode de Fermat.

Idée :

- Au lieu de chercher des entiers a et b tels que $n = a^2 - b^2$, on va chercher deux nombres a et b ($a \neq \pm b$) tels que $a^2 - b^2$ soit un multiple de n ou encore $a^2 = b^2 \bmod n$.
- Si $pgcd(a-b, n) \neq 1$ ou $pgcd(a+b, n) \neq 1$ on aura retrouvé un facteur non trivial de n.

Difficulté : Trouver a et b satisfaisant $a^2 = b^2 \bmod n \rightarrow$ méthode des indices.

Complexité de la factorisation : méthode de Dixon

1. Considérons une base $B = \{p_0, p_1, p_2, \cdots, p_K\}$ l'ensemble des K premiers nombres premiers et $p_0 = -1$.
2. Tirer au hasard des entiers x_i ($i = 1 \cdots L$) et essayer de factoriser $y_i = x_i^2 \bmod n$ (on prend le plus petit représentant en valuer absolue) dans la base B, i.e. $y_i = \prod_{i=1}^{K} p_i^{\alpha_i}$. Si y_i ne se factorise pas on considère un autre x_i
3. A chaque y_i factorisable dans la base, on associe le vecteur des puissances des premiers $(\alpha_1, \cdots, \alpha_n)$. On associe un autre vecteur $c^i = (c_1, \cdots, c_n) = (\alpha_1 \bmod 2, \cdots, \alpha_n \bmod 2)$.

Fonction à sens unique : multiplication/factorisation
Problème RSA
Primitive RSA en chiffrement
Attaques sur la primitive RSA en chiffrement
Standard RSA en chiffrement : PKCS#1 (OAEP)
Méthode de factorisation de Fermat
Méthode de factorisation de Dixon
Méthode de Pollard $p-1$

Complexité de la factorisation : méthode Dixon

1. On cherche ensuite une combinaison linéaire nulle des vecteurs c^i $i = 1 \cdots L$, i.e. $\oplus_{j=1}^{L} \gamma_j c^j$.
2. On a $\prod_i (x_i)^2 = \prod_i y_i \bmod n$ où le produit est pris sur l'ensemble des indices où $\gamma_i = 1$.
3. Par construction $\prod_i y_i$ est un carré et donc on a trouve deux nombres $a = \prod_i (x_i)$ et $b = \sqrt{\prod_i y_i}$ tels que $a^2 = b^2 \bmod n$.
4. Si $pgcd(a-b, n) \neq 1$ ou $pgcd(a+b, n) \neq 1$ (respectivement -1) on a un facteur non trivial de n . Sinon, on cherche d'autre a et b et on recommence.

Complexité de la factorisation : méthode Dixon

En pratique factoriser les y_i peut ne fonctionner correctement que si la base est suffisamment grande (ce qui diminue l'efficacité). La théorie des fractions continues permet de générer des nombres x_i tels que $x_i^2 \bmod n$ soit petit. Ces plus petits nombres ont plus de chance de se factoriser dans la base B de la méthode de Dixon. Cette méthode ne sera pas vue dans ce cours.

Complexité de la factorisation : Pollard $p-1$

Méthode de factorisation de Pollard $p-1$:
Idée générale de l'attaque : Produire des nombres qui ont une bonne chance d'avoir un pgcd non trivial avec le nombre à factoriser, pourvu que les diviseurs premiers de n possèdent certaines propriétes.

Nous avons besoin de la notion suivante :

Définition

Soit B un naturel. Le nombre $n \in \mathbb{N}$ est dit $B-$lisse si $n = \prod_{i \in I} p_i^{e_i}$, avec p_i premier, $p_i \leq B$ et $e_i \in \mathbb{N} \setminus \{0\}$ pour tout $i \in I$

Complexité de la factorisation : Pollard $p-1$

Idée précise de l'attaque :
Supposons que n possède un diviseur premier p tel que $p-1$ soit $B-$lisse pour un B raisonnable.

Définissons $k = ppcm\{p^i \mid p \in P_B \;\; p^i \leq n\}$.

Il est facile de voir que $p-1 \mid k$ car $p-1$ est B-lisse par hypothèse et car on a pris suffisamment de puissance dans k.

Par le théorème de Fermat, pour tout $a \in \mathbb{Z}_p^*$. Donc
$p \mid pgcd(a^k - 1, n)$

Par la propriété du pgcd,
$pgcd(a^k - 1, n) = pgcd((a^k - 1) \bmod n, n)$. Il suffit donc de calculer ce dernier pgcd pour espérer trouver ce diviseur p.

Complexité de la factorisation : Pollard $p-1$

Algorithme de Polard $p-1$:
Input : $n \in \mathbb{N} \setminus \{0\}$
Output : d un diviseur non trivial de n ($\neq 1$ et $\neq n$)

1. Choisir $B \in \mathbb{N} \setminus \{0\}$. Soit P_B l'ensemble des nombres premiers inférieurs ou égal à B.
   $k = ppcm\{p^i \mid p \in P_B \;\; p^i \leq n\}$.
2. Choisir au hasard $a \in_R \{2, \cdots, n-2\}$
3. Calculer $c = a^k \bmod n$
4. Calculer $pgcd(c-1, n) = d$
5. Si d est un diviseur trivial retour étape 2, sinon retourner d.

Complexité de la factorisation (suite)

Complexité de Pollard $p-1$:

- moyenne $\mathcal{O}(n^{1/4})$.
- pire des cas $\mathcal{O}(n^{1/2})$.

Pour éviter l'attaque, il est conseillé d'utiliser de nombres premiers p de la forme $p = 2p' + 1$ òu p' est un nombre premier $\Rightarrow$ "premier sûr".

Fonction à sens unique : multiplication/factorisation
Problème RSA
Primitive RSA en chiffrement
Attaques sur la primitive RSA en chiffrement
Standard RSA en chiffrement : PKCS#1 (OAEP)
Méthode de factorisation de Fermat
Méthode de factorisation de Dixon
Méthode de Pollard $p-1$

Complexité de la factorisation (suite)

Le meilleur algorithme connu mis en oeuvre est le crible algébrique (amélioration du crible quadratique : variante de la méthode de Dixon)

$$\mathcal{O}(e^{(\frac{64}{9} ln\ n)^{1/3} \cdot (ln\ ln\ n)^{3/2}})$$

Factorisation par ordinateur quantique (Shor) : $\mathcal{O}((ln\ n)^3)$ (factorisation de 15 en 2001).

Construction d'une fonction à trappe : Problème RSA

Nous allons à présent le problème RSA (ou problème de la racine *e*ième modulo $n = p \cdot q$).

RSAP

Soit p, q des nombres premiers, $n = p \cdot q$. Dénotons par $\Phi(n)$ la fonction totient d'Euler. Soit $e \in \mathbb{Z}$ tel $pgcd(e, \Phi(n)) = 1$ et soit d tel que $ed = 1 \mod \Phi(n)$.

Etant donné $c = m^e \mod n$ trouver $m = c^d \mod n$

Complexité du problème RSA et le problème de la factorisation

Il est clair que *RSAP* est réductible à *IFP*. En effet, si on peut factoriser n, on peut retrouver p et q et donc calculer $\Phi(n)$. On peut donc retrouver d à partir de e via l'algorithme d'Euclide étendu.

Idéalement on souhaiterait montrer que *IFP* est réductible à *RSAP* mais il s'agit d'une question ouverte.

Construction d'une fonction à trappe

Fonction RSA : Candidat pour une fonction à trappe

Initialisation

Soit

- p et q deux nombres premiers distincts de même taille
- $n = p \cdot q$ (module RSA)
- $e \in_R \mathbb{Z}^*_{\Phi(n)}$ où $\Phi(n)$ est la fonction totient d'Euler

Alors $RSA : \mathbb{Z}_n \to \mathbb{Z}_n : x \mapsto x^e \mod n$ est un bon candidat pour une fonction à sens unique avec trappe.

Fonction à sens unique : multiplication/factorisation
Problème RSA
Primitive RSA en chiffrement
Attaques sur la primitive RSA en chiffrement
Standard RSA en chiffrement : PKCS#1 (OAEP)
Fonction à sens unique avec trappe : RSA

Construction d'une fonction à trappe : Problème RSA

Théorème

Soit

- *p et q deux nombres premiers distincts*
- *$n = p \cdot q$ (module RSA)*
- *$e \in \mathbb{Z}^*_{\Phi(n)}$ où $\Phi(n)$ est la fonction totient d'Euler*

Alors $RSA : \mathbb{Z}_n \to \mathbb{Z}_n : x \mapsto x^e \mod n$ est une permutation sur $\mathbb{Z}_n$ dont l'inverse $RSA^{-1}(y) = y^d \mod n$ où $d = e^{-1} \mod \Phi(n)$.

Construction d'une fonction à trappe : Problème RSA

Preuve :
Soit $n = p \cdot q$ et p, q sont des premiers distincts.

Définissons $Id : \mathbb{Z}_n \to \mathbb{Z}_n : x \mapsto Id(x) = x$. Montrons que $RSA \circ RSA^{-1} = Id = RSA^{-1} \circ RSA$. Dans ce cas, cela signifie que RSA est inversible.

On souhaite montrer que $[m + n\mathbb{Z}]^{e \cdot d} = [m + n\mathbb{Z}]$ pour $m \in \{0, \cdots, n-1\}$.

Comme p et q sont des premiers distincts, $pgcd(p, q) = 1$. Par conséquent, l'anneau $\mathbb{Z}_{p \cdot q}$ est isomorphe à l'anneau $\mathbb{Z}_p \times \mathbb{Z}_q$ par le théorème des restes chinois.

Par conséquent, la thése revient à montrer que $[m + p\mathbb{Z}]^{e \cdot d} = [m + p \cdot \mathbb{Z}]$ et $[m + q\mathbb{Z}]^{e \cdot d} = [m + q \cdot \mathbb{Z}]$.

Construction d'une fonction à trappe : Problème RSA

Preuve : (suite)

Si m est un multiple de p (resp. q), on a bien $[m + p\mathbb{Z}]^{e \cdot d} = [m + p\mathbb{Z}]$ (resp. $[m + q\mathbb{Z}]^{e \cdot d} = [m + q\mathbb{Z}]$).

Si m n'est pas un multiple de p (resp. q), m est relativement premier avec p (resp. q). Aussi, $\Phi(n) = (p-1)(q-1)$ et $e \cdot d = 1 \bmod \Phi(n)$ donc $e \cdot d - 1 = k \cdot (p-1)(q-1)$.

Par le petit théorème de Fermat
$[m + p\mathbb{Z}]^{e \cdot d} = [m + p\mathbb{Z}]^{1 + k \cdot (p-1)(q-1)} = [m + p \cdot \mathbb{Z}]$ (resp.
$[m + q\mathbb{Z}]^{e \cdot d} = [m + q\mathbb{Z}]^{1 + k \cdot (p-1)(q-1)} = [m + q\mathbb{Z}]$).

Le résutat suit.

Fonction à sens unique avec trappe : RSA

Si on connaît d : Evaluer $RSA^{-1}(y)$ est facile (exponentiationn rapide). d est la trappe.

Si on ne connaît pas d : Il s'agit de RSAP dont la complexité n'est pas connue. On sait juste que IFP est au moins aussi difficile que RSAP.

RSA en chiffrement



n, e

$c = m^e \bmod n$ $\longrightarrow$ $m = c^d \bmod n$

- e est l'exposant public de chiffrement,
- d est l'exposant secret de déchiffrement.

Attaques sur la primitive RSA en chiffrement

Malléabilité : Un algorithme de chiffrement est dit malléable si étant donné un chiffré d'un message clair m, il est possible de générer un autre chiffré qui se déchiffre comme $g(m)$, pour une fonction g connue (sans nécessairement connaître m).

La primitive chiffrement RSA est malléable :

$$\text{si } c = m^e \bmod n \text{ alors } c^r = (m^r)^e \bmod n \text{ pour } r \in \mathbb{Z}.$$

Contre-mesure : utilisation d'un padding

Attaques sur le RSA en chiffrement

broadcast : envoi d'un même message à plusieurs personnes.

Contexte de l'attaque :

- Même module RSA ($n = p \cdot q$) pour les 2 personnes.
- $pgcd(e_1, e_2) = 1$
- On dispose $c_1 = m^{e_1} \bmod n$ et $c_2 = m^{e_2} \bmod n$.

Par le th. de Bezout, il existe $x, y \in \mathbb{Z}$ tels que $x \cdot e_1 + y \cdot e_2 = 1$.

Par conséquent, $m = m^1 = m^{xe_1 + ye_2} = c_1^x \cdot c_2^y \bmod n$.

Contre-mesure : imposer des modules différents

Attaques sur le RSA en chiffrement : Broadcast (théorème des restes chinois)

Pour des raisons de rapidité, il arrive souvent que l'on souhaite avoir un exposant public de chiffrement le plus petit possible (exemple : 3).

Supposons que trois personnes aient comme module n_1, n_2 et n_3 (respectivement) et un même exposant public $e = 3$.

Si le même message est chiffré, i.e.

$$c_1 = m^3 \bmod n_1 \quad c_2 = m^3 \bmod n_2 \quad c_3 = m^3 \bmod n_3 .$$

Il est possible de retrouver m en temps polynomial (théorème des restes chinois) : voir TD.

Contre mesure : imposer des messages différents : Padding

Attaques sur le RSA en chiffrement : Attaque de Wiener, Boneh & Durfee . Si d est trop petit

- Si $d < 1/3 \cdot n^{1/4}$ l'attaque de Wiener permet de retrouver d à partir de e et n en temps polynomial via les fractions continues (non vu).
- Si $d < 0.292n$, l'attaque de Boneh & Durfee permet de retrouver d à partir de e et n en temps polynomial via l'algorithme LLL (non vu).

Attaques sur le RSA en chiffrement : attaque par motif

Si on chiffre qu'un nombre restreint de messages (exemple : "OUI" ou "NON"), il n'y aura que deux chiffrés possibles. Par conséquent, une attaque par motif est possible.

Contre mesure : Mettre de l'aléa dans le message que l'on chiffre $\Rightarrow$ Padding.

Attaques sur le RSA en chiffrement : factorisation

Nous avons vu précédemment différents algorithmes de factorisation.

Pour que la factorisation du module RSA ($n = p \cdot q$) ne soit pas possible avec les algorithmes actuels, il faut que le module ait au moins 1024 bits.

De plus, étant donné l'algorithme de Fermat, les premiers p, q ne peuvent pas être trop proches.

Standard RSA : PKCS#1 (OAEP).

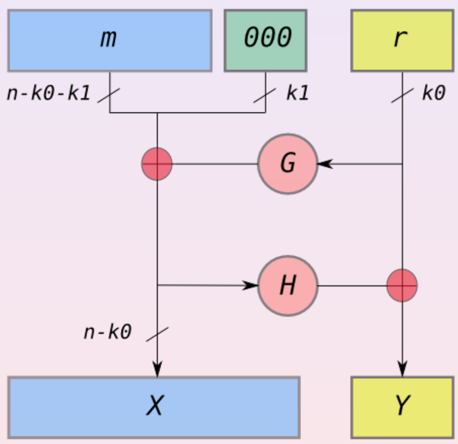
Les attaques précédentes ont montré des faiblesses sur la primitive RSA. Nous avons vu que l'utilisation d'un padding était souvent une contre-mesure efficace.

Un standard de padding a été défini dans la norme (PKCS#1). Il s'agit du padding OAEP.

Pour chiffrer un message, un utilise un aléa r et deux fonctions H et G (dite de hachage : voir cours prochain) et on calcule la valeur suivante :

$$OAEP(m) = m \oplus G(r) \mid\mid (r \oplus H(M \oplus G(r)))$$

Standard RSA : PKCS#1 (OAEP) (suite)



On chiffre ensuite cette valeur.

Standard RSA : PKCS#1 (OAEP) (suite)

Le destinataire déchiffre ensuite $OAEP(m)$ et obtient le message m à partir de $OAEP(m)$ en coupant $OAEP(m)$ en deux morceaux P_1 et P_2 avec $OAEP(m) = P1 \mid\mid P_2$. Il calcule ensuite

$$m = P_1 \oplus G(H(P1) \oplus P_2)$$

Standard RSA : PKCS#1 (OAEP) (suite)

Remarque : La sécurité du RSA utilisé avec le padding OAEP est prouvé sûr dans le modèle dit de "l'oracle aléatoire". Il s'agit d'un modèle idéalisé.

Source des images :

- png factory
- tux factory
- wikipedia