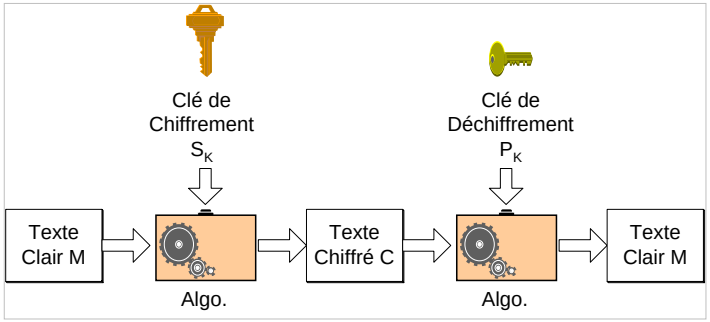
Cryptographie à clé publique

Les systèmes à clé publique

- Systèmes symétriques :
  - même clé pour le chiffrement et le déchiffrement
  - Problèmes :
    - transmission de la clé
    - 1 clé par destinataire
- Système asymétrique :
  - 2 clés distinctes avec impossibilité de déduire l'une de l'autre.
  - Une des clés peut être publiée et distribuée librement (sous certaines conditions)

Schéma général



Application des systèmes à clé publique

- Classification des utilisations en 3 catégories :
- **Chiffrement/déchiffrement** (fournir le secret)
- **Signatures numériques** (fournir l'authentification)
- **Échange de clés** (ou des clefs de session)
- quelques algorithmes conviennent pour tous les usages, d'autres sont spécifiques à un d'eux

Les systèmes à clé publique

- Concept date de 1976 – Diffie et Hellman
- Première implémentation : 1978 – Rivest, Shamir, Adleman (RSA)
- La sécurité de tels systèmes repose sur des problèmes calculatoires
  - RSA : factorisation de grands entiers : NP
  - ElGamal : logarithme discret d'un corps fini : NP
  - Chor – Rivest : problème du sac à dos : NPC
  - …

Sécurité des schémas à clé publique

- La recherche des clés par force brute est toujours théoriquement possible
  - mais les clefs utilisées sont trop grandes (> 512bits)
- la sécurité se fonde sur une assez grande différence en difficulté entre les problèmes faciles (dé/chiffrement) et difficiles (cryptanalyse)
  - plus généralement le problème difficile est connu, il est juste trop difficile à réaliser en pratique
  - exige l'utilisation des nombres très grands
- Constatation : lent comparé aux chiffrements symétriques

Les systèmes à clé publique

- Remarque :
  - Un système à clé publique ne peut jamais être inconditionnellement sûr.
    - Essayer tous les M possibles donnant C (connu) avec la clé publique
    - $\rightarrow$ étude exclusive de la sécurité calculatoire de ces systèmes
- Fonction à sens unique à trappe.
  - Sens unique : E_k facile (injective), D_k difficile
    - Remarque : $\nexists$ fonction démontrée
  - Trappe : la connaissance de la clé de déchiffrement rend la fonction bijective

Systèmes symétriques

Rappels théoriques

Rappels

- Théorème de Fermat :
  - Si p est premier, a>0 et p ∤ a :
    - $a^{p-1} \equiv 1 \bmod p$
    - $a^{p} \equiv a \bmod p$
- Fonction totient d'Euler
  - Si p est premier : $\phi(p) = p - 1$
  - Si p,q sont premier et n=pq : $\phi(n) = (p-1)(q-1)$
- Théorème d'Euler
  - Si (a,n) :
    - $a^{\phi(n)} \equiv 1 \bmod n$
    - $a^{\phi(n)+1} \equiv a \bmod n$

Rappels

- Corollaire du théorème d'Euler
  - Si p,q sont premiers, n=pq et 0 < m < n
    - $m^{\phi(n)+1} = m^{(p-1)(q-1)+1} \equiv m \bmod n$
    - $M^{k\phi(n)+1} \equiv [(m^{\phi(n)})^k * n] \bmod n$
      $\equiv ((1)^k * m) \bmod n$ (par le th d'Euler)
      $\equiv m \bmod n$
- Tests de primalité :
  - Par les tests de Fermat, Solovay-Strassen, …
- Théorème du reste chinois :
  - Permet de manipuler de grand nombre modulo M en termes de tuples de plus petit nombres

Rappels

- Algorithme d'Euclide :
  - Calcul du pgcd
- Algorithme d'Euclide étendu
  - Calcul du pgcd et de l'inverse modulaire

Merkle - Hellman

Introduction

- Définition du problème :
  - Soit un havresac de capacité T et un ensemble S d'objets qui occupent les espaces S = $\{a_1,a_2,a_3,\ldots,a_n\}$
  - Trouver un vecteur de sélection $V=\{v_1,v_2,v_3,\ldots,v_n\}$ satisfaisant la relation $\sum(a_i*v_i) = T$
- Exemple :
  - a = {17, 38, 73 ,4 ,11 ,1} et T = 53 = 38 + 4 + 11 donc V = {0, 1, 0, 1, 1, 0}
  - Pas de solution pour T = 45
  - Si T et a sont très grands, c'est beaucoup plus compliqué

Problème du Sac à dos : empiler différent types d'objets avec chacun un poids différents : faire un nouveau tas d'objet avec un poids total donné

Principe

- Idée de base :
  - coder le message comme une solution d'une suite de problèmes d'empilement.
  - Un bloc de texte en clair de longueur égale au nombre d'objets dans le tas sélectionnerait des objets
    - Les bits du texte en clair correspondraient aux valeurs des v_i : un 1 signifierait que l'objet est présent et 0 un objet absent
  - Et le texte chiffre serait la somme résultante
- Exple :
  - M : 1 1 1 0 0 1 0 1 0 1 1 0
  - Tas : 1 5 6 11 14 20 1 5 6 11 14 20
  - C : 1+5+6+20 = 32 5+11+14 = 30

Empilement et clés

- Il y a 2 problèmes d'empilement
  - 1 soluble en temps linéaire
  - 1 soluble en temps exponentiel
- L'empilement facile peut être transformé pour créer un empilement difficile
- Clé publique : empilement difficile, peut juste chiffrer
- Clé privée : empilement facile qui donne un moyen simple de déchiffrer les messages

Ceux qui ne connaissent pas la clé privée sont obligé de résoudre le problème d'empilement difficile

Empilement facile

- Empilement facile :
  - Si la liste des poids est super croissante (tout terme est plus grand que la somme des termes qui la précède)
  - Solution :
    - prendre le poids total et le comparer avec le plus grand nombre de la suite.
      - Si ce poids total est < à ce nombre, alors il n'est pas dans le tas
      - Si ce poids total est > à ce nombre, alors il est dans le tas
    - Réduire le poids du tas à créer de ce nombre et passer au plus grand nombre suivant de la suite
    - Répéter jusqu'à ce que ce soit terminé
    - Si le poids total a pu être ramené à 0 : il y a une solution

Merkle-Hellman

- Empilement difficile :
  - On ne connaît pas d'algorithme rapide
  - Tester méthodiquement toutes les solutions possibles
  - Ces algorithmes sont exponentiels
- Merkle-Hellman : exploite cette propriété
  - Clé privée : suite de poids supercroissante
  - Clé publique : suite de poids pour un problème d'empilement normal avec la même solution
  - Les auteurs ont trouvé une technique pour transformer un tas supercroissant en un havresac régulier correspondant

Calcul de la clé publique

- Calcul de la clé publique à partir de la clé privée
  - Prendre une suite supercroissante
  - La multiplier par n modulo m avec
    - m plus grand que la somme de tous les nombres de la suite
    - n ne doit avoir aucun facteur commun avec aucun nombre de la suite

Algorithme

- Chiffrement :
  - additionner les termes où un 1 apparaît
- Déchiffrement :
  - Calculer n^{-1} tel que $n*n^{-1} \equiv 1 \bmod m$
  - Multiplier chaque valeur du texte chiffré par $n^{-1} \bmod m$
- En pratique
  - 250 éléments
  - Chaque terme a une longueur de 200 à 400 bits
  - Le module a une longueur de 100 à 200 bits

Exemple

- Générer S qui est une séquence super-croissante de m entiers: par exemple 1,2,4,9
- Choisir un multiplicateur w et un module n
- Soit w = 15 et n = 17
  - 1 * 15 mod 17 => 15
  - 2 * 15 mod 17 => 13
  - 4 * 15 mod 17 => 9
  - 9 * 15 mod 17 => 16
- Le havresac difficile est donc H = {15, 13, 9, 16}

Exemple

- Le message est traité comme une séquence de bits
  - P = [p_1, p_2, p_3, ….. , p_k]
- On le divise en blocs de m bits
  - P_0 = [p_1, p_2, p_3, ….. , p_m], P_1 = [p_{m+1}, p_{m+2}, p_{m+3}, ….. , p_{2*m}], ……..
- On utilise chaque bloc comme vecteur V du problème de havresac
- Soit P = 0100101110100101 => 0100 1011 1010 0101
  - [0, 1, 0, 0] * [15, 13, 9, 16] => 13
  - [1, 0, 1, 1] * [15, 13, 9, 16] => 40
  - [1, 0, 1, 0] * [15, 13, 9, 16] => 24
  - [0, 1, 0, 1] * [15, 13, 9, 16] => 29
- Le message encrypté est donc 13, 40, 24, 29 en utilisant le havresac public (la clef publique) H = [15, 13, 9, 16]

Exemple

- Le destinataire légitime connaît le havresac simple S et les valeurs de w et de n
- Il peut donc déterminer w^{-1}
  - Exemple avec w = 15 et n = 17, w^{-1} est 8: 15 * 8 = 120 = 7 * 17 + 1
- Exemple
  - 13 * 8 mod 17 = 104 mod 17 = 2 = [1, 2, 4, 9] * [0100]
  - 40 * 8 mod 17 = 320 mod 17 = 14 = [1, 2, 4, 9] * [1011]
  - 24 * 8 mod 17 = 192 mod 17 = 5 = [1, 2, 4, 9] * [1010]
  - 29 * 8 mod 17 = 232 mod 17 = 11 = [1, 2, 4, 9] * [0101]
- et le texte en clair est 0100 1011 1010 0101 =>0100101110100101

Sécurité :

- Herlestam (1978) : souvent un bit de texte clair pouvait être retrouvé
- Shamir (1979) : l'algorithme peut être cassé dans certaines circonstances
- Shamir-Zippel (82-83-84) : failles dans la transformation qui permettent de reconstruire la suite super-croissante à partir de la suite normale

RSA (Rivest-Shamir-Adleman)

Présentation

Il est basé sur un chiffrement exponentiel.

Sa sécurité repose sur la fonction unidirectionnelle suivante:

Le calcul du produit de 2 nombres premiers est aisé.

La factorisation d'un nombre en ses deux facteurs premiers est beaucoup plus complexe.

Généralités

- par Rivest, Shamir et Adleman du MIT en 1977
- Système le + connu et le + largement répandu
- basé sur l'élévation à une puissance dans un champ fini sur des nombres entiers modulo un nombre premier
  - Le nombre d'exponentiation prend environ $O((\log n)^3)$ opérations $\rightarrow$ facile
- emploie de grands nombres entiers (par exemple 1024 bits)

Généralités

- Chiffrement : $C = M^e \bmod n$
- Déchiffrement : $M = C^d \bmod n$
- Les deux clés d et e sont interchangeables
- sécurité due au coût de factoriser de grands nombres
  - Le nombre de factorisation prend environ $O(e^{\log n \log \log n})$ opérations $\rightarrow$ difficile
- Usage :
  - confidentialité, authentification, combinaison des 2

Principe

- 2 clés :
  - Clé publique : une paire (e,n)
  - Clé privée : une paire (d,n)
- Première étape : choisir n
  - Valeur assez élevée
  - Produit de 2 nombres premiers très grands p et q
  - Typiquement : p et q ont 100 chiffres décimaux $\Rightarrow$ n a 200 chiffres
  - Taille de n = 512 bits, ou 768 ou 1024 selon le niveau de sécurité souhaité

Principe

- Deuxième étape :
  - Choisir un très grand entier e
    - Relativement premier à (p-1)*(q-1)
      - Astuce : choisir e comme un nombre plus grand que p-1 et q-1
  - Choisir d tel que
    - $ed \equiv 1 \bmod (\phi(n))$
- Troisième étape : on jette p et q

Justification de l'inversibilité

- Par le théorème d'Euler et de Fermat :
  - $a^{\phi(n)} \equiv 1 \bmod n \Leftrightarrow a^{\phi(n)} \bmod n = 1$ où (a,n)=1
- Dans le RSA on a :
  - n = p*q
  - $\phi(n)$ donne le nombre d'entiers positifs plus petits que n et relativement premiers à n
    - Si p est premier, $\phi(p)=p-1$
  - Si n = p*q, avec p et q premiers
    - $\phi(n) = \phi(p)*\phi(q)=(p-1)*(q-1)$

Démonstration

- A cause de la façon de choisir e et d
  - $e*d \equiv 1 \bmod \phi(n) = k*\phi(n) + 1$ pour un certain k
- Puisque p est premier (corollaire d'Euler)
  - $M^{p-1} \equiv 1 \bmod p$; $M^{k*\phi(n)} \equiv 1 \bmod p$; $M^{k*\phi(n)+1} \equiv M \bmod p$
- Même chose pour q
  - $M^{k*\phi(n)+1} \equiv M \bmod q$
- Si on combine :
  - $(M^e)^d = M^{e*d} = M^{k*\phi(n)+1} \equiv M \bmod p = M \bmod q = M \bmod n = M$

Résumé :

1. Génération de 2 nombres premiers p et q
2. Calcul de n = p*q
3. Déterminer e tel que $3 < e < \phi(n)$
4. Calculer d tel que $ed \equiv 1 \bmod \phi(n)$
5. Clés
   - publique : (e,n) privée : (d,n)
   - P et q doivent rester secrets
6. $C = M^e \bmod n$ et $M = C^d \bmod n$ (confidentialité)

Exemple

- Soient p = 11, q = 13 $\rightarrow$ n = 143
- $\phi(n) = 120$ (nombre d'éléments relativement premiers à n et < n)
- e = 23 (on a $(e,\phi(n))$ ce qui permettra de trouver d)
- d = 47 (inverse modulaire de e sur $\mathbb{Z}_n$)
- Clé publique : (23,143) – clé privée : (47,143)

1 – p et q sont bien premiers

2 – soient 2 nbres aléatoires : 61% de chances que relativement premiers entre eux

Remarque : clé de 8 bits !!!!!

Exemple …

- Normalement, les données sont compressées puis découpées en morceaux de taille < n
- Codage pour l'exemple

  1 car = 1 car ASCII :

j	u	s	t	e		u	n		t	e	s	t
106	117	115	116	101	32	117	110	32	116	101	115	116

```
c   =   m^e     mod n
-------------------------------------
c1  =  106^23   mod 143  = 46
c2  =  117^23   mod 143  = 13
c3  =  115^23   mod 143  = 136
c4  =  116^23   mod 143  = 51
c5  =  101^23   mod 143  = 30
c6  =  32^23    mod 143  = 76
c7  =  117^23   mod 143  = 13
c8  =  110^23   mod 143  = 11
c9  =  32^23    mod 143  = 76
c10 =  116^23   mod 143  = 51
c11 =  101^23   mod 143  = 30
c12 =  115^23   mod 143  = 136
c13 =  116^23   mod 143  = 51
```


1 - < n car les opérations effectuées sont des exponentiations modulo n.

Exemple …

- Le déchiffrement s'effectue en utilisant la clé (47,143)

```
m   =   c^d     mod n
-------------------------------------
m1  =  46^47    mod 143  = 106  = 'j'
m2  =  13^47    mod 143  = 117  = 'u'
m3  =  136^47   mod 143  = 115  = 's'
m4  =  51^47    mod 143  = 116  = 't'
m5  =  30^47    mod 143  = 101  = 'e'
m6  =  76^47    mod 143  = 32   = ' '
m7  =  13^47    mod 143  = 117  = 'u'
m8  =  11^47    mod 143  = 110  = 'n'
m9  =  76^47    mod 143  = 32   = ' '
m10 =  51^47    mod 143  = 116  = 't'
m11 =  30^47    mod 143  = 101  = 'e'
m12 =  136^47   mod 143  = 115  = 's'
m13 =  51^47    mod 143  = 116  = 't'
```


Rmq : ca fonctionne car ascii : 0-127 et 127 < 143

N doit être plus grand que les nombres que l'on souhaite chiffrer….

Remarques

- Il n'est pas très astucieux de choisir d'aussi petites valeurs :
  - On peut retrouver d très facilement :
    - 143 ne peut pas être autre chose que 11*13
    - On peut donc le retrouver en 11 essais
- En pratique :
  - prendre de très grandes valeurs de p et q
  - Pour trouver ces grandes valeurs :
    - Jacobi + test de Solovay-Strassen
    - Rabin-Miller
    - …

RSA

Sécurité

Attaques

- 3 approches à attaquer RSA :
  - recherche par force brute de la clé (impossible étant donné la taille des données)
  - attaques mathématiques (basées sur la difficulté de calculer $\phi(n)$, la factorisation du module N)
  - attaques de synchronisation (sur le fonctionnement du déchiffrage)
- L'approche mathématique prend 3 formes :
  - factoriser n=p*q et par conséquent trouver $\phi(n)$ et puis d
  - déterminer $\phi(n)$ directement et trouver d
  - trouver d directement

Problème de la factorisation

- Connaissances actuelles sur la factorisation
  - lentes améliorations au cours des années
    - Au mieux 130 chiffres décimaux (512) en août 99
  - meilleure amélioration : optimisation des algorithmes
  - excepté un changement dramatique : RSA 1024 est sûr
    - S'assurer p, q ont une telle taille et respecter les autres contraintes
  - Remarque :
    - trouver les 2 facteurs premiers p et q d'un nombre n=p*q où p et q sont de l'ordre 10^{100} $\Rightarrow$ n a une longueur égale à 200 chiffres décimaux $\simeq$ 30 millions d'années pour le casser
    - déterminer si un nombre de 200 chiffres est premier ou non : moins de 5 minutes (en 2002)

Attaque de synchronisation (timing attack)

- Développé dans le milieu des années 90's
- Exploitation des variations de synchronisation entre les opérations opérations
  - par exemple la multiplication par petit contre grand nombre
  - ou instruction IF changeant les instructions a exécuter
- Impliquer la taille d'opérande basée sur le temps pris
- RSA exploite le temps pris dans l'élévation à une puissance
- contre mesures
  - employer des temps constant d'élévation à une puissance
  - ajouter des délais aléatoires
  - rendre non visible les valeurs utilisées dans les calculs

Conseils d'utilisation du RSA

- Ne jamais utiliser de valeur n trop petite
- Ne pas utiliser de clé secrète trop courte ($<\sqrt{(n)}$)
- N'utiliser que des clés fortes (p-1 et q-1 ont un grand facteur premier)
- Ne pas chiffrer de blocs trop courts
- Ne pas utiliser de n communs à plusieurs clés
- Si (d,n) est compromise ne plus utiliser n
- Ne jamais chiffrer ou authentifier un message provenant d'un tiers sans le modifier

Exemples – Rivest : 1978

- Le meilleur algorithme connu pour factoriser un nombre de 129 digits requiert :
  - 40 000 trillions d'années soit $40*10^{15}$ années
  - En supposant l'utilisation d'un superordinateur capable d'effectuer
    - 1 multiplication de nombres à 129 digits en 1 ns
  - Rappel : age de l'univers : 10^{10} années

Records de factorisation de grands nombres

Years	Number of decimal digits	Number of bits	Required computational power (in MIPS-years)
1974	45	149	0.001
1984	71	235	0.1
1991	100	332	7
1992	110	365	75
1993	120	398	830

Estimations

Year	Number of bits of N	Number of decimal digits of N	Method	Estimated amount of computations
1994	430	129	QS	5000 MIPS-years
1996	433	130	GNFS	750 MIPS-years
1998	467	140	GNFS	2000 MIPS-years
1999	**467**	**140**	**GNFS**	**8000 MIPS-years**

Recommended key sizes for RSA

Old standard:

Individual users — ~~512 bits (155 decimal digits)~~

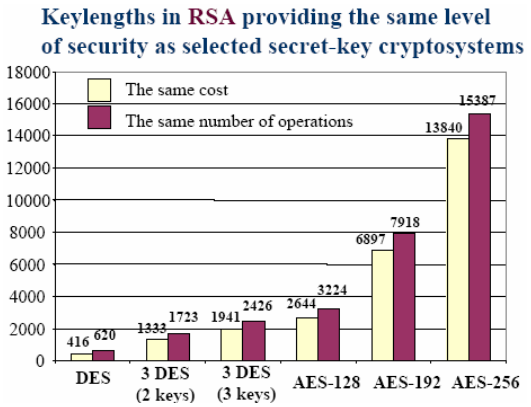
New standard:

Individual users — **768 bits** (231 decimal digits)

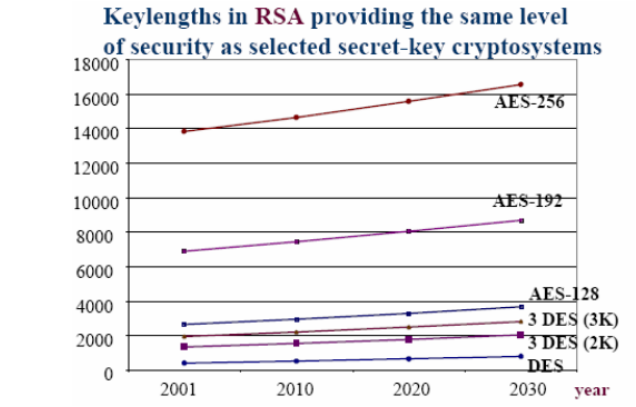
Organizations (short term) — **1024 bits** (308 decimal digits)

Organizations (long term) — **2048 bits** (616 decimal digits)

Comparaisons RSA - DES



Comparaisons RSA - DES



Challenge RSA

Lentgh of N in bits	Length of N in decimal digits	Award for factorization
576	174	$10,000
640	193	$20,000
704	212	$30,000
768	232	$50,000
896	270	$75,000
1024	309	$100,000
1536	463	$150,000
2048	617	$200,000

El Gamal

Présentation

- Algorithme à clef publique, datant de 1984, utilisé surtout comme base de la norme U.S. de signature électronique
- Basé sur la difficulté de calculer des logarithmes discontinus sur des champs finis.

Méthode pour générer les clés

- Choisir
  - un nombre premier p très grand et tel que p − 1 a un grand facteur premier
  - 2 nombres aléatoires g et x, tels que
    - $g \in (0 \ldots p-1)$ et $\forall z \in (1 \ldots p-2)\ g^z \neq 1 \bmod p$, $x \in (1 \ldots p-2)$
- Ensuite calculer
  - $y = g^x \bmod p$
- Clé publique : (y,g,p)
  - g et p peuvent être partagée par un groupe d'utilisateurs
- Clé privée : (x)

Chiffrement El Gamal

- Chiffrement
  - Soit M, k un nombre aléatoire, (k,p) = 1
    - K n'est connu que de celui qui chiffre et différent à chaque msg
  - Calculer :
    - $a = g^k \bmod p$
    - $b = y^k M \bmod p$
  - La paire (a, b) forme le texte chiffré → C = 2M
- Déchiffrement :
  - $M = b/a^x \bmod p$

Remarque

- y^k est un masque qui est appliqué sous forme multiplicative à M.
- Pour déchiffrer le message, il faut
  - soit trouver directement un masque jetable (en l'absence de la connaissance de a, y^k est une suite parfaitement imprévisible)
  - Soit trouver la clé privée x qui est la solution de $y = g^x \bmod p$ (donc trouver le logarithme discret)

Inversibilité de la méthode

- Puisque :
  - $a^x \equiv g^{kx} \pmod p$
  - On a :
    - $b/a^x \equiv y^k M / a^x \pmod p \equiv g^{xk} M / g^{xk} \equiv M \pmod p$
- Ou encore $(D_k(E_k(M))=M)$
  - $D_k(E_k(M)) = ((M*y^k \bmod p)/(g^k \bmod p)^x) \bmod p$
    $= ((M*y^k \bmod p)/(g^{xk} \bmod p)) \bmod p$
    $= ((M*y^k \bmod p)/(y^k \bmod p)) \bmod p = M$

Exemple

- Soient p=2579, g=2, x=765
- S_k = (765)
- P_k = (2579, 2 , 949) car $2^{765} \bmod 2579 = 949$
- Pour chiffrer M = 1299, on choisi k = 853
  - $a = 2^{853} \bmod 2579 = 435$
  - $b = 1299*949^{853} \bmod 2579 = 2396$
- On peut vérifier que effectivement :
  - $2396/(435^{765}) \bmod 2579 = 1299$

Efficacité et sécurité

- Calculs comparables à ceux du RSA $\rightarrow$ ordre de performance identique (2 fois plus lent)
- Inconvénient : la taille des données chiffrée est 2* celle des données en clair
- La recherche de la clé privée (x) à partir de la clé publique est équivalente au problème du logarithme discret (NP). MAIS il n'est pas prouvé que la cryptanalyse d'un message chiffré avec ElGamal soit équivalente au logarithme discret.

Efficacité et sécurité

- Les techniques de cryptanalyse de ElGamal sont directement liée à celles de Diffie-Hellman ([Stinson]-ch5)

Comparaisons

Asymétrique vs symétrique

- Symétrique :
  - Avantages
    - Rapidité (jusqu'à 1000 fois)
    - Facilité d'implantation sur hardware
    - Taille de clé : 128 bits (⇒ 16 caractères : mémorisable)
  - Inconvénients
    - Nombre de clés à gérer
    - Distribution des clés (authentification, confidentialité)
    - Certaines propriétés (p.ex. signatures) sont difficiles à réaliser

Asymétrique vs symétrique

- Asymétriques :
  - Avantages :
    - Nombre de clé à distribuer est réduit par rapport aux clés symétriques
    - Distributions des clés facilités : pas besoin de l'authentification
    - Permet de signer des messages facilement
  - Inconvénients :
    - Taille des clés
    - Vitesse de chiffrement
- Problèmes propres aux 2 systèmes
  - La gestion des clés 'secrètes' reste le maillon faible
  - Sécurité basée sur des arguments empiriques plutôt que théorique

Références internet

- http://www.bibmath.net/crypto/complements
- http://www.bibmath.net/crypto/moderne
- http://www.ulb.ac.be/di/scsi/markowitch/crypto/
- http://www.ift.ulaval.ca/~Mejri/cryptography/Notes/RSA.pdf
- http://www.cryptologie.com/
- http://abcdrfc.free.fr/rfc-vf/rfc2440/x1529.html
- www.hack.gr/users/dij/crypto/overview/rsa.html
- Home.ecn.ab.ca/~jsavard/crypto/publ05.html
- http://193.48.37.48/~douillet/publications/crypto/node17.html
- http://www.cryptosec.org/article.php3?id_article=10
- http://cui.unige.ch/tcs/cours/crypto/crypto7/crypto7.html
- http://members.tripod.com/irish_ronan/rsa/attacks.html

Références bibliographiques

- [Stinson] - ch5, ch6
- [Stallings] – ch 9
- [Schneier] – ch 19
- [Natkin] - 2.3

Questions

- Vous devez pouvoir expliquer
  - Le concept des systèmes à clés publique
  - L'algorithme du sac à dos (Merkle Hellman)
    - (principe, algorithme, justification, sécurité, …)
  - L'algorithme du RSA
    - (principe, algorithme, justification, sécurité, …)
  - L'algorithme de ElGamal
    - (principe, algorithme, justification, sécurité, …)
- Vous devez pouvoir comparer et illustrer les différences entre les systèmes symétriques et asymétriques