

Public Key Cryptography (Asymmetric Cryptography)

Outlook

- Mathematical Background
- Reference Problems
- Factoring
- RSA Encryption Algorithm
- ElGamal Encryption Algorithm
- Rabin Encryption Algorithm
- Elliptic Curves

Fondements Mathématiques

- **Théorème Fondamental de l'Arithmétique:**

Tout nombre entier strictement positif n s'écrit de façon unique (à l'ordre près) comme un produit de puissances de nombres premiers p_i distincts, à savoir:

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot p_3^{e_3} \cdot \dots \cdot p_m^{e_m}$$

- **Fonction Phi d'Euler:**

Soit $n \in \mathbb{Z}^+$, la **fonction phi d'Euler** $\Phi(n)$ est égale au nombre de d'entiers positifs plus petits que n qui sont relativement premiers à n .

*→ Si n est premier
 $\Phi(n) = n-1$*

- Calcul de la **valeur** de la fonction **phi d'Euler** $\Phi(n)$:

D'après le théorème fondamental de l'arithmétique, tout nombre entier $n > 1$ s'écrit:

$$n = \prod_{i=1}^m p_i^{e_i}$$

alors $\Phi(n)$ se calcule:

$$\Phi(n) = \prod_{i=1}^m (p_i^{e_i} - p_i^{(e_i-1)})$$

Fondements Mathématiques (II)

- En particulier, si n s'écrit comme $n = p \cdot q$ avec p et q premiers, alors:

$$\Phi(n) = (p-1) \cdot (q-1)$$

$\underbrace{\hspace{1cm}}_{\Phi(p)} \quad \underbrace{\hspace{1cm}}_{\Phi(q)}$

car p et q premiers

- Théorème d'Euler:**

Soient $n \in \mathbb{Z}^+$ et $a \in \mathbb{Z}$ avec $\text{pgcd}(a, n) = 1$, alors on a:

$$a^{\Phi(n)} \equiv 1 \pmod{n}$$

- Petit Théorème de Fermat** (cas particulier du théorème d'Euler si n est premier):

Soient $a \in \mathbb{Z}$ et p un nombre premier tel que p ne divise pas a , alors on a:

$$a^{p-1} \equiv 1 \pmod{p}$$

A noter que puisque p est premier, on a $\Phi(p) = p-1$.

- Réduction des exposants mod $\Phi(n)$:**

Si n est le produit de premiers distincts et $r, s \in \mathbb{Z}$ t.q. $r \equiv s \pmod{\Phi(n)}$ alors $\forall a \in \mathbb{Z}$:

$$a^r \equiv a^s \pmod{n}$$

- Application du Théorème d'Euler au calcul des inverses:**

Suite au théorème d'Euler, on a que:

$$a \cdot a^{\Phi(n)-1} \equiv 1 \pmod{n}$$

ce qui signifie que $a^{\Phi(n)-1}$ est l'inverse de $a \pmod{n}$. En particulier, a^{p-2} est l'inverse de $a \pmod{p}$ si p est premier.

Fondements Mathématiques (III)

- **Définition:** Le groupe multiplicatif de \mathbb{Z}_n , noté \mathbb{Z}_n^* est $= \{ a \in \mathbb{Z}_n \text{ t.q. } \text{pgcd}(a,n) = 1 \}$.
En particulier, si n est premier: $\mathbb{Z}_n^* = \{ a \text{ t.q. } 1 \leq a \leq n-1 \}$
- **Le nombre d'éléments ou ordre du groupe multiplicatif \mathbb{Z}_n^*** est $\Phi(n)$ (par déf. de Φ).
- **Définition:** Soit $a \in \mathbb{Z}_n$, l'ordre de a est le plus petit entier positif t pour lequel:
 $a^t \equiv 1 \pmod n$
- **Définition:** Soit $\alpha \in \mathbb{Z}_n^*$, si l'ordre de α est $\Phi(n)$, alors α est un **générateur** de \mathbb{Z}_n^* .
Lorsqu'un groupe \mathbb{Z}_n^* a un générateur, on dit qu'il est **cyclique**.
- **Propriétés des générateurs:**
 - \mathbb{Z}_n^* a un générateur ssi. $n = 2, 4, p^k$ ou $2p^k$, avec p premier, $p \neq 2$ et $k \geq 1$.
En particulier, si p est premier, \mathbb{Z}_p^* a un générateur.
 - Si α est un générateur de \mathbb{Z}_n^* , alors tous les éléments de \mathbb{Z}_n^* s'écrivent:
 $\mathbb{Z}_n^* = \{ \alpha^i \pmod n \text{ t.q. } 0 \leq i \leq \Phi(n) - 1 \}$
 - Le nombre de générateurs de \mathbb{Z}_n^* est $\Phi(\Phi(n))$.
 - α est un générateur de \mathbb{Z}_n^* ssi. \forall premier p t.q. p divise $\Phi(n)$, on a:
 $\alpha^{\Phi(n)/p} \not\equiv 1 \pmod n$.
En particulier si n est un **premier** de la forme $2p + 1$ avec p premier (un tel n est appelé un **safe prime**), α est générateur de \mathbb{Z}_n^* ssi. $\alpha^2 \not\equiv 1 \pmod n$ et $\alpha^p \not\equiv 1 \pmod n$.

$n=6$
 $\rightarrow \text{pgcd}(1,6) = 1 \checkmark$
 $\text{pgcd}(2,6) = 2$
 $\text{pgcd}(3,6) = 3$
 $\text{pgcd}(4,6) = 2$
 $\text{pgcd}(5,6) = 1$
 $\mathbb{Z}_6^* = \{1, 5\}$
 $\Phi(6) = 2$
 $a=1, t=1$
 $1^1 \pmod 6 = 1$
 $a=4, t=?$
 $4^1 \pmod 6 = 4$
 $4^2 \pmod 6 = 4$
 $4^3 \pmod 6 = 4$
 \vdots
 $a=5, t=?$
 $5^1 \pmod 6 = 5$
 \vdots

Fast Exponentiation

- **Fast exponentiation:** En utilisant la représentation binaire d'un nombre, on peut calculer des puissances très efficacement, p.ex calcul de $2^{644} \bmod 645$:

$$(644)_{10} = (1010000100)_2$$

Maintenant, on calcule les exposants correspondants aux puissances de 2, soient:

$$2 \equiv 2 \bmod 645$$

$$2^2 \equiv 4 \bmod 645$$

$$2^4 \equiv 16 \bmod 645$$

$$2^8 \equiv 256 \bmod 645$$

$$2^{16} \equiv 391 \bmod 645$$

...

$$2^{512} \equiv 256 \bmod 645$$

D'après la représentation binaire, on calcule:

$$2^{644} = 2^{512+128+4} = 2^{512} 2^{128} 2^4 = 1601536 \equiv 1 \bmod 645$$

La complexité de cet algorithme *fast exponentiation* est $O(\log^3 n)$.

En s'appuyant sur le théorème d'Euler, le calcul de l'inverse d'un nombre dans un tel groupe est donc effectué en temps polynomial.

- **L'algorithme d'Euclide étendu** peut être également utilisé pour trouver un x t.q:
 $ax \equiv 1 \bmod n$ puisque cette congruence s'écrit: $ax - 1 = kn$ et donc:
 $ax - kn = 1 = \text{pgcd}(a, n)$. La complexité de cet algorithme est également $O(\log^3 n)$.

Théorème des Restes Chinois

- Le **Théorème des Restes Chinois** (III^e siècle!) permet de résoudre des systèmes linéaires de congruences simultanées. Il résout des problèmes soulevés dans des anciens puzzles chinois. Il s'agissait, par exemple, de trouver un nombre qui produit un reste de 1 lorsqu'il est divisé par 3, de 2 lorsqu'il est divisé par 5 et de 3 lorsqu'il est divisé par 7... Il fut également utilisé pour calculer le moment exact d'alignement de plusieurs astres ayant des orbites (et donc des périodes) différentes.

- **Théorème des Restes Chinois:** Soient $n_1, n_2, \dots, n_t \in \mathbb{Z}^+$ premiers deux à deux (c.à.d. $\text{pgcd}(n_i, n_j) = 1, i \neq j$) et $a_1, a_2, \dots, a_t \in \mathbb{Z}$. Alors, le système de congruences:

$$x \equiv a_1 \pmod{n_1}$$

$$x \equiv a_2 \pmod{n_2}$$

...

$$x \equiv a_t \pmod{n_t}$$

a une solution unique $x \pmod{N := n_1 n_2 \dots n_t}$



- **Algorithme de Gauss** (1801) pour le calcul de x : $x = \sum_{i=1}^t a_i N_i M_i \pmod{N}$ avec $N_i = N/n_i$ et $M_i = N_i^{-1} \pmod{n_i}$. La complexité de cet algorithme est $O(\log^3 n)$.
- Il est donc possible en temps polynomial de passer des congruences **mod n_i** et aux congruences **mod N** !

Problèmes de Base

- **Problèmes génériques principaux**

- **Factorisation (FACTP):** Etant donné un entier positif n , trouver sa factorisation en nombre premiers. *Si n grand \rightarrow trop compliqué*
- **Logarithmes discrets (DLP):** Etant donné un nombre premier p , un générateur $\alpha \in \mathbb{Z}_p^*$ et un élément $\beta \in \mathbb{Z}_p^*$, trouver l'entier x , $0 \leq x \leq p-2$, tel que: $\alpha^x \equiv \beta \pmod{p}$.
- **Racine carrée dans \mathbb{Z}_n si n est composite (SQROOTP):** Etant donné un entier composite n et un résidu quadratique a , trouver la racine carrée de $a \pmod{n}$.

- **Problèmes spécifiques (propres à un système de cryptage):**

- **RSA (RSAP):** Etant donné un entier positif $n = pq$, un entier positif e avec $\gcd(e, (p-1)(q-1)) = 1$ et un entier c , trouver un entier m avec $m^e \equiv c \pmod{n}$. 
- **Diffie-Hellman (DHP):** Etant donnée un nombre premier p , un générateur $\alpha \in \mathbb{Z}_p^*$ et les éléments $\alpha^a \pmod{p}$ et $\alpha^b \pmod{p}$, trouver $\alpha^{ab} \pmod{p}$. 

- **Résultats prouvés:**

- $\text{DHP} \leq_p \text{DLP}$ (Equivalent sous certaines conditions¹)
- $\text{RSAP} \leq_p \text{FACTP}$ (Prouvé équivalent pour le problème générique, voir page 101)
- $\text{SQROOTP} \cong_p \text{FACTP}$

1. *The Relationship Between Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms*. U. Maurer, S. Wolf. SIAM Journal of Computing, volume 28, no. 5, pages 1689-1721, 1999.

Classical Factoring Techniques

Exponential Time

They run at best in $O(\exp(c * \ln(n)))$

- Trial Division
- Eratosthenes' Sieve (II B.C.)
- Fermat's Difference of Squares Method (~1650)
- Square Form Factorization (1971)
- Pollard's **p**-1 method (1974)
- Pollard's **Rho** Method (1975)

Subexponential Time

They run at best in $O(\exp(c * (\ln(n))^{1/3}))$

- Continued Fractions (1975)
- Quadratic Sieve (1981)
- Number Field Sieve - NFS (1990)
- General Number Field Sieve - GNFS (2006)

Polynomial Time Methods

- Shor's Algorithm in a Quantum Computer (1994) runs in $O(\log^c n)$

Factoring: New Developments

- Bernstein's specific NFS purpose computer¹ that (on his assumptions) factors a 1536 bit number would take the same time than a 512-bit computation in a conventional machine running NFS
- Lenstra, & al.² claim that the "Bernstein factor should be brought down to 1.17 but defend that a 1024-bit number one-day factoring machine can be built for a few thousand dollars under some optimistic assumptions.
- More on the Bernstein - Lenstra, Shamir controversy on <http://cr.yp.to/nfscircuit.html>
- Largest number factorization to date (2011) using NFS is RSA-768 (a 232-digit number)³. Using hundreds of general purpose processors in a 3 year effort.
- Factoring in a Quantum Computer
 - Significant problems to build the machine (errors, dispersion, etc.)
 - I. Chuang (IBM,Almaden) builds a 7 *qu-bit* computer (December 2001)
 - Works in progress: feasibility of a computer featuring millions of *qu-bits*...?

1.Daniel J. Bernstein. *Circuits for Integer Factorization: a Proposal* (October 2001)

2.Lenstra, Shamir et al. *Analysis of Bernstein's Factorization Circuit*. June 2002

3.Kleinjung, et al (2010-02-18). *Factorization of a 768-bit RSA modulus*. International Association for Cryptologic Research. Retrieved 2010-08-09

Procédé d'Encryption/Decryption de RSA¹

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
 - **A** choisit la taille du **modulus n** (p.ex. taille (n) = 1024 ou taille (n) = 2048).
 - **A** génère deux nombres premiers **p** et **q** de grande taille ($\sim n/2$). $p=13, q=19$
 - **A** calcule $n := pq$ et $\Phi(n) = (p-1)(q-1)$. $n=247, \Phi(n)=216$
 - **A** génère l'exposant d'encryption **e**, avec $1 < e < \Phi(n)$ t.q. $\text{pgcd}(e, \Phi(n)) = 1$. $\text{pgcd}(5, 216) = 1$
 $e=5$
 - **A** calcule l'exposant de decryption **d**, t.q.: $ed \equiv 1 \pmod{\Phi(n)}$ avec l'algorithme d'Euclide étendu ou avec l'algorithme *fast exponentiation* (page 94). $d=2$
 - Le couple **(n,e)** est la **clé publique de A**; **d** est la **clé privée de A**.

Encryption

- L'entité **B** obtient **(n,e)**, la clé publique authentique de **A**.
- **B** transforme son plaintext en une série d'entiers m_i , t.q. $m_i \in [0, n-1] \forall i$.
- **B** calcule le ciphertext $c_i := m_i^e \pmod{n}$, $\forall i$ avec l'algorithme *fast exponentiation*.
- **B** envoie à **A** tous les ciphertext c_i .

Decryption

- **A** utilise sa clé privée pour calculer les plaintexts $m_i = c_i^d \pmod{n}$.

1. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. R.Rivest, A.Shamir and L.M. Adleman. Communications of the ACM 21 (1978), 120-126

Preuve de RSA

- Soit **m** le plaintext et **c** le ciphertext avec **c** := **m^e mod n**, il s'agit de prouver:
m != **c^d mod n**:

En substituant **c** par sa valeur on obtient:

$$\mathbf{c^d \bmod n = m^{ed} \bmod n (*)}$$

mais, on sait que:

$$\mathbf{ed \equiv 1 \bmod \Phi(n)}$$

et donc par définition des congruences, il existe un entier **k** avec:

$$\mathbf{ed - 1 = k\Phi(n)}$$

en substituant dans (*):

$$\mathbf{(c^d \equiv m^{k\Phi(n)+1} \equiv m^{k\Phi(n)} m) \bmod n}$$

Si **pgcd (m,n) = 1**, on a par le théorème d'Euler:

$$\mathbf{1 \equiv m^{\Phi(n)} \bmod n}$$

donc:

$$\mathbf{(c^d \equiv m^{k\Phi(n)} m \equiv m) \bmod n} \qquad \mathbf{c.q.f.d. !}$$

Si **pgcd (m,n) ≠ 1**, **m** est nécessairement multiple de **p** ou de **q** (cas très peu probable...), on peut montrer en faisant les calculs **mod p** et **mod q** que la congruence reste vraie.

RSA: Sécurité

- Le problème **RSAP** (page 96) consistant à trouver **m** à partir de **c** n'est pas prouvé comme étant aussi difficile que la factorisation mais...:
- ... on peut prouver que si on trouve **d** on peut facilement calculer **p** et **q**. Ceci équivaut à dire que factoriser **n** et trouver **d** nécessitent un effort de calcul équivalent.
- On sait que les méthodes les plus rapides pour factoriser (page 97) ont une complexité *sub-exponentielle* ($\sim O(\exp(c * (\ln(n))^{1/3}))$). Le problème reste donc calculatoirement impossible pour des modulus ≥ 1048 bits (2048 bits est un choix fréquent pour une sécurité durable...).
- Afin d'améliorer la vitesse d'encryption, on a tendance à choisir des exposants **e** assez petits (typiquement: **e := 3**, **e:=17** et **e:=19**). On a cependant prouvé que le calcul d'une *i*-ème racine (avec *i* petit) modulo un composite **n** peut être nettement plus facile que la factorisation de **n**¹. Par contre, en 2008 on a prouvé que **la résolution générique du problème RSA est équivalent à la factorisation**.²
- L'exposant de decryption **d** doit impérativement être de grande taille³ (au moins la moitié de la taille de **n**) pour garantir la sécurité du système.
- Par conséquent, **l'encryption est normalement nettement plus rapide que la decryption** puisque les exposants utilisés sont beaucoup plus petits!

1. *Breaking RSA May Not Be Equivalent to Factoring*. D. Boneh et R. Venkatesan. Advances in Cryptology - EUROCRYPT '98.

2. *Breaking RSA Generically is Equivalent to Factoring*. D. Aggarwal and U. Maurer. CRYPTO 2008 Rump Session.

3. *Cryptanalysis of short RSA secret exponents*. M.J Wiener. IEEE transactions on Information Theory (1990), 553-558.

RSA: Attaques

- Lors qu'on souhaite encrypter le même message pour un groupe de correspondants, il convient d'introduire des variations (*randomization*) avant l'encryption pour éviter l'attaque suivant:

Admettons qu'on calcule des ciphertexts c_1, c_2, c_3 à partir du même plaintext m et du même exposant $e:=3$ adressés à trois entités avec des modulus: n_1, n_2, n_3 .

Le *Théorème des Restes Chinois* nous dit qu'il existe une solution $x \bmod n_1 n_2 n_3$, t.q.:

$$x \equiv c_1 \bmod n_1, \quad x \equiv c_2 \bmod n_2, \quad x \equiv c_3 \bmod n_3$$

Mais si m ne change pas pour les trois encryptions, on a que $x = m^3 \bmod n_1 n_2 n_3$ et, de plus: $m^3 < n_1 n_2 n_3$. On peut, donc, trouver m en calculant la **racine cubique entière** de m^3 , en sachant que pour ce calcul il existe des algorithmes efficaces!

- Plus généralement, si $m < n^{1/e}$, on peut appliquer des algorithmes rapides (dans \mathbb{Z}) pour calculer les **racines e-ièmes** de m^e . Il convient donc d'effectuer des opérations de "*randomization*" de m avant d'encrypter!
- La propriété multiplicative de RSA: $(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \bmod n$ donne lieu à des failles dangereuses (voir signatures aveugles).
- En admettant que les paramètres sont correctement choisis et que l'implantation n'a pas de failles, la méthode la plus efficace pour "casser" l'algorithme générique RSA reste la factorisation de n .



Procédé d'Encryption/Decryption d'ElGamal¹

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
 - **A** génère un nombre premier **p** de grande taille (**len(p) ≥ 1024 bits**) et un générateur α du groupe multiplicatif \mathbb{Z}_p^* .
 - **A** génère un nombre aléatoire **a**, t.q. $1 \leq a \leq p-2$ et calcule $\alpha^a \bmod p$.
 - La clé publique de **A** est **(p, α , $\alpha^a \bmod p$)**, la clé privée de **A** est **a**.

Encryption

- L'entité **B** obtient **(p, α , $\alpha^a \bmod p$)**, la clé publique authentique de **A**.
- **B** transforme son plaintext en une série d'entiers **m_i**, t.q. **m_i ∈ [0, p-1] ∀i**.
- Pour chaque message **m_i**:
 - **B** génère un nombre aléatoire unique **k**, t.q. $1 \leq k \leq p-2$.
 - **B** calcule $\lambda := \alpha^k \bmod p$ et $\delta := m_i (\alpha^a)^k \bmod p$ et envoie le ciphertext **c := (λ, δ)**.

Decryption

- **A** utilise sa clé privée **a** pour calculer $\lambda^{p-1-a} \bmod p$ (à noter que:
 $\lambda^{p-1-a} \equiv \lambda^{-a} \equiv \alpha^{-ak} \bmod p$).
- **A** retrouve le plaintext en calculant: $(\alpha^{-ak}) \delta \bmod p$.

1. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. T. ElGamal. Advances in Cryptology- Proceedings of CRYPTO 84 (LNCS 196), 10-18, 1985.

Procédé d'ElGamal: Remarques

- La preuve que la decryption fonctionne est évidente en substituant δ par sa valeur:
$$(\alpha^{-ak}) \delta \bmod p \equiv (\alpha^{-ak}) \mathbf{m} (\alpha^a)^k \bmod p \equiv \mathbf{m} \bmod p.$$
- Le procédé d'ElGamal se base sur la difficulté de calculer des logarithmes discrets modulo un nombre premier (voir problème **DLP** en page 96) même s'il n'a pas été prouvé qu'il soit strictement équivalent à ce problème.
- Les algorithmes les plus efficaces connus à cette date ont une complexité *sub-exponentielle* très proche de celle de la factorisation (on utilise souvent les mêmes algorithmes...)
- Les exposants choisis **(k,a)** doivent être de grande taille car il existe des algorithmes efficaces pour calculer des logarithmes discrets modulo un nombre premier lorsque l'exposant est petit (*baby-step giant-step algorithm*¹)
- Un inconvénient d'ElGamal est qu'il multiplie par 2 la longueur du ciphertext.
- Il est essentiel pour la sécurité du procédé que le nombre aléatoire **k ne soit pas répété**, autrement: soient (λ_1, δ_1) et (λ_2, δ_2) les deux ciphertexts générés, on a que $\delta_1/\delta_2 = \mathbf{m}_1/\mathbf{m}_2$ et par conséquent, il est trivial de retrouver un plaintext à partir de l'autre...
- Le procédé d'ElGamal peut se généraliser à d'autres groupes comme **GF(2ⁿ)** ou les courbes elliptiques (voir page 111).

1. Algorithm by D.Shanks appearing in: *The Art of Computer Programming-Fundamental Algorithms*. D.E Knuth. Volume 1. Addison Wesley 1973.

Procédé d'Encryption/Decryption de Rabin¹

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
 - **A** génère deux nombres premiers aléatoires **p** et **q** de grande taille ($\text{len}(pq) \geq 1024$).
 - **A** calcule **n** := **pq**.
 - La clé publique de **A** est **n** la clé privée de **A** est (**p,q**).

Encryption

- L'entité **B** obtient **n**, la clé publique authentique de **A**.
- **B** transforme son plaintext en une série d'entiers **m_i**, t.q. **m_i** $\in [0, n-1] \forall i$.
- **B** calcule **c_i** = **m_i**² mod **n** pour chaque message **m_i**.
- **B** envoie tous les ciphertext **c_i** à **A**.

Decryption

- **A** utilise sa clé privée (**p,q**) pour retrouver les **4 solutions** de l'équation: **c_i** = **x**² mod **n** en utilisant des algorithmes efficaces pour calculer des racines carrées mod **p** et mod **q**.
- **A** détermine soit par une indication supplémentaire de **B**, soit par une analyse de redondance lequel des 4 messages **m₁**, **m₂**, **m₃**, **m₄** est le plaintext original.

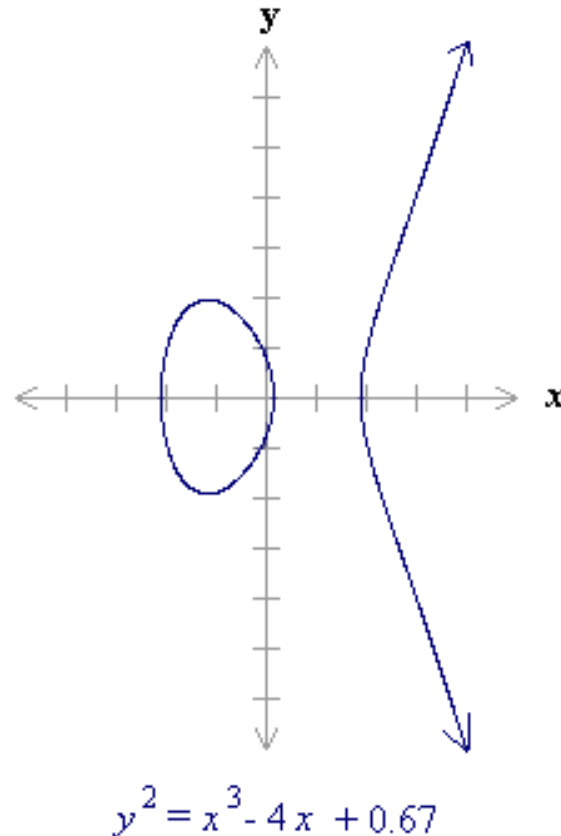
1. *Digitalized Signatures and Public Key Functions as Intractable as Factorization*. M.O.Rabin. MIT/LCS/TR 212. MIT Laboratory for Computer Science 1979.

Procédé de Rabin: Remarques

- Le procédé de Rabin est basé sur l'impossibilité de trouver des racines carrées modulo un composite de factorisation inconnue (problème SQROOTP, voir page 96).
- L'intérêt principal de cet algorithme réside dans le fait qu'il a été prouvé comme étant équivalent à la factorisation ($\text{SQROOTP} \cong \text{FACTP}$). Cet algorithme appartient donc à la catégorie *provably secure* pour toute attaque passive.
- Les **attaques actives** peuvent, dans certains cas, compromettre la sécurité de l'algorithme. Plus précisément, si on monte l'attaque *chosen ciphertext* (on demande à A de decrypter un ciphertext choisi) suivant:
 - L'attaquant **M** génère un **m** et envoie à **A** le ciphertext $\mathbf{c} = \mathbf{m}^2 \bmod \mathbf{n}$.
 - **A** répond avec une racine \mathbf{m}_x parmi les 4 possibles $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{m}_4$.
 - Si $\mathbf{m} \equiv \pm \mathbf{m}_x \bmod \mathbf{n}$ (probabilité 0.5), **M** recommence avec un nouveau **m**.
 - Sinon, **A** calcule $\text{pgcd}(\mathbf{m} - \mathbf{m}_x, \mathbf{n})$ et obtient ainsi un des deux facteurs de **n**...
- Cette attaque pourrait être évitée si le procédé exigeait une redondance suffisante dans les plaintexts permettant à A d'identifier sans ambiguïté laquelle des solutions possibles est le plaintext original. Dans ce cas, A répondrait toujours **m** et jetterait les autres solutions n'ayant pas le niveau de redondance préétabli.

Courbes Elliptiques¹

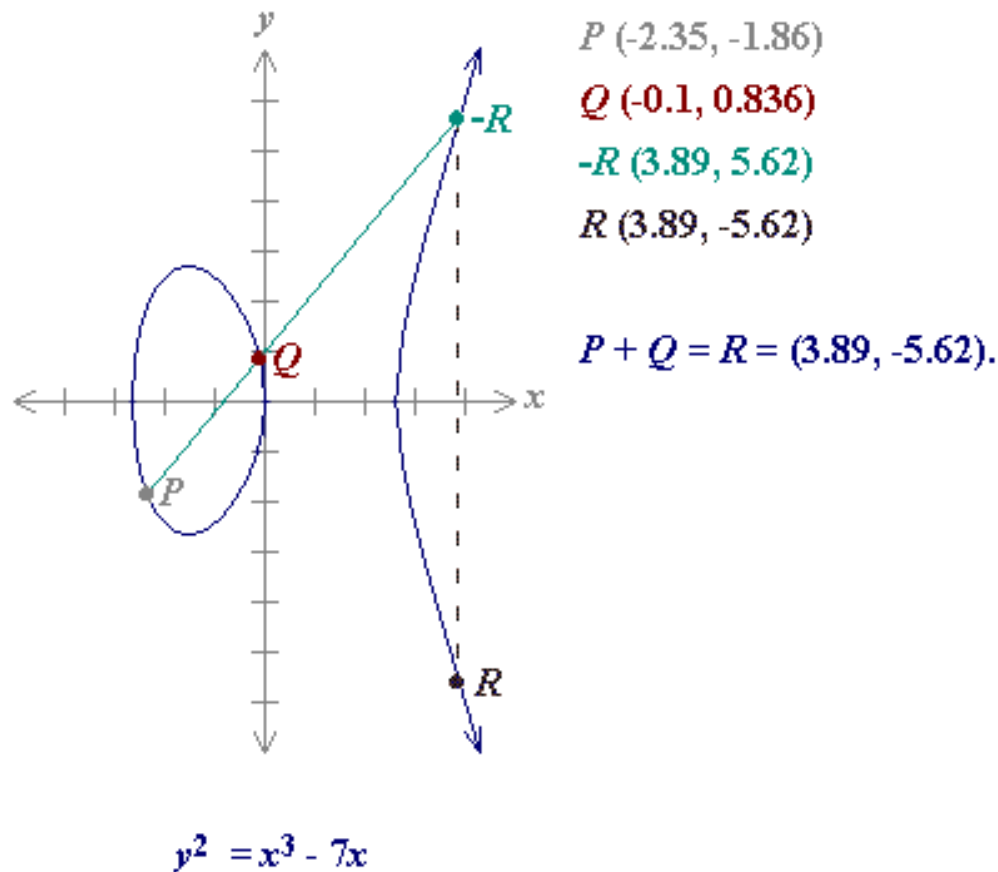
- Une **courbe elliptique** est un ensemble de points **E** défini par l'équation:
 $y^2 = x^3 + ax + b$, avec x, y, a et b des nombres **rationnels, entiers** ou **entiers modulo m** ($m > 1$). L'ensemble E contient également un "**point à l'infini**" noté ∞ . Le point ∞ n'est pas dans la courbe mais il est l'élément identité de E .
- On choisira pour nos calculs les courbes elliptiques n'ayant pas de racines multiples ou, en d'autres termes, des courbes où le discriminant $4a^3 + 27b^2 \neq 0$.



1. Source de toutes les images sur les courbes elliptiques: <http://www.certicom.com>

Courbes Elliptiques: Addition

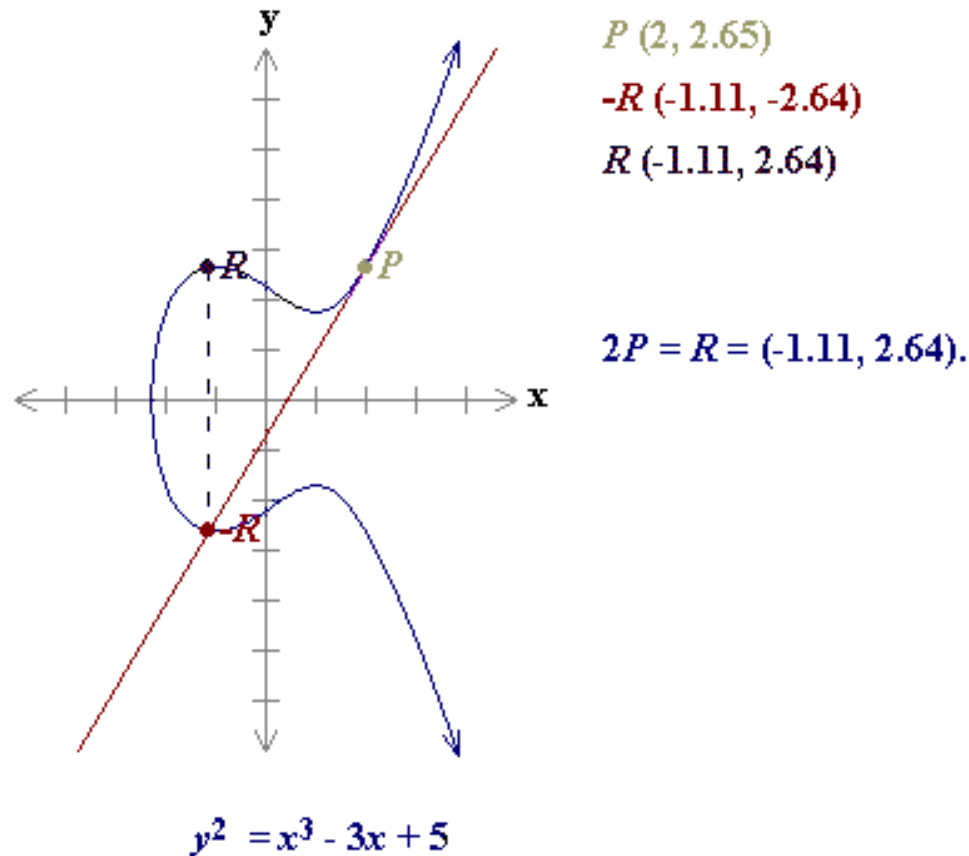
- Soit $P := (x, y) \in E$, on définit $-P$ comme $-P := (x, -y)$. Graphiquement, $-P$ est le point symétrique de P par rapport à l'axe des x . A noter que $P + (-P) = \infty$.
- Soient deux points $P, Q \in E$, tels que $Q \neq -P$, on définit l'addition $P + Q := R$ où $R \in E$ t.q. $-R$ est le 3^e point d'intersection entre la courbe et la droite qui passe par P et Q :



- L'ensemble E avec ∞ définit un **groupe commutatif** pour l'addition.

Courbes Elliptiques: Multiplication par un scalaire

- Soit $P \in E$, le point $2P = R$, t.q. $-R$ est le point d'intersection de la courbe avec la droite tangente à la courbe au point P .



- Lorsque la courbe elliptique est définie sur le corps \mathbb{Z}_p avec p un nombre premier de grande taille ($y^2 \equiv x^3 + ax + b \pmod{p}$), le calcul de $k \in \mathbb{Z}_p$ t.q. $Q = kP$ avec (P, Q) connus, est très difficile (nécessite un effort exponentiel...). Ce problème est connu comme: **Elliptic Curve Discrete Logarithm Problem (ECDLP)**

Courbes Elliptiques: Remarques

- L'avantage principal de la cryptographie publique basée sur des courbes elliptiques est que la taille des nombres utilisés (et donc, des clés) est plus petite.
- Ceci est dû à la complexité accrue des calculs sur E_p (courbe elliptique définie sur le corps Z_p) par rapport aux corps habituels tels que Z_p ou $GF(2^m)$.
- Ce tableau montre les rapports des tailles des clés par rapport à celles de RSA:

NIST guidelines for public key sizes for AES			
ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

- La représentation d'un plaintext en points de la courbe reste une opération complexe.
- En Octobre 2003¹, la *US National Security Agency* (NSA) a acheté un brevet de *Certicom* pour l'utilisation de la cryptographie à courbes elliptiques.
- En Septembre 2013 Claus Diem montr² que sous certaines conditions le problème **ECDLP** pouvait être résolu en temps *sub-exponentiel*.

1. http://www.certicom.com/index.php?action=company,press_archive&view=175

2. <http://www.math.uni-leipzig.de/~diem/preprints/dlp-ell-curves-II.pdf>

Procédé d'ElGamal sur des Courbes Elliptiques

Génération des clés

- Chaque entité (**A**) crée une paire de clés (publique et privée) comme suit:
 - **A** choisit une courbe elliptique E_p avec p , un nombre premier de grande taille ($\text{len}(p) \sim 200$ bits) et un point $P_o \in E_p$.
 - **A** génère un nombre aléatoire a , t.q. $1 < a < p$ et calcule $P_a = aP_o$ (multiplication par un scalaire sur E_p , pour laquelle, il existe des algorithmes efficaces).
 - La clé publique de **A** est (E_p, P_o, P_a) , la clé privée de **A** est a .

Encryption

- L'entité **B** obtient (E_p, P_o, P_a) , la clé publique authentique de **A**.
- **B** transforme son plaintext en une série d'entiers m_i , t.q. $m_i \in E_p \forall i$.
- Pour chaque message m_i :
 - **B** génère un nombre aléatoire unique k , t.q. $1 < k < p$.
 - **B** calcule $\lambda := kP_o$ et $\delta := kP_a + m_i$ et envoie le ciphertext $c := (\lambda, \delta)$.

Decryption

- **A** utilise sa clé privée a pour calculer: $a\lambda = akP_o = kP_a$.
- **A** retrouve le plaintext en calculant: $\delta - kP_a = kP_a + m_i - kP_a = m_i$.
- La sécurité du schéma s'appuie sur **ECDLP**!

Tableau Récapitulatif¹

Table 1

Public-key system	Mathematical Problem	Examples
Integer factorization	Given a number n , find its prime factors	RSA, Rabin-Williams
Discrete logarithm	Given a prime n , and numbers g and h , find x such that $h = g^x \bmod n$	ElGamal, Diffie-Hellman, DSA
Elliptic curve discrete logarithm	Given an elliptic curve E and points P and Q on E , find x such that $Q = xP$	EC-Diffie-Hellman, ECDSA

Table 2

Public-key system	Best known methods for solving mathematical problem	Running times
Integer factorization	Number field sieve: $\exp(1.923 (\log n)^{1/3} (\log \log n)^{2/3})$	Sub-exponential
Discrete logarithm	Number field sieve: $\exp(1.923 (\log n)^{1/3} (\log \log n)^{2/3})$	Sub-exponential
Elliptic curve discrete logarithm	Pollard-rho algorithm: square root of n	Fully exponential

1. *Next Generation Security for Wireless*. S.A Vanstone, 2004. http://www.compseconline.com/hottopic/hottopic20_8/Next.pdf