Table of contents

Cryptographie Asymétrique (à Clés Publiques)

Fondements Mathématiques

Théorème Fondamental de l'Arithmétique et Fonction Phi d'Euler

La cryptographie asymétrique repose sur des fondements mathématiques solides issus de la théorie des nombres. Deux concepts sont essentiels :

Théorème Fondamental de l'Arithmétique : Tout nombre entier strictement positif n s'écrit de façon unique (à l'ordre près) comme produit de puissances de nombres premiers :

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

Fonction Phi d'Euler $\phi(n)$: Nombre d'entiers positifs plus petits que n qui sont premiers avec n.

Pour calculer $\phi(n)$:

$$\phi(n) = \prod_{i=1}^{m} p_i^{e_i} \cdot \left(1 - \frac{1}{p_i}\right)$$

Cas particulier important : Si $n = p \cdot q$ avec p et q premiers, alors :

$$\phi(n) = (p-1)(q-1)$$

Texte Original

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 :

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

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

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

En particulier, si $n = p \cdot q$ avec p et q premiers, alors :

$$\phi(n) = (p-1)(q-1)$$

Révision Rapide

- **Décomposition unique** : tout entier = produit de nombres premiers
- $\phi(n)$: compte les entiers $< n$ premiers avec n
- **Clé pour RSA** : si $n = pq$ (premiers) alors $\phi(n) = (p-1)(q-1)$

Théorèmes d'Euler et de Fermat

Ces théorèmes sont au cœur du fonctionnement de RSA et d'autres algorithmes asymétriques.

Théorème d'Euler : Si $n \in \mathbb{Z}^+$ et $a \in \mathbb{Z}$ avec $\text{pgcd}(a, n) = 1$, alors :

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

Petit Théorème de Fermat (cas particulier si $n = p$ premier) : Si $a \in \mathbb{Z}$ et p premier ne divise pas a :

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

Applications importantes :

1. **Réduction des exposants** : Si n est produit de premiers distincts et $r \equiv s \pmod{\phi(n)}$, alors :
$$a^r \equiv a^s \pmod{n}$$
2. **Calcul des inverses** : $a^{\phi(n)-1}$ est l'inverse de a modulo n. En particulier, si p est premier, a^{p-2} est l'inverse de a modulo p.

Texte Original

Fondements Mathématiques (II)

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 :

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

À 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 modulo n**. En particulier, a^{p-2} est l'inverse de a modulo n si p est premier.

Révision Rapide

- **Théorème d'Euler** : $a^{\phi(n)} \equiv 1 \pmod{n}$
- **Fermat** : cas spécial si p premier : $a^{p-1} \equiv 1 \pmod{p}$
- **Inverse modulaire** : $a^{-1} \equiv a^{\phi(n)-1} \pmod{n}$
- **Base de RSA** : permet encryption/decryption avec exposants

---

Groupes Multiplicatifs et Générateurs

Groupe multiplicatif $\mathbb{Z}_n^*$: Ensemble des éléments de $\mathbb{Z}_n$ premiers avec n :

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \text{pgcd}(a, n) = 1\}$$

Si n est premier : $\mathbb{Z}_n^* = \{1, 2, \ldots, n-1\}$

Ordre d'un élément : Plus petit entier positif t tel que $a^t \equiv 1 \pmod{n}$

Générateur : Un élément α est un générateur de $\mathbb{Z}_n^*$ si son ordre est $\phi(n)$. On dit alors que $\mathbb{Z}_n^*$ est **cyclique**.

Propriétés des générateurs :

1. $\mathbb{Z}_n^*$ a un générateur ssi $n = 2, 4, p^k$ ou $2p^k$ (avec p premier, $p \neq 2$ et $k \geq 1$)
2. Si p est premier, $\mathbb{Z}_p^*$ a toujours un générateur
3. Si α est générateur, tous les éléments s'écrivent : $\mathbb{Z}_n^* = \{\alpha^i \bmod n \mid 0 \leq i < \phi(n)\}$
4. Le nombre de générateurs est $\phi(\phi(n))$

Test de générateur

- α est un générateur de $\mathbb{Z}_n^*$ ssi $\forall$ premier p divisant $\phi(n)$, $\alpha^{\phi(n)/p} \not\equiv 1 \pmod{n}$
- si $n = 2p + 1$ est un "safe prime" avec p premier : α est générateur ssi $\alpha^2 \not\equiv 1 \pmod{n}$ et $\alpha^p \not\equiv 1 \pmod{n}$

Texte Original

Fondements Mathématiques (III)

Définition : Le **groupe multiplicatif de** $\mathbb{Z}_n$, noté $\mathbb{Z}_n^*$ est :

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \text{pgcd}(a, n) = 1\}$$

En particulier, si n est premier : $\mathbb{Z}_n^* = \{a \mid 1 \leq a \leq n - 1\}$
Le **nombre d'éléments ou ordre** du groupe multiplicatif $\mathbb{Z}_n^*$ est $\phi(n)$ (par définition 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 :

$$\mathbb{Z}_n^* = \{\alpha^i \bmod n \mid 0 \leq i \leq \phi(n) - 1\}$$

- Le nombre de générateurs de $\mathbb{Z}_n^*$ est $\phi(\phi(n))$.

- α est un générateur de $\mathbb{Z}_n^*$ ssi pour tout premier p divisant $\phi(n)$, on a :

$$\alpha^{\phi(n)/p} \not\equiv 1 \pmod{n}$$

En particulier si n est un premier de la forme $n = 2p + 1$ 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 $\alpha^p \not\equiv 1 \pmod{n}$.

Révision Rapide

- $\mathbb{Z}_n^*$: éléments premiers avec n, cardinal = $\phi(n)$
- **Générateur** : élément d'ordre $\phi(n)$ (génère tout le groupe)
- **Crucial pour DH et ElGamal** : sécurité basée sur logarithme discret dans groupe cyclique
- **Safe prime** : $n = 2p + 1$ avec p et n premiers

Fast Exponentiation (Exponentiation Rapide)

Calcul efficace de $a^k \bmod n$ en temps polynomial, essentiel pour tous les algorithmes asymétriques.

Principe : Utiliser la représentation binaire de l'exposant k.

Exemple : Calcul de $2^{644} \bmod 645$

1. Représentation binaire : $(644)_{10} = (1010000100)_2$
2. Calculer les puissances de 2 successives modulo 645 :
   - $2^1 \bmod 645$
   - $2^2 \bmod 645$
   - $2^4 \bmod 645$
   - $2^8 \bmod 645$
   - ...
   - $2^{512} \bmod 645$
3. Combiner selon les bits à 1 : $2^{644} = 2^{512} \cdot 2^{128} \cdot 2^4$

Complexité : $O(\log^3 n)$ - très efficace !

Application : Calcul de l'inverse avec le théorème d'Euler en temps polynomial.

Alternative : **Algorithme d'Euclide étendu** pour trouver x tel que $ax \equiv 1 \pmod{n}$ en résolvant $ax - kn = 1 = \text{pgcd}(a, n)$. Complexité également $O(\log^3 n)$.

Texte Original

Fast Exponentiation

Fast exponentiation : En utilisant la représentation binaire d'un nombre, on peut calculer des puissances très efficacement.
Exemple : calcul de $2^{644} \bmod 645$
$(644)_{10} = (1010000100)_2$
Maintenant, on calcule les exposants correspondants aux puissances de 2, soient :

$$2^1 \bmod 645, \quad 2^2 \bmod 645, \quad 2^4 \bmod 645, \quad \ldots, \quad 2^{512} \bmod 645$$

D'après la représentation binaire, on calcule :

$$2^{644} = 2^{512+128+4} = 2^{512} \cdot 2^{128} \cdot 2^4 = 160 \cdot 153 \cdot 6 \bmod 645$$

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 tel que :

$$ax \equiv 1 \pmod{n}$$

puisque cette congruence s'écrit : $ax - 1 = kn$ et donc :

$$ax - kn = 1 = \text{pgcd}(a, n)$$

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

Révision Rapide

- **Idée** : représentation binaire de l'exposant
- **Complexité** : $O(\log^3 n)$ - polynomial !
- **Essentiel** : rend RSA, ElGamal, DH praticables
- **Alternative** : algorithme d'Euclide étendu pour inverses

---

Théorème des Restes Chinois (CRT)

Le CRT permet de résoudre des systèmes de congruences simultanées, avec des applications importantes en cryptographie.

Théorème : Soient $n_1, n_2, \ldots, n_t \in \mathbb{Z}^+$ premiers deux à deux ($\text{pgcd}(n_i, n_j) = 1$ si $i \neq j$) et $a_1, a_2, \ldots, a_t \in \mathbb{Z}$. Alors le système :

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \vdots \\ x \equiv a_t \pmod{n_t} \end{cases}$$

a une solution unique $x \bmod N$ avec $N := n_1 \cdot n_2 \cdots n_t$.

Algorithme de Gauss (1801) pour calculer x :

$$x = \sum_{i=1}^{t} a_i N_i M_i \bmod N$$

avec :

- $N_i = N/n_i$
- $M_i = N_i^{-1} \bmod n_i$ (inverse modulaire)

Complexité : $O(\log^3 n)$ - polynomial !

Applications cryptographiques :

1. Accélération des calculs RSA (utiliser p et q séparément)
2. Partage de secret (secret sharing schemes)
3. Certaines attaques sur RSA (si exposant petit et messages multiples)

> Texte Original
>
> **Théorème des Restes Chinois**
>
> Le **Théorème des Restes Chinois** (IIIe 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 $n_1, n_2, \ldots, n_t \in \mathbb{Z}^+$ premiers deux à deux (c.à.d.

$\text{pgcd}(n_i, n_j) = 1$, $\forall i \neq j$) et $a_1, a_2, \ldots, a_t \in \mathbb{Z}$. Alors, le système de congruences :

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \vdots \\ x \equiv a_t \pmod{n_t} \end{cases}$$

a une solution unique $x \bmod N := n_1 n_2 \cdots n_t$
Algorithme de Gauss (1801) pour le calcul de x :

$$x = \sum_{i=1}^{t} a_i N_i M_i \bmod N$$

avec $N_i = N/n_i$ et $M_i = N_i^{-1} \bmod n_i$.
La **complexité** de cet algorithme est $O(\log^3 n)$.
Il est donc possible en **temps polynomial** de passer des congruences mod n_i aux congruences mod N !

Révision Rapide

- **Résout** : systèmes de congruences avec moduli premiers entre eux
- **Solution unique** : modulo produit des moduli
- **Complexité** : $O(\log^3 n)$ (polynomial)
- **Usage crypto** : optimisation RSA, attaques si petit exposant

---

Problèmes de Base et Complexité

Classification des Problèmes Difficiles

La sécurité de la cryptographie asymétrique repose sur des problèmes mathématiques réputés difficiles :

Problèmes génériques :

1. **Factorisation (FACTP)** : Étant donné n, trouver sa factorisation en nombres premiers
   - Base de **RSA** et **Rabin**
2. **Logarithmes discrets (DLP)** : Étant donné p premier, un générateur $\alpha \in \mathbb{Z}_p^*$ et $\beta \in \mathbb{Z}_p^*$, trouver x tel que :
$$\alpha^x \equiv \beta \pmod{p}$$

- Base de **ElGamal** et **Diffie-Hellman**

3. **Racine carrée modulo composite (SQROOTP)** : Étant donné n composite et un résidu quadratique a, trouver $\sqrt{a} \bmod n$
   - Base de **Rabin**

Problèmes spécifiques :

1. **RSA Problem (RSAP)** : Étant donné $n = pq$, e avec $\text{pgcd}(e, \phi(n)) = 1$ et c, trouver m tel que $m^e \equiv c \pmod{n}$
2. **Diffie-Hellman Problem (DHP)** : Étant donné p premier, α générateur, $\alpha^a \bmod p$ et $\alpha^b \bmod p$, trouver $\alpha^{ab} \bmod p$

Équivalences prouvées :

- **DHP** **DLP** (équivalent sous certaines conditions)
- **RSAP** **FACTP** (prouvé équivalent pour le cas générique)
- **SQROOTP** **FACTP**

Texte Original

Problèmes de Base

Problèmes génériques principaux :

- **Factorisation (FACTP)** : Étant donné un entier positif n, trouver sa factorisation en nombres premiers.
- **Logarithmes discrets (DLP)** : Étant 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)** : Étant donné un entier composite n et un résidu quadratique a, trouver la racine carrée de a mod n.

Problèmes spécifiques (propres à un système de cryptage) :

- **RSA (RSAP)** : Étant 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)** : Étant donné un nombre premier p, un générateur $\alpha \in \mathbb{Z}_p^*$ et les éléments $\alpha^a \bmod p$ et $\alpha^b \bmod p$, trouver $\alpha^{ab} \bmod p$.

Résultats prouvés :

- **DHP** **DLP** (Équivalent sous certaines conditions)
- **RSAP** **FACTP** (Prouvé équivalent pour le problème générique)
- **SQROOTP** **FACTP**

Révision Rapide

- **FACTP** : factoriser $n \rightarrow$ base de RSA/Rabin
- **DLP** : trouver logarithme discret $\rightarrow$ base ElGamal/DH
- **SQROOTP** : racine carrée mod composite $\rightarrow$ Rabin
- **Équivalences** : cassage = résolution du problème de base

---

Techniques de Factorisation

La sécurité de RSA dépend de la difficulté de factoriser de grands nombres.

Méthodes à temps exponentiel : $O(\exp(c \cdot \ln(n)))$

- Trial Division (division successive)
- Crible d'Ératosthène (IIe siècle av. J.-C.)
- Méthode de Fermat (~1650)
- Méthode ρ de Pollard (1975)
- Méthode $p-1$ de Pollard (1974)

Méthodes à temps sous-exponentiel : $O(\exp(c \cdot (\ln(n))^{1/3}))$

- Continued Fractions (1975)
- **Quadratic Sieve (1981)** - très efficace en pratique
- **Number Field Sieve - NFS (1990)** - le plus rapide actuellement
- General Number Field Sieve - GNFS (2006)

Méthodes à temps polynomial :

- **Algorithme de Shor** (1994) : $O(\log^c n)$ sur **ordinateur quantique**

Records actuels (2020) :

- Plus grand nombre factorisé : **RSA-829** (250 chiffres, 829 bits)
- Temps de calcul : 2700 années-cœur (CPUs Intel Xeon Gold 6130)
- Méthode : General Number Field Sieve

Implications :

- Clés RSA < 1024 bits : **vulnérables**
- Clés RSA 1024 bits : **limites** (états avec ressources importantes)
- Recommandation : **2048 bits minimum** (3072-4096 pour long terme)

Texte Original

Classical Factoring Techniques et New Developments

Temps exponentiel : $O(\exp(c \cdot \ln(n)))$

- 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)

Temps sous-exponentiel : $O(\exp(c \cdot (\ln(n))^{1/3}))$

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

Temps polynomial :

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

Développements récents :

- L'ordinateur NFS spécifique de Bernstein pour factoriser un nombre de 1536 bits prendrait le même temps qu'un calcul 512 bits sur machine conventionnelle
- **Plus grande factorisation à ce jour (2020)** : RSA-829 (nombre de 250 chiffres) utilisant NFS
- Temps de calcul total : **2700 années-cœur** (CPUs Intel Xeon Gold 6130 à 2.1GHz)

Factorisation sur ordinateur quantique :

- Problèmes significatifs (erreurs, dispersion, etc.)
- 2001 : ordinateur 7 qubits (IBM Almaden)
- Faisabilité d'un ordinateur avec millions de qubits… ?

Révision Rapide

- **Sous-exponentiel** : NFS actuellement le plus rapide
- **Record 2020** : RSA-829 (829 bits) en 2700 années-cœur
- **Recommandation** : clés 2048 bits pour RSA

- **Menace future** : ordinateurs quantiques (Shor)

L'Algorithme RSA

Fonctionnement de RSA (Encryption/Decryption)

RSA (Rivest-Shamir-Adleman, 1978) est l'algorithme asymétrique le plus utilisé.

Génération des clés :

1. Choisir deux nombres premiers **grands** p et q (1024 bits chacun)
2. Calculer $n := p \cdot q$ et $\phi(n) = (p-1)(q-1)$
3. Choisir exposant d'encryption e avec :
   - $1 < e < \phi(n)$
   - $\text{pgcd}(e, \phi(n)) = 1$
4. Calculer exposant de décryption d tel que :

$$e \cdot d \equiv 1 \pmod{\phi(n)}$$

   (avec algorithme d'Euclide étendu ou fast exponentiation)

Clés résultantes :

- Clé **publique** : (n, e)
- Clé **privée** : d (garder p et q secrets aussi !)

Encryption (par Bob, vers Alice) :

1. Obtenir clé publique authentique (n, e) d'Alice
2. Transformer plaintext en entiers $m_i \in [0, n-1]$
3. Calculer ciphertexts : $c_i := m_i^e \bmod n$
4. Envoyer les c_i à Alice

Decryption (par Alice) :

- Utiliser clé privée d pour calculer :

$$m_i = c_i^d \bmod n$$

Preuve de fonctionnement :

$$c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$$

Comme $ed \equiv 1 \pmod{\phi(n)}$, il existe k tel que $ed = 1 + k\phi(n)$, donc :

$$c^d \equiv m^{1+k\phi(n)} \equiv m \cdot (m^{\phi(n)})^k \equiv m \cdot 1^k \equiv m \pmod{n}$$

(par le théorème d'Euler)

Texte Original

Procédé d'Encryption/Decryption de RSA et Preuve

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 $(n/2)$.
  - A calcule $n := pq$ et $\phi(n) = (p-1)(q-1)$.
  - A génère l'exposant d'encryption e, avec $1 < e < \phi(n)$ t.q. pgcd$(e, \phi(n)) = 1$.
  - 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.
- 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 $c_i := m_i^e \bmod n$, $\forall i$ avec l'algorithme fast exponentiation.
- B envoie à A tous les ciphertext c_i.

Decryption :

- A utilise sa clé privée pour calculer les plaintexts $m_i = c_i^d \bmod n$.

Preuve : Soit m le plaintext et c le ciphertext avec $c := m^e \bmod n$, il s'agit de prouver : $m \overset{!}{=} c^d \bmod n$
En substituant c par sa valeur on obtient :

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

mais, on sait que :

$$ed \equiv 1 \pmod{\phi(n)}$$

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

$$ed - 1 = k\phi(n)$$

en substituant dans (*) :

$$c^d \equiv m^{k\phi(n)+1} \equiv m^{k\phi(n)} \cdot m \pmod{n}$$

Si $\text{pgcd}(m, n) = 1$, on a par le **théorème d'Euler** :

$$m^{\phi(n)} \equiv 1 \pmod{n}$$

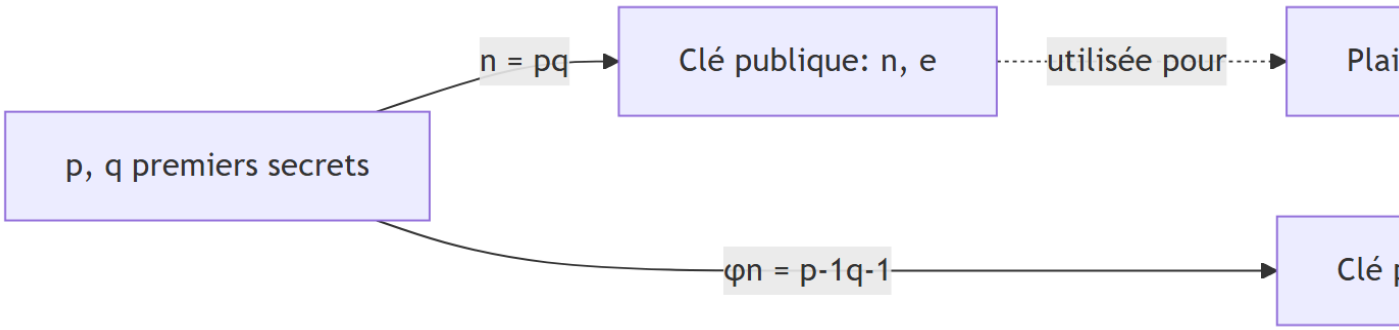
donc :

$$c^d \equiv (m^{\phi(n)})^k \cdot m \equiv m \pmod{n}$$

c.q.f.d. !
Si $\text{pgcd}(m, n) \neq 1$, 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.

Révision Rapide

- **Clé publique** : (n, e) avec $n = pq$
- **Clé privée** : d tel que $ed \equiv 1 \pmod{\phi(n)}$
- **Chiffrement** : $c = m^e \bmod n$
- **Déchiffrement** : $m = c^d \bmod n$
- **Sécurité** : basée sur difficulté de factoriser n



Sécurité de RSA

Équivalence problème RSA Factorisation :

- Trouver d factoriser n (prouvé équivalent)
- Décrypter sans d n'est **pas prouvé** aussi difficile que factoriser, mais…
- Aucune méthode plus rapide que factorisation n'est connue

Complexité de la factorisation :

- Méthodes les plus rapides : $O(\exp(c \cdot (\ln(n))^{1/3}))$ (sous-exponentiel)
- Calculatoirement impossible pour $n \geq 1024$ bits
- **Recommandation actuelle** : 2048 bits minimum (3072-4096 pour sécurité durable)

Choix des exposants :

- **Exposant d'encryption** e :
  - Souvent **petit** pour accélérer : $e = 3, 17, 65537$ (commun)
  - Attention : si e trop petit ET $m < n^{1/e}$, attaque possible (racine e-ième dans $\mathbb{Z}$)
  - Solution : **randomization** (padding) du message
- **Exposant de décryption** d :
  - Doit être **grand** : au moins la moitié de la taille de n
  - Si d petit : vulnérable à l'attaque de Wiener

Conséquence performance :

- **Encryption rapide** (e petit)
- **Decryption lente** (d grand)

Texte Original

RSA: Sécurité

Le problème **RSAP** 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 ont une **complexité sub-exponentielle** $O(\exp(c \cdot (\ln(n))^{1/3}))$. 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. Par contre, en 2008 on a prouvé que la résolution générique du problème RSA est équivalent à la factorisation.
- L'**exposant de decryption** d **doit impérativement être de grande taille** (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 !

Révision Rapide

- **Sécurité** : basée sur difficulté de FACTP (factorisation)
- **Taille recommandée** : $n \geq 2048$ bits
- e **petit** : encryption rapide (3, 17, 65537)
- d **grand** : au moins taille$(n)/2$
- **Clés séparées** : encryption signature

Attaques sur RSA

Attaque sur exposant petit avec même message

Si on envoie le même message m à 3 destinataires avec $e = 3$:

- $c_1 \equiv m^3 \pmod{n_1}$
- $c_2 \equiv m^3 \pmod{n_2}$
- $c_3 \equiv m^3 \pmod{n_3}$

Le **Théorème des Restes Chinois** donne une solution unique $x \bmod n_1 n_2 n_3$ telle que :

$$x \equiv c_1 \pmod{n_1}, \quad x \equiv c_2 \pmod{n_2}, \quad x \equiv c_3 \pmod{n_3}$$

Si $m^3 < n_1 n_2 n_3$ (souvent vrai), alors $x = m^3$ dans $\mathbb{Z}$ et on peut calculer m en prenant simplement la racine cubique entière !

Protection : toujours randomizer le message avant encryption (padding OAEP)

Attaque si message petit

Si $m < n^{1/e}$, alors $m^e < n$, donc $c = m^e$ (dans $\mathbb{Z}$, pas modulo). On peut calculer directement la racine e-ième !

Protection : padding obligatoire

Propriété multiplicative

$$E(m_1) \cdot E(m_2) \equiv (m_1 \cdot m_2)^e \equiv E(m_1 \cdot m_2) \pmod{n}$$

Permet des attaques de type chosen-ciphertext et blind signatures.

Attaque générale

La méthode la plus efficace reste la **factorisation de** n (si paramètres bien choisis et implémentation correcte).

Texte Original

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 suivante :
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. :

$$x \equiv c_1 \pmod{n_1}, \quad x \equiv c_2 \pmod{n_2}, \quad x \equiv c_3 \pmod{n_3}$$

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 : $(m_1 m_2)^e \equiv m_1^e \cdot m_2^e \equiv c_1 \cdot c_2 \pmod{n}$
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.

Révision Rapide

- **Même message, petit** e : CRT permet d'extraire m !
- **Message trop petit** : $m < n^{1/e} \rightarrow$ racine directe
- **Propriété multiplicative** : $E(m_1) \cdot E(m_2) = E(m_1 m_2)$
- **Protection** : toujours padding/randomization (OAEP)

L'Algorithme ElGamal

Système asymétrique (1985) basé sur le **problème du logarithme discret (DLP)**.

Clés :

- Choisir premier p, générateur $\alpha \in \mathbb{Z}_p^*$, secret a
- Calculer $y = \alpha^a \bmod p$
- **Publique** : (p, α, y) | **Privée** : a

Chiffrement : Pour message m, choisir aléatoire k unique

- $\gamma = \alpha^k \bmod p$
- $\delta = m \cdot y^k \bmod p$
- Envoyer (γ, δ)

Déchiffrement : $m = \delta \cdot \gamma^{-a} \bmod p$

Texte original du cours

Procédé d'Encryption/Decryption d'ElGamal

Génération des clés
Chaque entité (A) crée une paire de clés (publique et privée) comme suit:

- A génère un nombre premier p (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 $y := \alpha^a \bmod p$
- La **clé publique** de A est (p, α, y), la **clé privée** de A est a

Encryption

- L'entité B obtient $(p, \alpha, \alpha^a \bmod p)$, la clé publique authentique de A
- B transforme son plaintext en une série d'entiers m_i, t.q. $m_i \in [0, p-1]$ $\forall i$
- Pour chaque message m_i :

- B génère un nombre aléatoire **unique** k, t.q. $1 \leq k \leq p-2$
- B calcule $\gamma := \alpha^k \bmod p$ et $\delta := m_i \cdot (\alpha^a)^k \bmod p$ et envoie le ciphertext $c := (\gamma, \delta)$

Decryption

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

Révision rapide

Base : DLP dans $\mathbb{Z}_p^*$
Chiffré : $(\alpha^k, m \cdot y^k)$
Sécurité : k doit être unique et grand
Inconvénient : double la taille du message

Remarques essentielles

- **Preuve** : $\delta \cdot \gamma^{-a} = m \cdot (\alpha^a)^k \cdot (\alpha^k)^{-a} = m \bmod p$
- **Sécurité** : basée sur DLP (complexité sub-exponentielle proche de la factorisation)
- **Exposants** : k et a doivent être grands (sinon vulnérable à baby-step giant-step)
- **Réutilisation interdite** : si k répété, $\delta_1/\delta_2 = m_1/m_2$ révèle les messages
- **Inconvénient majeur** : expansion $\times 2$ de la taille du chiffré
- **Généralisation** : fonctionne sur $GF(2^n)$ ou courbes elliptiques

Texte original - Remarques

Preuve que le schéma fonctionne : Si $s \equiv k^{-1}(m_h - ar) \bmod (p-1)$, on a que: $m_h \equiv (ar + ks) \bmod (p-1)$ et $v_2 = \alpha^{H(m)} \bmod p$. Si, comme on souhaite montrer $m_h = H(m)$, en réduisant les exposants mod $(p-1)$, on peut réécrire v_2: $v_2 \equiv \alpha^{ar+ks} \bmod p$. D'autre part: $v_1 = y^r \alpha^{rs} \equiv \alpha^{ar}\alpha^{ks} \equiv \alpha^{ar+ks} \bmod p$.
Le procédé d'ElGamal se base sur la difficulté de calculer des **logarithmes discrets modulo un nombre premier** (problème DLP) même s'il n'a pas été prouvé qu'il soit strictement équivalent à ce problème.
Les **algorithmes les plus efficaces** connus 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).

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 (γ_1, δ_1) et (γ_2, δ_2) les deux ciphertexts générés, on a que $\delta_1/\delta_2 = m_1/m_2$ 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 $GF(2^n)$ ou les courbes elliptiques.

Révision rapide - Remarques

Équivalence : basé sur DLP (non prouvé équivalent)
k **unique** : CRITIQUE - sinon m_1/m_2 révélé
Taille clés : exposants grands nécessaires
Extensions : $GF(2^n)$, courbes elliptiques

Algorithme de Rabin

Système asymétrique **équivalent à la factorisation** (provably secure).

Clés :

- Générer deux premiers p, q (1024 bits total), calculer $n = pq$
  - **Publique** : n
  - **Privée** : (p, q)

Chiffrement : $c = m^2 \bmod n$

Déchiffrement :

- Calculer les 4 racines carrées de $c \bmod n$ (via racines mod p et mod q)
- Identifier le bon message par redondance

Texte original du cours

Procédé d'Encryption/Decryption de Rabin

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 (len(pq) 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 $c_i = m_i^2 \bmod n$ 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: $c_i = x^2 \bmod n$ 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 m_1, m_2, m_3, m_4 est le plaintext original

Révision rapide

Base : SQROOTP (racine carrée mod composite)
Avantage : équivalent prouvé à factorisation
Problème : 4 solutions possibles, nécessite redondance
Vulnérabilité : attaque chosen-ciphertext révèle facteurs

Remarques essentielles

- **Sécurité prouvée** : SQROOTP FACTP (seul algorithme avec équivalence prouvée)
- **Attaque chosen-ciphertext** : si A décrypte $c = m^2 \bmod n$ choisi par adversaire M
  - M reçoit une racine m_x parmi 4 possibles
  - Si $m \neq m_x \bmod n$ (prob. 0.5), alors $\gcd(m - m_x, n)$ donne un facteur de n
- **Solution** : exiger redondance suffisante pour identifier solution unique sans ambiguïté

Texte original - Remarques

Le procédé de Rabin est basé sur l'**impossibilité de trouver des racines carrées modulo un composite de factorisation inconnue** (problème SQROOTP).
L'**intérêt principal** de cet algorithme réside dans le fait qu'il a été **prouvé comme étant équivalent à la factorisation** (SQROOTP 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** suivant:

- L'attaquant M génère un m et envoie à A le ciphertext $c = m^2 \bmod n$.
- A répond avec une racine m_x parmi les 4 possibles m_1, m_2, m_3, m_4.
- Si $m \neq m_x \bmod n$ (probabilité 0.5), M recommence avec un nouveau m.
- Sinon, A calcule $\gcd(m - m_x, n)$ 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.

Révision rapide - Remarques

Unique : seul algorithme prouvé équivalent à FACTP
Attaque : chosen-ciphertext donne facteurs (prob. 0.5)
Parade : redondance obligatoire dans messages

Comparaison RSA - ElGamal - Rabin

Critère	RSA	ElGamal	Rabin
Problème	RSAP	DLP	SQROOTP
Sécurité	Équiv. factorisation (cas générique)	Basée sur DLP	**Prouvée** factorisation
Expansion	1:1	**1:2**	1:1
Déchiffrement	Déterministe	Déterministe	**4 solutions**
Signature	Oui	Oui	Oui (avec précautions)

Courbes Elliptiques (Idée de base)

Concept fondamental

Une **courbe elliptique** E est définie par : $y^2 = x^3 + ax + b$ (avec discriminant $4a^3 + 27b^2 \neq 0$).

Opération clé : Addition de points

- Géométriquement : tracer une droite entre deux points P et Q, trouver le 3 point d'intersection, puis prendre son symétrique
- Forme un **groupe commutatif** avec point à l'infini $\mathcal{O}$ comme identité
- **Multiplication scalaire** : $kP = P + P + ... + P$ (k fois)

Avantage cryptographique :

- Le **problème ECDLP** : trouver k tel que $Q = kP$ est très difficile (effort exponentiel)
- **Clés plus courtes** pour même sécurité qu'en $\mathbb{Z}_p^*$

Texte original - Définition

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 $\mathcal{O}$. Le point $\mathcal{O}$ 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** $4a^3 + 27b^2 \neq 0$.

Révision rapide - Concept

Équation : $y^2 = x^3 + ax + b$
Structure : groupe avec $\mathcal{O}$
Opération : addition géométrique
Problème dur : ECDLP

Addition sur courbes elliptiques

Soit $P := (x, y) \in E$, on définit $-P := (x, -y)$ (symétrique par rapport à l'axe des x). On a $P + (-P) = \mathcal{O}$.

Pour deux points $P, Q \in E$ avec $Q \neq -P$, on définit $P + Q := R$ où $-R$ est le 3 point d'intersection entre la courbe et la droite passant par P et Q.

Pour le **doublement** : $2P = R$ où $-R$ est le point d'intersection de la courbe avec la tangente à la courbe au point P.

Texte original - 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. À noter que $P + (-P) = \mathcal{O}$.
Soient deux points $P, Q \in E$, tels que $Q \neq -P$, on définit l'addition $P + Q := R$ où $R \in E$ tel que $-R$ est le 3 point d'intersection entre la courbe et la droite qui passe par P et Q.
L'ensemble E avec $\oplus$ définit un **groupe commutatif** pour l'addition.
Soit $P \in E$, le point $2P = R$, tel que $-R$ est le point d'intersection de la courbe avec la droite tangente à la courbe au point P.

Révision rapide - Addition

Inverse : $-P = (x, -y)$
Addition : 3 point d'intersection + symétrie
Doublement : tangente + symétrie
Propriété : groupe commutatif

ECDLP et avantages cryptographiques

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

Avantage principal : taille des clés beaucoup plus petite pour une sécurité équivalente.

Texte original - ECDLP et avantages

Lorsque la courbe elliptique est définie sur le corps $\mathbb{Z}_p$ avec p un nombre premier de grande taille ($y^2 \equiv x^3 + ax + b \bmod p$), le calcul de $k \in \mathbb{Z}_p$ tel que $Q = kP$ avec (P, Q) connus, est très difficile (nécessite un effort exponentiel). Ce problème est connu comme: **Elliptic Curve Discrete Logarithm Problem (ECDLP)**.
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 $\mathbb{Z}_p$) par rapport aux corps habituels tels que $\mathbb{Z}_p$ ou $GF(2^m)$.
La **représentation d'un plaintext en points** de la courbe reste une opération complexe.
En Octobre 2003, 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 montré que sous certaines conditions le problème ECDLP pouvait être résolu en temps **sub-exponentiel**.

Révision rapide - ECDLP

Problème : trouver k dans $Q = kP$ (exponentiel)
Gain : clés ~6-10× plus courtes
Limite : représenter messages en points difficile
NSA : adopté en 2003

Tableau de comparaison des tailles de clés

AES (symétrique)	RSA/DH	Courbes Elliptiques	Rapport
56 bits	512 bits	112 bits	1:4.6
80 bits	1024 bits	160 bits	1:6.4
112 bits	2048 bits	224 bits	1:9.1
128 bits	3072 bits	256 bits	1:12
256 bits	15360 bits	512 bits	1:30

Texte original - Tableau

Ce tableau montre les rapports des tailles des clés par rapport à celles de RSA pour une sécurité équivalente.
(Tableau extrait du document original)

ElGamal sur Courbes Elliptiques

Adaptation directe

Remplacer opérations dans $\mathbb{Z}_p^*$ par opérations sur E_p

Clés :

- Choisir courbe E_p et point $P_0 \in E_p$ de grand ordre
- Secret x, calculer $P_a = xP_0$
- **Publique** : (E_p, P_0, P_a) | **Privée** : x

Chiffrement : Pour message $m_i \in E_p$

- Choisir k aléatoire

- $\gamma = kP_0$, $\delta = kP_a + m_i$
- Envoyer (γ, δ)

Déchiffrement : $m_i = \delta - x\gamma$

Texte original - ElGamal EC

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 (len(p) bits) et un point $P_0 \in E_p$.
- A génère un nombre aléatoire x, tel que $1 \leq x \leq p$ et calcule $P_a = xP_0$ (multiplication par un scalaire sur E_p, pour laquelle, il existe des algorithmes efficaces).
- La clé publique de A est (E_p, P_0, P_a), la clé privée de A est x.

Encryption
L'entité B obtient (E_p, P_0, P_a), la clé publique authentique de A.

- B transforme son plaintext en une série d'entiers m_i, tel que $m_i \in E_p$ pour tout i.
- Pour chaque message m_i :
  - B génère un nombre aléatoire unique k, tel que $1 \leq k \leq p$.
  - B calcule $\gamma := kP_0$ et $\delta := kP_a + m_i$ et envoie le ciphertext $c := (\gamma, \delta)$.

Decryption

- A utilise sa clé privée x pour calculer: $x\gamma = xkP_0 = kP_a$.
- 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** !
Il est également nécessaire d'**authentifier** les parties publiques échangées afin d'éviter les attaques man-in-the middle précédemment décrites.
Les propriétés du protocole sont identiques au cas $\mathbb{Z}_p^*$.

Révision rapide - ElGamal EC

Principe : même qu'ElGamal sur E_p
Opérations : $+$ et $\times$ scalaire sur points
Sécurité : ECDLP
Authentification : nécessaire contre MitM
Avantage : clés courtes