

## Table of contents

<b>Signatures Digitales</b>	<b>1</b>
Introduction et Définitions . . . . .	1
Cadre Formel : Signatures avec Appendice . . . . .	2
Cadre Formel : Signatures avec Reconstitution . . . . .	3
Signature RSA . . . . .	5
Signatures Aveugles (Blind Signatures) . . . . .	7
Signature Rabin . . . . .	8
Signature ElGamal . . . . .	10
Signatures et Crypto-monnaies . . . . .	12
Tableau Récapitulatif des Schémas . . . . .	13
Types d'Attaques . . . . .	14

## Signatures Digitales

### Introduction et Définitions

Une **signature digitale** est une chaîne de données associant un message à une entité d'origine, équivalent numérique d'une signature manuscrite.

Classification :

- **Avec appendice** : nécessite le message original pour vérification (ElGamal, DSS)
- **Avec reconstitution** : permet de reconstruire le message (RSA, Rabin)

Les signatures utilisent principalement la **cryptographie asymétrique** pour identifier explicitement une entité.

**arbitrated digital signatures**: technologie symétrique + Thrusted Third Parties (TPP)

#### Texte original

**Signature digitale** : chaîne de données permettant d'associer un message (sous forme digitale) à une entité d'origine.

**Schéma de signature digitale** : algorithme de génération + algorithme de vérification.

**Procédé de signature** : formatage du message + algorithme de génération de signature.

**Procédé de vérification** : algorithme de vérification + (reconstruction du message).

**Classification des signatures digitales** :

- **Signatures digitales avec appendice** qui nécessitent la présence du message original pour vérifier la validité de la signature. Ce sont les plus couramment utilisées.  
Exemples : ElGamal, DSS.

- **Signatures digitales avec reconstitution du message** qui offrent, en plus, la possibilité de reconstruire le message à partir de la signature. Exemples : RSA, Rabin.

Les signatures digitales sont pour la plupart basées sur la **crypto asymétrique** du fait que la notion clé partagée n'est pas adaptée aux besoins d'identifier une entité de façon explicite.

Des engagements semblables à ceux obtenus par une signature à clé publique (comme la non-répudiation d'origine) peuvent cependant être obtenus avec la technologie symétrique et des tierces de confiance (Trusted Third Parties ou TTP). Ces méthodes sont nommées : **arbitrated digital signatures**.

### Révision rapide

**Signature digitale** = chaîne associant message + entité

Deux types

- avec appendice (nécessite message original)
- avec reconstitution (reconstruit le message)

Basée sur crypto asymétrique

## Cadre Formel : Signatures avec Appendice

Espaces de travail :

- $M$  : espace des messages
- $M_h$  : messages hashés où  $m_h = H(m)$  avec  $H$  une fonction de hachage
- $S$  : espace des signatures possibles

Fonctionnement :

Chaque entité A définit :

- $S_A : M_h \rightarrow S$  (application de signature, utilise clé privée)
- $V_A : M_h \times S \rightarrow \{\text{vrai}, \text{faux}\}$  (vérification, utilise clé publique)

Avec  $V_A(m_h, s) = \text{vrai}$  si et seulement si  $S_A(m_h) = s$

Propriétés essentielles :

- $S_A$  et  $V_A$  faciles à calculer avec les bonnes clés

- Impossible de trouver  $(m', s')$  valide sans la clé privée de A

### Texte original

On admet que chaque entité a une clé privée pour signer des messages et une copie authentique des clés publiques des correspondants.

**Notation :** - M : Espace de messages -  $M_h : m_h = H(m)$  avec  $m \in M$ ,  $m_h \in M_h$  et  $H$  une hash function - S : Espace des valeurs pouvant être obtenues par un procédé de signature

### Description :

Chaque entité définit une application injective  $S_A : M_h \rightarrow S$  (ie. la signature)

L'application  $S_A$  donne lieu à une application  $V_A : V_A : M_h \times S \rightarrow \{\text{vrai, faux}\}$  (ie. la vérification)

tel que  $\forall m_h \in M_h, s \in S$ , on a :  $V_A(m_h, s) = \text{vrai}$  si  $S_A(m_h) = s$  et  $V_A(m_h, s) = \text{faux}$  sinon

Les opérations  $S_A$  nécessitent la clé privée de A alors que les opérations  $V_A$  utilisent la clé publique de A.

### Quelques propriétés simples :

- Les opérations  $S_A$  et  $V_A$  doivent être faciles à calculer (en ayant les clés corresp.)
- Il est impossible (calculatoirement) pour une entité n'ayant pas la clé privée de A de trouver un  $m'$  et un  $s'$  avec  $m' \in M$  et  $s' \in S$  tel que  $V_A(m'_h, s') = \text{vrai}$  avec  $m'_h = H(m')$ .

### Révision rapide

Signature :  $S_A(m_h) = s$  (clé privée).

Vérification :  $V_A(m_h, s)$  (clé publique).

Impossible de forger sans clé privée.

## Cadre Formel : Signatures avec Reconstitution

### Espaces additionnels :

- $M_S$  : espace sur lequel s'applique la signature
- $R : M \rightarrow M_S$  : fonction de redondance (injective, inversible, publique)
- $M_R = \text{Im}(R)$  : image de R

### Fonctionnement :

- Signature :  $S_A : M_S \rightarrow S$  (injective)

- Vérification :  $V_A : S \rightarrow M_S$  avec  $V_A \circ S_A = \text{Identité}$

**Génération :**

1. Calculer  $m_R = R(m)$  et  $s = S_A(m_R)$
2. Publier  $s$  comme signature de A sur  $m$

**Vérification :**

1. Calculer  $m_R = V_A(s)$  avec clé publique
2. Vérifier  $m_R \in M_R$  (sinon rejeter)
3. Reconstituer  $m = R^{-1}(m_R)$

**Fonction de redondance :**

Essentielle pour la sécurité. Si  $M_R = M_S$ , il devient trivial de forger des signatures.

**Exemple :**  $R(m) = m \parallel m$  (concaténation). Probabilité de forger :  $(1/2)^n$  pour un message de  $n$  bits.

#### Texte original

**Notation :** en plus des définitions précédentes, on a :

$M_S$  : L'espace des éléments sur lesquels peut s'appliquer une signature.

$R$  : Une application injective :  $M \rightarrow M_S$ , appelée fonction de redondance. Elle doit être inversible et publique.

$M_R = \text{Im}(R)$

#### Description :

Chaque entité définit une application injective  $S_A : M_S \rightarrow S$  (ie. la signature)

L'application  $S_A$  donne lieu à une application  $V_A : S \rightarrow M_S$  (ie. la vérification) tel que  $V_A \circ S_A = \text{Identité}$  sur  $M_S$

A noter que la vérification s'effectue sans la clé privée de A

**Génération de signature :** 1. Calculer  $m_R = R(m)$  et  $s = S_A(m_R)$  2. Rendre publique  $s$  en tant que signature de A sur  $m$ . Ceci permet aux autres entités de vérifier la signature et reconstituer  $m$ .

**Vérification :** 1. Calculer  $m_R = V_A(s)$  (avec la clé publique de A) 2. Vérifier que  $m_R \in M_R$  (sinon rejeter la signature) 3. Reconstituer  $m$  en calculant :  $R^{-1}(m_R)$

#### Propriétés :

- Les opérations  $S_A$  et  $V_A$  doivent être faciles à calculer (en ayant les clés corresp.)
- Il est impossible (calculatoirement) pour une entité n'ayant pas la clé privée de A de trouver un  $s' \in S$  tel que  $V_A(s') \in M_R$

#### Remarques sur la fonction de redondance :

- Le choix d'une fonction de redondance est **essentiel pour la sécurité** du système.

- Si  $M_R = M_S$  et  $R$  et  $S_A$  sont des bijections respectivement de  $M$  dans  $M_R$  et de  $M_S$  dans  $S$ , alors  $M$  et  $S$  ont une taille identique et, par conséquent, il est trivial de forger des messages portant la signature de A.

**Exemple de fonction de redondance :** soit  $M = \{m : m \in \{0, 1\}^n\}$  (n taille du message) et  $M_S = \{t : t \in \{0, 1\}^{2n}\}$ . Soit  $R : M \rightarrow M_S$  tel que  $R(m) = m \parallel m$  ( $\parallel$  étant la concaténation de 2 messages). La probabilité de tomber sur un tel message en essayant de forger un message à partir d'une signature est de :  $|M_R|/|M_S| = (1/2)^n$ , ce qui est négligeable pour des grands messages.

**Attention !** : Une fonction de redondance adaptée pour un schéma de signature digitale peut provoquer des failles dans un autre différent !

### Révision rapide

**Avec reconstitution :** Fonction redondance  $R(m) = m_R$ .

Signature  $s = S_A(m_R)$ .

Vérification :  $m_R = V_A(s)$ , reconstituer  $m = R^{-1}(m_R)$ .

Redondance cruciale pour sécurité.

## Signature RSA

### Génération des clés :

- Choisir deux grands nombres premiers  $p$  et  $q$
- Calculer  $n = pq$  et  $\phi(n) = (p - 1)(q - 1)$
- Choisir  $e$  avec  $\text{pgcd}(e, \phi(n)) = 1$
- Calculer  $d$  tel que  $ed \equiv 1 \pmod{\phi(n)}$
- **Clé publique** :  $(n, e)$  ; **Clé privée** :  $d$

### Signature :

1. Calculer  $m_R = R(m)$  (fonction de redondance)
2. Calculer  $s = m_R^d \pmod{n}$
3. Envoyer  $s$

### Vérification :

1. Calculer  $m'_R = s^e \pmod{n}$
2. Vérifier  $m'_R \in M_R$  (rejeter sinon)
3. Reconstituer  $m = R^{-1}(m'_R)$

**Variante avec appendice :**

- **Signature** :  $m_h = H(m)$ , puis  $s = m_h^d \pmod{n}$
- **Vérification** :  $m'_h = s^e \pmod{n}$ , accepter si  $m'_h = H(m)$

**Caractéristiques :**

- Signature plus lente que vérification ( $d$  grand,  $e$  petit)
- Différencier clés signature/encryption
- Vulnérable aux mêmes attaques que RSA encryption

### Texte original

**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 de vérification  $e$ , avec  $1 < e < \phi(n)$  tel que  $\text{pgcd}(e, \phi(n)) = 1$ .
- A calcule l'exposant de signature  $d$ , tel que :  $ed \equiv 1 \pmod{\phi(n)}$  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.

**Signature :**

- A calcule la fonction de redondance du message  $m$  :  $m_R := R(m)$ .
- A calcule la signature :  $s := m_R^d \pmod{n}$  et envoie  $s$  à B.

**Vérification :**

- L'entité B obtient  $(n, e)$ , la clé publique authentique de A.
- B calcule  $m'_R = s^e \pmod{n}$ , vérifie  $m'_R \in M_R$  et rejette la signature si  $m'_R \notin M_R$ .
- B retrouve le message correctement signé par A en calculant :  $m = R^{-1}(m'_R)$ .

**Remarques :**

La preuve de fonctionnement est identique à celle du procédé d'encryption. L'ordre d'exponentiation n'a pas d'influence puisque :

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

Le procédé peut également être utilisé pour produire des **signatures avec appendice** avec les modifications suivantes :

**Signature** : - A utilise une fonction de hachage  $H$  et calcule  $m_h := H(m)$ . - A calcule la signature de  $m_h$  :  $s := m_h^d \pmod{n}$  et envoie le couple  $(m, s)$  à B.

**Vérification :** - B calcule  $m'_h = s^e \bmod n$  et  $H(m)$  et vérifie l'égalité  $m'_h = H(m)$ . - Si l'égalité est vérifiée, B accepte la signature  $s$  de A sur le message M.

Le calcul de signature est **plus lent** que la vérification à cause de différence de taille entre l'exposant  $d$  ( $\text{taille}(d) \approx \text{taille}(\phi(n))$ ) et  $e$ .

Les risques et attaques mentionnés dans le procédé d'encryption s'appliquent également pour la signature.

Il convient de **différencier les paires de clés d'encryption et de signature** puisqu'elles nécessitent des politiques de stockage, sauvegarde et mise à jour distinctes.

### Révision rapide

**RSA signature :**  $s = m_R^d \bmod n$  (privée).

Vérif :  $m'_R = s^e \bmod n$  (publique).

Avec appendice :  $s = H(m)^d \bmod n$ .

Signature lente, vérif rapide.

## Signatures Aveugles (Blind Signatures)

**Principe :** A envoie une information à B pour signature. À partir de la réponse, A peut calculer la signature de B sur un message différent, que B n'a jamais vu.

**Exploitation de la propriété multiplicative RSA :**

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

**Fonctions de camouflage :**

Soit  $k$  un entier avec  $\text{pgcd}(n, k) = 1$  :

- **Blinding** :  $f(m) = m \cdot k^e \bmod n$
- **Unblinding** :  $g(m) = k^{-1} \cdot m \bmod n$

Résultat :  $g(S_B(f(m))) = g(S_B(mk^e \bmod n)) = g(m^d k \bmod n) = m^d \bmod n = S_B(m)$

**Protocole :**

1. A → B :  $m' = f(m)$  (message camouflé)
2. A ← B :  $s' = S_B(m')$  (signature du message camouflé)
3. A calcule  $g(s')$  et obtient  $S_B(m)$  (signature du message original)

**Applications :** Argent électronique anonyme, systèmes de vote électronique.

## Texte original

### Schéma inventé par Chaum.

**Idée :** A envoie une information à B pour signature. B retourne à A l'information signée. A partir de cette signature, A peut calculer la signature de B sur un autre message choisi à priori par A. Ceci permet à A d'avoir une signature de B sur un message que B n'a jamais vu (d'où le nom de signature aveugle...).

En fait il s'agit d'une faille basée sur la **propriété multiplicative de RSA** :  $(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod{n}$  qui a été exploitée pour en faire un nouveau procédé de signature.

**Algorithme :** Soit  $S_B$  la signature de RSA de B avec  $(n, e)$  et  $d$ , resp. les clés publiques et privées de B. Soit  $k$  un entier fixé avec  $\text{pgcd}(n, k) = 1$  :

$f : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$  avec  $f(m) = m \cdot k^e \pmod{n}$  (blinding function)

$g : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$  avec  $g(m) = k^{-1} \cdot m \pmod{n}$  (unblinding function)

ce qui donne :

$$g(S_B(f(m))) = g(S_B(mk^e \pmod{n})) = g(m^d k \pmod{n}) = m^d \pmod{n} = S_B(m)$$

### Protocole :

- A → B :  $m' = f(m)$
- A ← B :  $s' = S_B(m')$
- A calcule  $g(s')$  et obtient la signature souhaitée en utilisant (\*).

## Révision rapide

**Blind signature :** Exploite multiplicativité RSA.

Camouflage  $f(m) = m \cdot k^e$

Décamouflage  $g(m) = k^{-1} \cdot m$ .

B signe  $f(m)$ , A obtient  $S_B(m)$  sans que B voie  $m$ .

## Signature Rabin

### Génération des clés :

- Générer deux grands nombres premiers  $p$  et  $q$
- Calculer  $n = pq$
- **Clé publique** :  $n$  ; **Clé privée** :  $(p, q)$

### Signature :

1. Calculer  $m_R = R(m)$  (fonction de redondance)
2. Calculer  $s = \sqrt{m_R} \pmod{n}$  (racine carrée mod  $n$ )

3. Envoyer  $s$  (une des 4 racines carrées)

**Vérification :**

1. Calculer  $m'_R = s^2 \bmod n$
2. Vérifier  $m'_R \in M_R$  (rejeter sinon)
3. Reconstituer  $m = R^{-1}(m'_R)$

**Caractéristiques :**

- Basé sur le problème SQROOTP (racines carrées mod composite)
- **Provably secure** : équivalent à la factorisation
- Vulnérable aux attaques chosen-ciphertext actives

**Texte original**

**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  $(p, q)$ .

**Signature :**

- A calcule la fonction de redondance du message  $m$  :  $m_R := R(m)$ .
- A utilise sa clé privée pour calculer la signature :  $s := m_R^{1/2} \bmod n$  en utilisant des algorithmes efficaces pour calculer des racines carrées mod  $p$  et mod  $q$ .
- A envoie  $s$  à B ( $s$  est une des 4 racines carrées obtenues).

**Vérification :**

- L'entité B obtient  $n$ , la clé publique authentique de A.
- B calcule  $m'_R = s^2 \bmod n$ , vérifie  $m'_R \in M_R$  et rejette la signature si  $m'_R \notin M_R$
- B retrouve le message correctement signé par A en calculant :  $m = R^{-1}(m'_R)$ .

**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 (on demande à A de déchiffrer un ciphertext choisi) 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  $m \not\equiv m_x \pmod{n}$  (probabilité 0.5), M recommence avec un nouveau  $m$ .
- Sinon, A calcule  $\text{pgcd}(m - m_x, n)$  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 jettterait les autres solutions n'ayant pas le niveau de redondance préétabli.

### Révision rapide

**Rabin :**  $s = \sqrt{m_R} \bmod n$ .

Vérif :  $m'_R = s^2 \bmod n$ .

**Provably secure** (équivalent factorisation).

Vulnérable attaques actives chosen-ciphertext.

## Signature ElGamal

### Génération des clés :

- Générer premier  $p$  et générateur  $\alpha \in \mathbb{Z}_p^*$
- Générer secret  $a$  aléatoire, calculer  $y = \alpha^a \bmod p$
- **Clé publique** :  $(p, \alpha, y)$  ; **Clé privée** :  $a$

### Signature :

1. Calculer  $m_h = H(m)$
2. Générer  $k$  aléatoire avec  $\text{pgcd}(k, p - 1) = 1$
3. Calculer  $r = \alpha^k \bmod p$
4. Calculer  $s = k^{-1}(m_h - ar) \bmod (p - 1)$
5. Signature :  $(r, s)$

### Vérification :

1. Vérifier  $1 \leq r \leq p - 2$  (rejeter sinon)
2. Calculer  $v_1 = y^r r^s \bmod p$
3. Calculer  $v_2 = \alpha^{H(m)} \bmod p$
4. Accepter si  $v_1 = v_2$

**Preuve :** Si  $s \equiv k^{-1}(m_h - ar) \pmod{p-1}$ , alors  $m_h \equiv ar + ks \pmod{p-1}$

Donc  $v_2 = \alpha^{m_h} \equiv \alpha^{ar+ks} \equiv (\alpha^a)^r(\alpha^k)^s \equiv y^r r^s = v_1 \pmod{p}$

**Caractéristiques :**

- Fonctionne uniquement avec appendice (hash)
- Base du DSA (Digital Signature Algorithm)
- $k$  doit être unique pour chaque signature

### Texte original

**Génération des clés :**

- Chaque entité (A) crée une paire de clés (publique et privée) comme suit :
- A génère un nombre premier  $p$  ( $\text{len}(p) \geq 1024$  bits) et un générateur  $\alpha$  de  $\mathbb{Z}_p^*$ .
- A génère un nombre aléatoire  $a$ , tel que  $1 \leq a \leq p-2$  et calcule  $y := \alpha^a \pmod{p}$ .
- La clé publique de A est  $(p, \alpha, y)$ , la clé privée de A est  $a$ .

**Signature :**

- A utilise une fonction de hachage  $H$  et calcule  $m_h := H(m)$ .
- A génère un nombre aléatoire  $k$  ( $1 \leq k \leq p-2$  et  $\text{pgcd}(k, p-1) = 1$ ) et calcule  $k^{-1} \pmod{p-1}$
- A calcule  $r := \alpha^k \pmod{p}$  et ensuite  $s := k^{-1}(m_h - ar) \pmod{p-1}$
- La signature de A sur le message  $m$  est le couple  $(r, s)$ .

**Vérification :**

- L'entité B obtient  $(p, \alpha, \alpha^a \pmod{p})$ , la clé publique authentique de A.
- B vérifie que  $1 \leq r \leq p-2$ , sinon rejette la signature.
- B calcule  $v_1 := y^r r^s \pmod{p}$
- B calcule  $H(m)$  et  $v_2 := \alpha^{H(m)} \pmod{p}$
- B accepte la signature ssi.  $v_1 = v_2$ .

**Remarques :**

**Preuve que le schéma fonctionne :** Si  $s \equiv k^{-1}(m_h - ar) \pmod{p-1}$ , on a que :

$$m_h \equiv (ar + ks) \pmod{p-1}$$

et

$$v_2 = \alpha^{H(m)} \pmod{p}$$

si, comme on souhaite montrer  $m_h = H(m)$ , en réduisant les exposants mod  $(p-1)$ , on peut réécrire  $v_2$  :

$$v_2 \equiv \alpha^{ar+ks} \pmod{p}$$

D'autre part :

$$v_1 = y^r r^s \equiv \alpha^{ar} \alpha^{ks} \equiv \alpha^{ar+ks} \pmod{p}$$

c.q.f.d.

Par construction, le schéma d'ElGamal fonctionne **uniquement avec appendice** (résultat de l'application d'une fonction de hachage). Le schéma de Nyberg-Rueppel introduit une variation permettant la reconstitution du message.

Le **Digital Signature Algorithm (DSA)**, approuvé par le US National Institute of Standards and Technology est devenu le standard de signature le plus couramment utilisé. Il est construit sur la base d'un dérivé direct du schéma d'ElGamal avec la fonction de hachage SHA-1.

### Révision rapide

**ElGamal :**  $(r, s)$  avec  $r = \alpha^k \pmod{p}$ ,  $s = k^{-1}(m_h - ar) \pmod{p-1}$ .

Vérif :  $y^r r^s \stackrel{?}{=} \alpha^{H(m)} \pmod{p}$ .

Base de DSA.

$k$  unique crucial.

## Signatures et Crypto-monnaies

Les crypto-monnaies utilisent massivement les signatures digitales pour authentifier les transactions.

### Bitcoin et Ethereum :

- Utilisent **ECDSA** (Elliptic Curve Digital Signature Algorithm)
- Dérivé d'ElGamal sur courbes elliptiques
- Sécurité basée sur ECDLP

### Processus de transaction :

Chaque dépense/transmission nécessite :

- Signature avec la **clé privée du détenteur** actuel
- Le détenteur était le destinataire de la transaction précédente
- Chaque transaction forme une chaîne d'authentification

### Avantages ECDSA :

- Clés plus courtes pour sécurité équivalente
- Calculs plus efficaces

- Adapté aux contraintes des blockchains

#### Texte original

La plupart des crypto-monnaies se basent sur la cryptographie asymétrique. Le bitcoin p.ex. utilise des signatures digitales pour authentifier ses transactions.

La dépense ou la transmission de bitcoins nécessite la signature avec la clé privée du détenteur (qui était à son tour le destinataire de la transaction précédente).

Bitcoin et Ethereum utilisent l'algorithme **ECDSA (Elliptic Curve Digital Signature Algorithm)** dérivé de l'algorithme de signature de ElGamal sur les courbes elliptiques dont la sécurité repose sur ECDLP.

[Image : Schéma montrant la chaîne de transactions Bitcoin avec signatures]

Source Image : Bitcoin: A Peer-to-Peer Electronic Cash System. Satoshi Nakamoto

#### Révision rapide

**Crypto-monnaies :** Bitcoin/Ethereum utilisent **ECDSA** (ElGamal sur courbes elliptiques).

Chaque transaction signée avec clé privée détenteur.

Sécurité basée ECDLP.

### Tableau Récapitulatif des Schémas

Classe	Schéma	Message Recovery	Problème de base
<b>Signatures Classiques</b>	RSA	Oui	RSAP
	Rabin	Oui	SQROOTP
	ElGamal	Non	DLP
	DSS	Non	DLP
<b>One-time Signatures</b>	Lamport	Non	dépend de la OWF
	Bos-Chaum	Non	dépend de la OWF
<b>Undeniable Signatures</b>	Chaum-van Antwerpen	Non	DLP
<b>Fail-Stop Signatures</b>	van Heyst-Pedersen	Non	DLP
<b>Blind Signatures</b>	Chaum	Oui	RSAP

#### Texte original

[Tableau complet avec toutes les informations ci-dessus]

Le fonctionnement des procédés de signature One-time, Undeniable et Fail-Stop peut être consulté dans [Men97].

## Révision rapide

Signatures classiques :

- RSA/Rabin (recovery)
- ElGamal/DSS (appendice)

Spécialisées :

- One-time
- Undeniable
- Fail-Stop
- Blind

Problèmes base : RSAP, SQROOTP, DLP, dépend de la OWF.

## Types d'Attaques

Critères pour “casser” un schéma :

- **Total Break** : Calculer la clé privée ou algorithme efficace de génération
- **Falsification sélective** : Générer signature pour message/classe fixé(e)
- **Falsification existentielle** : Forger au moins une signature (sans contrôle du message)

Attaques de base :

- **Key-only** : Seule la clé publique est connue
- **Known-messages** : Accès à signatures de messages connus
- **Chosen-messages** : Attaquant choisit messages à signer
- **Adaptive chosen-messages** : Choix dépend des réponses précédentes

Ces attaques sont équivalentes aux attaques sur systèmes d'encryption (known/chosen-plaintext/ciphertext) mais appliquées aux messages.

## Texte original

Critères pour “casser” un schéma de signature digitale :

- **Total Break** : Calculer la clé privée du signataire ou un algorithme efficace (polynomial) pour générer des signatures.
- **Falsification sélective (selective forgery)** : L'adversaire est capable de générer une signature valide pour un message (ou une classe de messages) fixé.
- **Falsification existentielle (existential forgery)** : L'adversaire est capable de

forger une signature pour (au moins) un message (dont il n'a pas le contrôle).

#### Attaques de base :

- **Attaques key-only** : L'adversaire a seulement connaissance de la clé publique du signataire.
- **Attaques basées sur les messages** : L'adversaire a accès à des signatures correspondantes à des :
  - known-messages
  - chosen-messages
  - adaptive chosen-messages

Equivalents à des attaques x-ciphertext mais avec des messages !

#### Révision rapide

##### Casser signature :

- Total break (clé privée)
- falsification sélective (message fixé)
- existentielle (un message)

##### Attaques :

- key-only
- known/chosen/adaptive-chosen-messages.