CIA - La Preuve d'Origine

Signatures Numériques et Mises à Jour Sécurisées

Loïc Rouquette
EPITA // IF
2 octobre 2025

CIA - La Preuve d'Origine © 2025 Loïc Rouquette is licensed under CC BY 4.0.
To view a copy of this license, visit https ://creativecommons.org/licenses/by/4.0/.

Table des matières

Contexte et Enjeux

Plongée dans ECDSA

La Rupture Quantique

Construction des standards PQC

Secure Over-the-Air (OTA) Updates

licensed under CC BY 4.0

1

Contexte et Enjeux

Objectifs & Prérequis

Objectifs

- Formaliser le modèle de sécurité EUF-CMA (Existential Unforgeability under Chosen Message Attack)
- Maîtriser la construction d'ECDSA sur les groupes cycliques de courbes elliptiques.
- Analyser les vecteurs d'attaques sur les implémentations d'ECDSA (Nonce, Side-Channel, Fautes).
- Comprendre le principe de l'algorithme de Shor et son impact sur la cryptographie à clé publique.
- Décortiquer la construction des schémas de signature post-quantiques standardisés par le NIST

Prérequis

- Théorie des groupes, corps finis ($GF(p)$, $GF(2^m)$)
- Principes de la cryptographie asymétrique (RSA, Diffie-Hellman)
- Propriétés des fonctions de hachage (CR, PR, SPR)

licensed under CC BY 4.0

Le Modèle de Sécurité Formel : EUF-CMA

Le jeu de sécurité standard pour les signatures.

Acteurs

Un Challengeur (honnête) et un Adversaire A.

1. **Setup :** Le Challenger génère une paire de clés pk, sk et donne pk à l'Adversaire.
2. **Oracle d'accès :** L'Adversaire peut demander au Challenger de signer un nombre polynomial de message de son choix $(m_1, m_2, ..., m_q)$. Le Challenger retourne les signatures valides $(\sigma_1, \sigma_2, ..., \sigma_q)$.
3. **Forge :** L'Adversaire produit une nouvelle paire m^*, σ^*.

Conditions de victoire

L'Adversaire gagne si :

- $\texttt{Verify}(pk, m^*, \sigma^*)$ retourne `True`.
- m^* n'a jamais été soumis à l'Oracle de signatures.

Un schéma est **EUF-CMA** si la probabilité de victoire de tout adversaire A en temps polynomial est **négligeable**.

licensed under CC BY 4.0

L'Écosystème de Confiance : Au-delà de l'Algo

Les Composants essentiels (rappel)

- **PKI (Public Key Infrastructure) :** L'architecture de confiance
- **Certificats X.509 :** La liaison cryptographique entre une entité et une clé publique, signée par une Autorité de Certification (AC).
- **Chaîne de Confiance :** Validation récursive des certificats jusqu'à un certificat racine (Root CA).

Mécanismes de Révocation

- **CRL (Certificate Revocation List) :** Listes de diffusion périodiques. Problèmes de fraîcheur et de volume.
- **OCSP (Online Certificate Status Protocol) :** Interrogation en temps réel. Problèmes de centralisation et de vie privée.
- **OCSP Stapling :** Le serveur web "agrafe" une réponse OCSP signée à son certificat, améliorant performance et confidentialité.

licensed under CC BY 4.0

2

Plongée dans ECDSA

Pourquoi ECDSA ? Rappel des Avantages

- **Sécurité par bit de clé :** Nettement supérieur à RSA/DSA.
- **Complexité du problème sous-jacent :** Le problème du Logarithme Discret sur les Courbes Elliptiques (ECDLP) est (classiquement) plus ”difficile” que la factorisation ou le DLP sur F_p^* (groupe multiplicatif modulo p).
  - Meilleur algorithme classique connue (Pollard's ρ) : complexité en $\mathcal{O}(\sqrt{n})$, où n est l'ordre du groupe.
  - Pour RSA/DSA (Number Field Sieve) : complexité sub-exponentielle.
- **Conséquences :** Clés plus courtes, signatures plus petites, calculs plus rapides (surtout la vérification).

licensed under CC BY 4.0

Structure de Groupe sur les Courbes Elliptiques (Rappels)

Définition Formelle

Soit E une courbe définie sur un corps fini F_q, définie par $y^2 = x^3 + ax + b$.

L'ensemble des points de E sur F_q, noté $E(F_q)$, forme un groupe abélien fini avec :

- Loi de composition interne : L'addition de points $(+)$.
- Élément neutre : Le point à l'infini, $\mathcal{O}$.
- Inverse : Pour un point $P = (x, y)$, son inverse $-P$ est $(x, -y)$.

Pratique Cryptographique

On ne travaille pas sur $E(F_q)$ entier, mais sur un **sous-groupe cyclique** d'ordre n (un grand nombre entier), généré par une point de base G.

licensed under CC BY 4.0

Paramètres de Domaine d'une Courbe

Un standard de courbe (*e.g.* : **`secp256r1`** alias NIST P-256) spécifie un sextuplet (p, a, b, G, n, h) :

- p : Le module premier du corps F_p.
- a, b : Les coefficients de l'équation de la courbe.
- G : Le point de base (G_x, G_y) qui génère le sous-groupe.
- n : L'ordre du sous-groupe (un grand premier).
- h : Le cofacteur, $h = \#E(F_p)/n$. Idéalement $h = 1$.

licensed under CC BY 4.0

Génération des Clés ECDSA

Clé privée (secrète) d

Un entier choisi uniformément au hasard dans $[1, n-1]$.

Clé publique Q

- Le calcul $d \times G$ est efficace (*e.g.* : algorithme *Double and Add*).
- Retrouver d à partir de Q et G est le problème ECDLP.

licensed under CC BY 4.0

Algorithme de Signature ECDSA : La Dérivation

Objectif : Produire une signature (r, s) qui prouve la connaissance de d sans le révéler.

licensed under CC BY 4.0

Algorithme de Signature ECDSA : La Dérivation

Objectif : Produire une signature (r, s) qui prouve la connaissance de d sans le révéler.

Le schéma est une variante du schéma de signature d'ElGamal.

licensed under CC BY 4.0

Algorithme de Signature ECDSA : La Dérivation

Objectif : Produire une signature (r, s) qui prouve la connaissance de d sans le révéler.

Le schéma est une variante du schéma de signature d'ElGamal.

1. Calculer $h = H(m)$ (tronqué à la taille de n).

licensed under CC BY 4.0

Algorithme de Signature ECDSA : La Dérivation

Objectif : Produire une signature (r, s) qui prouve la connaissance de d sans le révéler.

Le schéma est une variante du schéma de signature d'ElGamal.

1. Calculer $h = H(m)$ (tronqué à la taille de n).
2. Générer un **nonce éphémère ET secret** k dans $[1, n-1]$.

licensed under CC BY 4.0

Algorithme de Signature ECDSA : La Dérivation

Objectif : Produire une signature (r, s) qui prouve la connaissance de d sans le révéler.

Le schéma est une variante du schéma de signature d'ElGamal.

1. Calculer $h = H(m)$ (tronqué à la taille de n).
2. Générer un **nonce éphémère ET secret** k dans $[1, n-1]$.
3. "Commitment" : Calculer le point $k \times G = (x_1, y_1)$

licensed under CC BY 4.0

Algorithme de Signature ECDSA : La Dérivation

Objectif : Produire une signature (r, s) qui prouve la connaissance de d sans le révéler.

Le schéma est une variante du schéma de signature d'ElGamal.

1. Calculer $h = H(m)$ (tronqué à la taille de n).
2. Générer un **nonce éphémère ET secret** k dans $[1, n-1]$.
3. "Commitment" : Calculer le point $k \times G = (x_1, y_1)$
4. Convertir le point en entier : $r = x_1 \mod n$. r est la première partie de la signature, il "lie" le nonce k à la signature.

licensed under CC BY 4.0

Algorithme de Signature ECDSA : La Dérivation

Objectif : Produire une signature (r, s) qui prouve la connaissance de d sans le révéler.

Le schéma est une variante du schéma de signature d'ElGamal.

1. Calculer $h = H(m)$ (tronqué à la taille de n).
2. Générer un **nonce éphémère ET secret** k dans $[1, n-1]$.
3. "Commitment" : Calculer le point $k \times G = (x_1, y_1)$
4. Convertir le point en entier : $r = x_1 \mod n$. r est la première partie de la signature, il "lie" le nonce k à la signature.
5. "Challenge/Response" : Calculer $s = k^{-1} \times (h + r \times d) \mod n$. s lie le message h, la clé secrète d et le nonce k.

licensed under CC BY 4.0

Algorithme de Vérification : La Preuve Mathématique

Le vérificateur connaît $(m, (r, s))$ et la clé publique Q.

licensed under CC BY 4.0

Algorithme de Vérification : La Preuve Mathématique

Le vérificateur connaît $(m, (r, s))$ et la clé publique Q.

Idée

Reconstruire le point $k \times G$ en utilisant uniquement les informations publique et vérifier que sa coordonnée x correspond à r.

licensed under CC BY 4.0

Algorithme de Vérification : La Preuve Mathématique

Le vérificateur connaît $(m, (r, s))$ et la clé publique Q.

Idée

Reconstruire le point $k \times G$ en utilisant uniquement les informations publique et vérifier que sa coordonnée x correspond à r.

1. À partir de l'équation de s, on peut isoler k :

$$s \times k \equiv h + r \times d \mod n$$
$$k \equiv s^{-1} \times (h + r) \times d \mod n$$

licensed under CC BY 4.0

Algorithme de Vérification : La Preuve Mathématique

Le vérificateur connaît $(m, (r, s))$ et la clé publique Q.

Idée

Reconstruire le point $k \times G$ en utilisant uniquement les informations publique et vérifier que sa coordonnée x correspond à r.

1. À partir de l'équation de s, on peut isoler k :

$$s \times k \equiv h + r \times d \mod n$$
$$k \equiv s^{-1} \times (h + r) \times d \mod n$$

2. Multiplions par G :

$$k \times G = ((s^{-1})(h + r \times d)) \times G$$
$$k \times G = (s^{-1}h) \times G + (s^{-1}r \times d) \times G$$
$$k \times G = (s^{-1}h) \times G + (s^{-1}r) \times (d \times G)$$

licensed under CC BY 4.0

Algorithme de Vérification : La Preuve Mathématique

Le vérificateur connaît $(m, (r, s))$ et la clé publique Q.

Idée

Reconstruire le point $k \times G$ en utilisant uniquement les informations publique et vérifier que sa coordonnée x correspond à r.

1. À partir de l'équation de s, on peut isoler k :

$$s \times k \equiv h + r \times d \mod n$$
$$k \equiv s^{-1} \times (h + r) \times d \mod n$$

2. Multiplions par G :

$$k \times G = ((s^{-1})(h + r \times d)) \times G$$
$$k \times G = (s^{-1}h) \times G + (s^{-1}r \times d) \times G$$
$$k \times G = (s^{-1}h) \times G + (s^{-1}r) \times (d \times G)$$

3. En posant $w = s^{-1}$, $u_1 = h \times w$ et $u_2 = r \times w$ et en se rappelant que $Q = d \times G$: $k \times G = u_1 \times G + u_2 \times Q$.

Algorithme de Vérification : La Preuve Mathématique

Le vérificateur connaît $(m, (r, s))$ et la clé publique Q.

Idée

Reconstruire le point $k \times G$ en utilisant uniquement les informations publique et vérifier que sa coordonnée x correspond à r.

1. À partir de l'équation de s, on peut isoler k :

$$s \times k \equiv h + r \times d \mod n$$
$$k \equiv s^{-1} \times (h + r) \times d \mod n$$

2. Multiplions par G :

$$k \times G = ((s^{-1})(h + r \times d)) \times G$$
$$k \times G = (s^{-1}h) \times G + (s^{-1}r \times d) \times G$$
$$k \times G = (s^{-1}h) \times G + (s^{-1}r) \times (d \times G)$$

3. En posant $w = s^{-1}$, $u_1 = h \times w$ et $u_2 = r \times w$ et en se rappelant que $Q = d \times G$: $k \times G = u_1 \times G + u_2 \times Q$.

Test de vérification

Le vérificateur calcule u_1, u_2, puis le point $P = u_1 \times G + u_2 \times Q$. Si $P \neq \mathcal{O}$, il vérifie que la coordonnée x de P modulo n est égale à r.

licensed under CC BY 4.0

Le Point Critique : la Génération du Nonce k

- **La règle d'Or :** k doit être

licensed under CC BY 4.0

Le Point Critique : la Génération du Nonce k

- **La règle d'Or :** k doit être **unique**,

licensed under CC BY 4.0

Le Point Critique : la Génération du Nonce k

- **La règle d'Or :** k doit être **unique**, **imprévisible** et

licensed under CC BY 4.0

Le Point Critique : la Génération du Nonce k

- **La règle d'Or :** k doit être **unique**, **imprévisible** et **secret**.

licensed under CC BY 4.0

Le Point Critique : la Génération du Nonce k

- **La règle d'Or :** k doit être **unique**, **imprévisible** et **secret**.
- **Réutilisation de k (rappels) :** Permet de calculer la clé d trivialement.

licensed under CC BY 4.0

Le Point Critique : la Génération du Nonce k

- **La règle d'Or :** k doit être **unique**, **imprévisible** et **secret**.
- **Réutilisation de k (rappels) :** Permet de calculer la clé d trivialement.
- **Nonce biaisé :** Si k n'est pas tiré uniformément aléatoire, des attaques statistiques sont possibles.
  - *e.g.* Si les premiers bits de k sont prévisibles sur plusieurs signatures.
  - Attaque par réseaux : cette attaque a touché Bitcoin en 2013 [3, 2].

licensed under CC BY 4.0

Solution : Les Nonces Déterministes (RFC 6979)

- **Problème :** faire confiance à un générateur de nombres aléatoires (RNG) est un point de défaillance majeur.
- **Solution RFC 6979 :** Générer k de manière **déterministe** à partir de la clé privée d et du hash du message h.
- **Mécanisme :** Utilise HMAC avec d comme clé privée pour dériver k un nombre pseudo-aléatoire à partir de h.

$$h = \texttt{HMAC}(d, h) \text{ (simplifié).}$$

- **Avantages :**
  - Élimine la dépendance à un RNG externe pour la signature.
  - Si d et h sont les mêmes, k sera le même. Ce qui n'est pas un problème car on ne signe jamais deux fois exactement le même message (sauf erreur protocolaire). Si on signe deux messages différents, h sera différent, donc k sera différent.
  - Immunise contre les attaques par fautes sur le RNG.

licensed under CC BY 4.0

Les autres vecteurs d'attaque sur ECDSA

Attaques par canaux auxilaires

- **Attaque par temps (timing Attacks) :** Si la multiplication scalaire (d*G) n'est pas implémentée en temps constant, un attaquant peut déduire des bits de d en mesurant le temps de signature.

licensed under CC BY 4.0

Les autres vecteurs d'attaque sur ECDSA

Attaques par canaux auxilaires

- **Attaque par temps (timing Attacks) :** Si la multiplication scalaire (d*G) n'est pas implémentée en temps constant, un attaquant peut déduire des bits de d en mesurant le temps de signature.
- **Analyse de consommation (Power Analysis - SPA/DPA) :** L'attaquant mesure la consommation électrique de l'appareil (ex : carte à puce) pendant la signature. Les opérations "Double" et "Add" peuvent avoir des signatures énergétiques différentes, révélant la séquence des bits de d.

licensed under CC BY 4.0

Les autres vecteurs d'attaque sur ECDSA

Attaques par canaux auxilaires

- **Attaque par temps (timing Attacks) :** Si la multiplication scalaire (d*G) n'est pas implémentée en temps constant, un attaquant peut déduire des bits de d en mesurant le temps de signature.
- **Analyse de consommation (Power Analysis - SPA/DPA) :** L'attaquant mesure la consommation électrique de l'appareil (ex : carte à puce) pendant la signature. Les opérations "Double" et "Add" peuvent avoir des signatures énergétiques différentes, révélant la séquence des bits de d.

Contre-mesures :

Algorithmes en temps constant (Montgomery Ladder), "blinding" de la clé privée.

licensed under CC BY 4.0

Attaques par Injection de Fautes (Fault Attacks)

Principe

Induire une erreur de calcul (via laser, glitch de tension...) pendant la signature et analyser la signature erronée.

Bellcore Attack [1]

1. L'attaquant obtient une signature correcte r, s pour le message m.
2. Il demande une autre signature de m, mais induit une faute pendant le calcul de $k \times G$. Le résultat erroné est x'_1, y'_1.
3. La signature erronée r', s' est calculée avec le même k et d, mais r' est différent.
4. En combinant les deux équations (la correcte et l'erronée), l'attaquant peut isoler et calculer la clé privée d.

Contre-mesures :

Vérifier le resultat de chaque signature avant de la publier.

licensed under CC BY 4.0

Récapitulatif des vecteurs d'attaques théoriques

- **Le Nonce k : Le Talon d'Achille**
  - **Réutilisation :** Catastrophique, mène à une divulgation complète de la clé privée.
  - **Biais / Prédictabilité :** Permet des attaques cryptanalytiques avancées (via réseaux euclidiens) pour retrouver la clé privée.
- **Les Canaux auxiliaires (Side-Channels)**
  - L'implémentation "fuit" de l'information sur le secret qu'elle manipule.
  - **Timing, Consommation Énergétique (DPA), Émissions électromaniétiques (EMA).**
- **Les Injections de Fautes**
  - Altérer le calcul pour produire une signature erronée qui, comparée à une signature correcte, révèle le secret.

licensed under CC BY 4.0

ECDSA un algorithme puissant : une implémentation délicate

Récapitulatif des vecteurs d'attaques théoriques

- **Le Nonce k : Le Talon d'Achille**
  - **Réutilisation :** Catastrophique, mène à une divulgation complète de la clé privée.
  - **Biais / Prédictabilité :** Permet des attaques cryptanalytiques avancées (via réseaux euclidiens) pour retrouver la clé privée.
- **Les Canaux auxiliaires (Side-Channels)**
  - L'implémentation "fuit" de l'information sur le secret qu'elle manipule.
  - **Timing, Consommation Énergétique (DPA), Émissions électromaniétiques (EMA).**
- **Les Injections de Fautes**
  - Altérer le calcul pour produire une signature erronée qui, comparée à une signature correcte, révèle le secret.

La question centrale

La sécurité d'ECDSA dépend-elle plus de la qualité de son implémentation que de la difficulté de l'ECDLP ?

licensed under CC BY 4.0

Étude de Cas n°1 - La faille des portefeuilles Bitcoin sur Android (2013)

Le Contexte

En août 2013, des Bitcoins ont été volés de portefeuilles générés sur des applications Android.

licensed under CC BY 4.0

Étude de Cas n°1 - La faille des portefeuilles Bitcoin sur Android (2013)

Le Contexte

En août 2013, des Bitcoins ont été volés de portefeuilles générés sur des applications Android.

La Cause Racine

Un bug dans la classe `java.security.SecureRandom` d'Android. Au lieu d'utiliser le générateur cryptographique du noyau Linux (`/dev/urandom`), ce qui a fait que le PRNG n'était pas correctement initialisé.

licensed under CC BY 4.0

L'Attaque

- Le générateur de nombres aléatoires produisait des valeurs de k qui n'étaient pas uniformément distribuées. Plusieurs signatures, effectuées depuis le même portefeuille (donc la même clé privée d), se sont retrouvées avec le même r.
- Rappel : $r = (k \times G)_x \mod n$. Si r est le même, cela implique que le k utilisé était le même (ou $k' = n - k$).
- Des attaquants ont scanné la blockchain Bitcoin, cherché des transactions avec des r répétés provenant de la même adresse publique, et ont appliqué l'attaque de réutilisation du nonce pour calculer les clés privées et vider les portefeuilles.

licensed under CC BY 4.0

Étude de Cas n°1 - La faille des portefeuilles Bitcoin sur Android (2013)

L'Attaque

- Le générateur de nombres aléatoires produisait des valeurs de k qui n'étaient pas uniformément distribuées. Plusieurs signatures, effectuées depuis le même portefeuille (donc la même clé privée d), se sont retrouvées avec le même r.
- Rappel : $r = (k \times G)_x \mod n$. Si r est le même, cela implique que le k utilisé était le même (ou $k' = n - k$).
- Des attaquants ont scanné la blockchain Bitcoin, cherché des transactions avec des r répétés provenant de la même adresse publique, et ont appliqué l'attaque de réutilisation du nonce pour calculer les clés privées et vider les portefeuilles.

Leçon

Une faille dans une bibliothèque système standard peut anéantir la sécurité d'un algorithme cryptographique robuste. **La chaîne de sécurité est aussi forte que son maillon le plus faible.**

licensed under CC BY 4.0

Étude de Cas n2 - Le "Jailbreak" de la Playstation 3 (2010)

Contexte

Sony utilisait ECDSA pour signer les exécutables et empêcher le code non autorisé de tourner sur la PS3.

licensed under CC BY 4.0

Étude de Cas n2 - Le "Jailbreak" de la Playstation 3 (2010)

Contexte

Sony utilisait ECDSA pour signer les exécutables et empêcher le code non autorisé de tourner sur la PS3.

La Cause Racine

Pour chaque signature, Sony a utilisé la même valeur de k. Ce n'était pas un accident ou un bug de RNG ; le nonce était une constante codée en dur dans le logiciel de signature.

licensed under CC BY 4.0

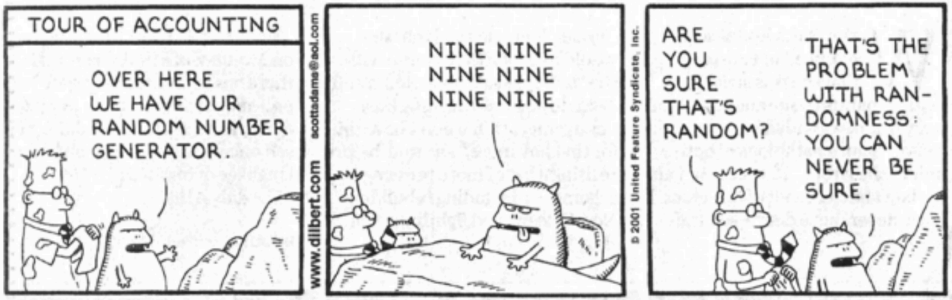


Figure 1 : Un bon générateur aléatoire selon Sony.

Étude de Cas n2 - Le "Jailbreak" de la Playstation 3 (2010)

L'Attaque

- Le groupe de hackers **`fail0verflow`** a extrait deux signatures différentes de deux logiciels signés par Sony.
- Ayant $(m_1, (r, s_1))et(m_2, (r, s_2))$, avec r et k identiques, ils ont pu appliquer l'algèbre simple de l'attaque par réutilisation du nonce.
- $s_1 - s_2 = k^{-1}(h_1 - h_2)$
- $k = (h_1 - h_2) * (s_1 - s_2)^{-1} \mod n$
- Une fois k connu, ils ont retrouvé la clé privée maîtresse de la console : $d = (s_1 \times k - h_1) \times r^{-1} \mod n$.

licensed under CC BY 4.0

Étude de Cas n2 - Le "Jailbreak" de la Playstation 3 (2010)

L'Attaque

- Le groupe de hackers **`fail0verflow`** a extrait deux signatures différentes de deux logiciels signés par Sony.
- Ayant $(m_1, (r, s_1))et(m_2, (r, s_2))$, avec r et k identiques, ils ont pu appliquer l'algèbre simple de l'attaque par réutilisation du nonce.
- $s_1 - s_2 = k^{-1}(h_1 - h_2)$
- $k = (h_1 - h_2) * (s_1 - s_2)^{-1} \mod n$
- Une fois k connu, ils ont retrouvé la clé privée maîtresse de la console : $d = (s_1 \times k - h_1) \times r^{-1} \mod n$.

Leçon

Une incompréhension fondamentale d'un paramètre cryptographique a suffi à compromettre la sécurité de toute une plateforme. C'est l'exemple le plus célèbre et le plus pur de cette vulnérabilité.

licensed under CC BY 4.0

Approche Haut Niveau : Bibliothèque `cryptography`

Règle de Sécurité Fondamentale

- Ne ré-implémentez **JAMAIS** la cryptographie vous-même.
- Utilisez uniquement des bibliothèques reconnues, auditées et maintenues.

Exemple de Code Python

```python
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec
# 1. Génération de la clé privée avec la courbe NIST P-256
private_key = ec.generate_private_key(ec.SECP256R1())
# 2. Dérivation de la clé publique
public_key = private_key.public_key()
# Les clés sont prêtes à être sérialisées (stockées/échangées)
```

Points Clés :

- L'objet `private_key` encapsule l'entier secret d et les paramètres de la courbe.
- La complexité de la sélection d'un entier d dans le bon intervalle est totalement abstraite pour le développeur.

licensed under CC BY 4.0

Exemple de signature

```python
# Message à signer (sous forme de bytes)
message = b"Ceci est un message tres important a signer."

# 3. Processus de signature
signature = private_key.sign(
    message,
    # Le padding/schéma de signature est spécifié par la courbe
    ec.ECDSA(hashes.SHA256()) # Utilisation de SHA-256 comme fonction de hachage
)

print(f"Signature (en hex): {signature.hex()}")
```

Ce qui se passe en coulisses (et que nous n'avons PAS à gérer) :

- Le message **`message`** est d'abord haché avec SHA-256.
- La bibliothèque génère un nonce k de manière **déterministe** en suivant la **RFC 6979**. Pas de dépendance au RNG du système !
- Elle effectue le calcul (r, s).
- La signature est retournée sous un format standard (DER encoding).

licensed under CC BY 4.0

Vérification d'une signature

```python
from cryptography.exceptions import InvalidSignature

# 4. Processus de vérification
# Le vérificateur possède la clé publique, le message original et la signature.

try:
    public_key.verify(
        signature,
        message,
        ec.ECDSA(hashes.SHA256())
    )
    print("Verification reussie : La signature est valide.")

except InvalidSignature:
    print("ECHEC de la verification : La signature est INVALIDE !")

# Essayons avec un message altéré
message_altere = b"Ceci est un message tres important a signer" # Oups, un point en moins
try:
    public_key.verify(signature, message_altere, ec.ECDSA(hashes.SHA256()))
except InvalidSignature:
    print("Verification sur message altere : La signature est bien INVALIDE.")
```


licensed under CC BY 4.0

Mauvaises Pratiques d'Implementation : Le "Hall of Shame"

1. **”Roll Your Own Crypto” :** Tenter de ré-implémenter l’algèbre des corps finis et l’addition de points. Vous introduirez des bugs et des failles de side-channel.
2. **Utiliser des courbes Exotiques / Non-standard :** À moins d’être un cryptographe de renommée mondiale, vous ne pouvez pas évaluer la sécurité d’une courbe. Utilisez les standards (NIST, Brainpool, Curve25519).
3. **”Oublier” la validation des paramètres :**
   - Ne pas vérifier qu’une clé publique reçue est bien un point sur la courbe attendue. Cela ouvre la porte aux **attaques par courbe invalide**.
   - Ne pas vérifier que r and s sont dans l’intervalle $[1, n-1]$.
4. **Mauvaise gestion des secrets :**
   - Clés privées en clair dans le code source, dans une base de données non chiffrée, etc.
   - Utiliser la même clé privée pour des usages différents (ex : signature et chiffrement).
5. **Tronquer les Signatures ou les Hashs :** Penser ”gagner de la place” en ne gardant que les 32 premiers octets d’une signature de 64 octets. Cela détruit les garanties de sécurité.

licensed under CC BY 4.0

Attaque sur les courbes invalides

Contexte

Pertinent surtout pour les protocoles d'échange de clés comme ECDH, mais le principe est important.

Le Scénario

- Un serveur a une clé privée d_S et une clé publique $Q_S = d_S \times G$.
- L'attaquant Alice envoie sa clé publique Q_A au serveur pour établir un secret partagé.
- **L'astuce :** Alice choisit Q_A pour qu'il ne soit PAS sur la courbe standard E, mais sur une autre courbe E' spécialement choisie, dont les points ont un **ordre très petit**.
- Le serveur (s'il est naïf) ne vérifie pas que Q_A appartient à E. Il calcule le secret partagé $S = d_S \times Q_A$.
- Comme Q_A est dans un groupe de petit ordre k, le secret partagé S ne pourra prendre qu'un petit nombre de valeurs possibles.
- En observant le comportement du serveur (via un side-channel, par exemple un message d'erreur TLS), Alice peut déduire quelle était la valeur de S.
- Connaître S lui donne des informations sur $d_S \mod k$. En répétant l'attaque avec différentes courbes, elle peut reconstruire d_S via le Théorème des Restes Chinois.

Leçon

Toujours valider les entrées externes ! Une implémentation robuste doit vérifier que tout point reçu est sur la courbe et dans le bon sous-groupe.

licensed under CC BY 4.0

ECDSA est-il un algorithme intrinsèquement ”fragile” ?

Arguments pour le Oui

licensed under CC BY 4.0

ECDSA est-il un algorithme intrinsèquement ”fragile” ?

Arguments pour le Oui

- La sécurité s'effondre de manière spectaculaire (clé privée compromise) sur une simple erreur d'implémentation (réutilisation de k).

licensed under CC BY 4.0

ECDSA est-il un algorithme intrinsèquement ”fragile” ?

Arguments pour le Oui

- La sécurité s'effondre de manière spectaculaire (clé privée compromise) sur une simple erreur d'implémentation (réutilisation de k).
- La sensibilité aux canaux auxiliaires est très élevée et requiert des contre-mesures complexes (temps constant, blinding...).

licensed under CC BY 4.0

ECDSA est-il un algorithme intrinsèquement ”fragile” ?

Arguments pour le Oui

- La sécurité s'effondre de manière spectaculaire (clé privée compromise) sur une simple erreur d'implémentation (réutilisation de k).
- La sensibilité aux canaux auxiliaires est très élevée et requiert des contre-mesures complexes (temps constant, blinding...).
- L'algèbre sous-jacente est complexe, augmentant la surface d'attaque pour des bugs subtils (cf. attaques par courbe invalide).

licensed under CC BY 4.0

ECDSA est-il un algorithme intrinsèquement "fragile" ?

Arguments pour le Oui

- La sécurité s'effondre de manière spectaculaire (clé privée compromise) sur une simple erreur d'implémentation (réutilisation de k).
- La sensibilité aux canaux auxiliaires est très élevée et requiert des contre-mesures complexes (temps constant, blinding...).
- L'algèbre sous-jacente est complexe, augmentant la surface d'attaque pour des bugs subtils (cf. attaques par courbe invalide).

Arguments pour le Non

- La signature déterministe (RFC 6979) résout le problème le plus critique (k).

licensed under CC BY 4.0

ECDSA est-il un algorithme intrinsèquement "fragile" ?

Arguments pour le Oui

- La sécurité s'effondre de manière spectaculaire (clé privée compromise) sur une simple erreur d'implémentation (réutilisation de k).
- La sensibilité aux canaux auxiliaires est très élevée et requiert des contre-mesures complexes (temps constant, blinding...).
- L'algèbre sous-jacente est complexe, augmentant la surface d'attaque pour des bugs subtils (cf. attaques par courbe invalide).

Arguments pour le Non

- La signature déterministe (RFC 6979) résout le problème le plus critique (k).
- Les bibliothèques modernes et les implémentations matérielles intègrent les contre-mesures side-channel.

licensed under CC BY 4.0

ECDSA est-il un algorithme intrinsèquement "fragile" ?

Arguments pour le Oui

- La sécurité s'effondre de manière spectaculaire (clé privée compromise) sur une simple erreur d'implémentation (réutilisation de k).
- La sensibilité aux canaux auxiliaires est très élevée et requiert des contre-mesures complexes (temps constant, blinding...).
- L'algèbre sous-jacente est complexe, augmentant la surface d'attaque pour des bugs subtils (cf. attaques par courbe invalide).

Arguments pour le Non

- La signature déterministe (RFC 6979) résout le problème le plus critique (k).
- Les bibliothèques modernes et les implémentations matérielles intègrent les contre-mesures side-channel.
- RSA a aussi ses propres pièges d'implémentation (*e.g.* : attaques de Bleichenbacher sur le padding PKCS#1 v1.5). La fragilité n'est pas exclusive à ECDSA.

licensed under CC BY 4.0

D'où viennent les Courbes et peut-on leur faire Confiance ?

licensed under CC BY 4.0

D'où viennent les Courbes et peut-on leur faire Confiance ?

Les Courbes NIST (P-256, P-384, P-521)

- Standardisées par le National Institute of Standards and Technology (USA).

licensed under CC BY 4.0

D'où viennent les Courbes et peut-on leur faire Confiance ?

Les Courbes NIST (P-256, P-384, P-521)

- Standardisées par le National Institute of Standards and Technology (USA).
- Largement les plus utilisées dans l'industrie (TLS, certificats web...).

licensed under CC BY 4.0

D'où viennent les Courbes et peut-on leur faire Confiance ?

Les Courbes NIST (P-256, P-384, P-521)

- Standardisées par le National Institute of Standards and Technology (USA).
- Largement les plus utilisées dans l'industrie (TLS, certificats web...).
- **La Controverse :** Les constantes (a, b, G) qui définissent ces courbes ont été choisies par la NSA. Leurs origines sont opaques. Bien qu'il n'y ait aucune preuve de porte dérobée (backdoor), le manque de transparence a généré de la méfiance dans la communauté crypto, surtout après les révélations de Snowden.

licensed under CC BY 4.0

D'où viennent les Courbes et peut-on leur faire Confiance ?

Les Courbes NIST (P-256, P-384, P-521)

- Standardisées par le National Institute of Standards and Technology (USA).
- Largement les plus utilisées dans l'industrie (TLS, certificats web...).
- **La Controverse :** Les constantes (a, b, G) qui définissent ces courbes ont été choisies par la NSA. Leurs origines sont opaques. Bien qu'il n'y ait aucune preuve de porte dérobée (backdoor), le manque de transparence a généré de la méfiance dans la communauté crypto, surtout après les révélations de Snowden.

Les Alternatives

- **Courbes Brainpool :** Standardisées en Allemagne, générées de manière plus transparente.
- **Curve25519 (et Ed25519 pour la signature) :** Conçue par Daniel J. Bernstein. Tous les paramètres sont dérivés d'une seule graine de manière totalement transparente et publique. Elle est également conçue pour être résistante à de nombreuses attaques d'implémentation par défaut.

licensed under CC BY 4.0

Questions ??

3

La Rupture Quantique

Qubit

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{où} \quad \alpha, \beta \in \mathbb{C} \quad \text{et} \quad |\alpha|^2 + |\beta|^2 = 1$$

licensed under CC BY 4.0

Qubit

$$\underbrace{|\psi\rangle} = \alpha|0\rangle + \beta|1\rangle \quad \text{où} \quad \alpha, \beta \in \mathbb{C} \quad \text{et} \quad |\alpha|^2 + |\beta|^2 = 1$$

Notation de Dirac : état quantique du qubit.

licensed under CC BY 4.0

Qubit

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{où} \quad \alpha, \beta \in \mathbb{C} \quad \text{et} \quad |\alpha|^2 + |\beta|^2 = 1$$

$|0\rangle$ et $|1\rangle$ sont les états de base,
analogues aux valeurs 0 et 1 d'un bit classique.

licensed under CC BY 4.0

Qubit

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{où} \quad \alpha, \beta \in \mathbb{C} \quad \text{et} \quad |\alpha|^2 + |\beta|^2 = 1$$

α et β sont des nombres complexes appelés amplitudes de probabilités.

licensed under CC BY 4.0

Qubit

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{où} \quad \alpha, \beta \in \mathbb{C} \quad \text{et} \quad \underbrace{|\alpha|^2 + |\beta|^2 = 1}$$

Cette partie est **cruciale**. Elle s'appelle condition de normalisation. signifie simplement que la somme des probabilités de trouver le qubit dans l'un ou l'autre état doit être égale à 100 %. Quand vous effectuerez une mesure, vous obtiendrez soit 0, soit 1; il n'y a pas d'autre possibilité.

licensed under CC BY 4.0

Qubit

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{où} \quad \alpha, \beta \in \mathbb{C} \quad \text{et} \quad |\alpha|^2 + |\beta|^2 = 1$$

Superposition

Un qubit , grâce à la superposition, peut être à la fois 0 et 1 en même temps. Ce n'est pas l'un ou l'autre, mais une combinaison des deux, avec une certaine probabilité pour chaque état.

licensed under CC BY 4.0

Qubit

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{où} \quad \alpha, \beta \in \mathbb{C} \quad \text{et} \quad |\alpha|^2 + |\beta|^2 = 1$$

Superposition

Un qubit , grâce à la superposition, peut être à la fois 0 et 1 en même temps. Ce n'est pas l'un ou l'autre, mais une combinaison des deux, avec une certaine probabilité pour chaque état.

Intrication

L'intrication, c'est quand deux qubits (ou plus) sont liés de manière inséparable, peu importe la distance qui les sépare. L'état d'un qubit est instantanément corrélé à l'état de l'autre.

licensed under CC BY 4.0

Les Portes Quantiques

Pour effectuer un calcul, on doit manipuler l'état des qubits. On utilise pour cela des portes quantiques, qui sont des opérations. Elles font "pivoter" l'état de superposition des qubits.

- **Porte Hadamard (H)** : C'est la porte qui crée la superposition. Si tu appliques une porte Hadamard à un qubit qui est à 0, il entrera dans un état de superposition parfait (50% de chance d'être 0, 50% de chance d'être 1).
- **Porte CNOT (Controlled-NOT)** : C'est une porte qui opère sur deux qubits. Elle permet de créer l'intrication. Elle fonctionne comme un "SI... ALORS...". SI le premier qubit (le contrôle) est 1, ALORS elle inverse l'état du second qubit (la cible). Sinon, elle ne fait rien.

licensed under CC BY 4.0

Les Portes Quantiques

Pour effectuer un calcul, on doit manipuler l'état des qubits. On utilise pour cela des portes quantiques, qui sont des opérations. Elles font "pivoter" l'état de superposition des qubits.

- **Porte Hadamard (H)** : C'est la porte qui crée la superposition. Si tu appliques une porte Hadamard à un qubit qui est à 0, il entrera dans un état de superposition parfait (50% de chance d'être 0, 50% de chance d'être 1).
- **Porte CNOT (Controlled-NOT)** : C'est une porte qui opère sur deux qubits. Elle permet de créer l'intrication. Elle fonctionne comme un "SI... ALORS...". SI le premier qubit (le contrôle) est 1, ALORS elle inverse l'état du second qubit (la cible). Sinon, elle ne fait rien.

Mesure : Le retour au monde classique

Tant qu'on ne le mesure pas, le système quantique reste dans son état de superposition. La mesure est l'acte d'observer le qubit.
Cet acte force le qubit à choisir un état classique : soit 0, soit 1. L'état de superposition est alors "détruit".

- **Un résultat probabiliste :** Le résultat que l'on obtient (0 ou 1) dépend des probabilités définies par l'état de superposition juste avant la mesure. Si un qubit avait 70% de chance d'être 1 et 30% d'être 0, alors sur 100 mesures, on obtiendra environ 70 fois la valeur 1.
- **L'effondrement :** C'est comme si notre toupie en rotation s'arrêtait soudainement pour tomber sur pile ou face. Une fois tombée, elle reste sur ce résultat.

licensed under CC BY 4.0

C'est le cœur du calcul quantique ! Si la superposition permet d'explorer toutes les solutions en même temps, comment trouver la bonne ? Grâce à l'**interférence**.
Tout comme des vagues à la surface de l'eau, les probabilités associées aux états des qubits peuvent interagir.

- **Interférence constructive :** Quand deux vagues se rencontrent en phase, elles s'additionnent et créent une vague plus grande. En informatique quantique, un algorithme bien conçu va augmenter la probabilité (l'amplitude) des "bonnes" réponses.
- **Interférence destructive :** Quand deux vagues se rencontrent en opposition de phase, elles s'annulent. L'algorithme va faire en sorte que les probabilités des "mauvaises" réponses s'annulent mutuellement.

licensed under CC BY 4.0

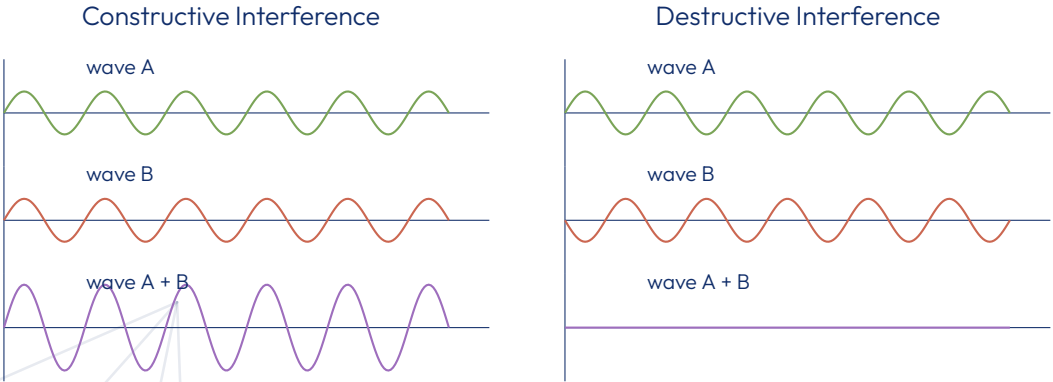


Figure 2 : Wave interference

licensed under CC BY 4.0

Algorithme de Shor : Le Principe i

Un algorithme hybride

Partie Classique

Résout efficacement un problème de factorisation si on sait résoudre efficacement un problème de recherche d'ordre.

Partie Quantique

Résout efficacement un problème de recherche d'ordre

licensed under CC BY 4.0

Algorithme de Shor : Le Principe ii

Partie Classique de l'algorithme Shor

Pour N un grand nombre à factoriser, on recherche un nombre a et un nombre r tels que :

$$a^r \equiv 1 \mod N$$

r est un nombre pair.

On a alors $(a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \mod N$.
Dans ce cas favorable $(a^{r/2} - 1)$ et $(a^{r/2} + 1)$ sont des facteurs non triviaux de N.
Dans le cas défavorable, on recommence avec un autre a.

licensed under CC BY 4.0

Partie Quantique de l'algorithme de Shor

Elle permet, étant donnés N et a de calculer le plus petit r tel que :

$$a^r \equiv 1 \mod N$$

r est appelé l'ordre de la fonction f_a de multiplication par a modulo N.

$$\begin{array}{rccc} f_a : & \mathbb{Z}/N\mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & ax \end{array}$$

licensed under CC BY 4.0

Algorithme de Shor : Le Principe iv

Fonction mathématique

$$f_a : \begin{array}{ccc} \mathbb{Z}/N\mathbb{Z} & \rightarrow & \mathbb{Z}/N\mathbb{Z} \\ x & \mapsto & ax \end{array}$$

Fonction quantique

$$\hat{U}_{f_a}|x \mod N\rangle = |ax \mod N\rangle$$

Si N est écrit sur b bits, le registre quantique est constitué de b bits.

licensed under CC BY 4.0

Algorithme de Shor : Le Principe iv

Fonction mathématique

$$f_a : \begin{array}{ccc} \mathbb{Z}/N\mathbb{Z} & \rightarrow & \mathbb{Z}/N\mathbb{Z} \\ x & \mapsto & ax \end{array}$$

Fonction quantique

$$\hat{U}_{f_a}|x \mod N\rangle = |ax \mod N\rangle$$

Si N est écrit sur b bits, le registre quantique est constitué de b bits.

- Comment $a^r \equiv 1 \mod N$, toutes les valeurs propres de $\hat{U}_{f_a}$ sont des racines r-ième de l'unité.

licensed under CC BY 4.0

Algorithme de Shor : Le Principe iv

Fonction mathématique

$$f_a : \begin{array}{ccc} \mathbb{Z}/N\mathbb{Z} & \rightarrow & \mathbb{Z}/N\mathbb{Z} \\ x & \mapsto & ax \end{array}$$

Fonction quantique

$$\hat{U}_{f_a}|x \mod N\rangle = |ax \mod N\rangle$$

Si N est écrit sur b bits, le registre quantique est constitué de b bits.

- Comment $a^r \equiv 1 \mod N$, toutes les valeurs propres de $\hat{U}_{f_a}$ sont des racines r-ième de l'unité.
- Grace à l'estimation quantique de phase, on peut calculer l'une des valeurs propres et en déduire la valeur de r.

licensed under CC BY 4.0

Algorithme de Shor : Le Principe v

$$\hat{U}_{f_a}|x \mod N\rangle = |ax \mod N\rangle$$
$$\hat{U}_{f_a} \circ \hat{U}_{f_a} \circ \ldots \circ \hat{U}_{f_a}|x \mod N\rangle = |a^k x \mod N\rangle$$
$$\hat{U}_{f_{(a^k)}}|x \mod N\rangle = |a^k x \mod N\rangle$$

licensed under CC BY 4.0

Algorithme de Shor : Le Principe v

$$\hat{U}_{f_a}|x \mod N\rangle = |ax \mod N\rangle$$
$$\hat{U}_{f_a} \circ \hat{U}_{f_a} \circ \ldots \circ \hat{U}_{f_a}|x \mod N\rangle = |a^k x \mod N\rangle$$
$$\hat{U}_{f_{(a^k)}}|x \mod N\rangle = |a^k x \mod N\rangle$$

Implémenter $\hat{U}_{f_{(a^k)}}$ ne coûte pas plus cher qu'implémenter U_{f_a}.

licensed under CC BY 4.0

Algorithme de Shor : Le Principe v

$$\hat{U}_{f_a}|x \mod N\rangle = |ax \mod N\rangle$$
$$\hat{U}_{f_a} \circ \hat{U}_{f_a} \circ \ldots \circ \hat{U}_{f_a}|x \mod N\rangle = |a^k x \mod N\rangle$$
$$\hat{U}_{f_{(a^k)}}|x \mod N\rangle = |a^k x \mod N\rangle$$

Implémenter $\hat{U}_{f_{(a^k)}}$ ne coûte pas plus cher qu'implémenter U_{f_a}.

On peut calculer $(U_{f_a})^k$ plus efficacement qu'en appliquant k fois U_{f_a}.

licensed under CC BY 4.0

Application de Shor à RSA

- **Objectif :** Factoriser $N = pq$.

licensed under CC BY 4.0

Application de Shor à RSA

- **Objectif :** Factoriser $N = pq$.
- **Le Lien :** La factorisation peut être réduite à la recherche de la période r de la fonction $f(x) = a^x \mod N$ pour un a aléatoire.

licensed under CC BY 4.0

Application de Shor à RSA

- **Objectif :** Factoriser $N = pq$.
- **Le Lien :** La factorisation peut être réduite à la recherche de la période r de la fonction $f(x) = a^x \mod N$ pour un a aléatoire.
- **Processus :**
  - Choisir a tel que $\texttt{pgcd}(a, N) \neq 1$.

licensed under CC BY 4.0

Application de Shor à RSA

- **Objectif :** Factoriser $N = pq$.
- **Le Lien :** La factorisation peut être réduite à la recherche de la période r de la fonction $f(x) = a^x \mod N$ pour un a aléatoire.
- **Processus :**
  - Choisir a tel que $\textbf{pgcd}(a, N) \neq 1$.
  - Utiliser l'algorithme de Shor pour trouver la période r de $a^x \mod N$. r est l'ordre de a dans $(\mathbb{Z}/N\mathbb{Z})^*$.

Application de Shor à RSA

- **Objectif :** Factoriser $N = pq$.
- **Le Lien :** La factorisation peut être réduite à la recherche de la période r de la fonction $f(x) = a^x \mod N$ pour un a aléatoire.
- **Processus :**
  - Choisir a tel que $\textbf{pgcd}(a, N) \neq 1$.
  - Utiliser l'algorithme de Shor pour trouver la période r de $a^x \mod N$. r est l'ordre de a dans $(\mathbb{Z}/N\mathbb{Z})^*$.
  - Si r est pair et $a^{r/2} \neq -1 \mod N$ alors les facteurs de N sont $\textbf{pgcd}(a^{r/2} - 1, N)$ et $\textbf{pgcd}(a^{r/2} + 1, N)$.

licensed under CC BY 4.0

Application de Shor à RSA

- **Objectif :** Factoriser $N = pq$.
- **Le Lien :** La factorisation peut être réduite à la recherche de la période r de la fonction $f(x) = a^x \mod N$ pour un a aléatoire.
- **Processus :**
  - Choisir a tel que $\textbf{pgcd}(a, N) \neq 1$.
  - Utiliser l'algorithme de Shor pour trouver la période r de $a^x \mod N$. r est l'ordre de a dans $(\mathbb{Z}/N\mathbb{Z})^*$.
  - Si r est pair et $a^{r/2} \neq -1 \mod N$ alors les facteurs de N sont $\textbf{pgcd}(a^{r/2} - 1, N)$ et $\textbf{pgcd}(a^{r/2} + 1, N)$.
- **Conclusion :** Shor résout la factorisation en temps polynomial, brisant RSA.

licensed under CC BY 4.0

Application de Short à ECDLP

- **Objectif :** Trouver q tel que $Q = d \times G$.

licensed under CC BY 4.0

Application de Short à ECDLP

- **Objectif :** Trouver q tel que $Q = d \times G$.
- Le problème du logarithme discret (quel que soit le groupe) peut aussi être résolu via la recherche de période.

licensed under CC BY 4.0

Application de Short à ECDLP

- **Objectif :** Trouver q tel que $Q = d \times G$.
- Le problème du logarithme discret (quel que soit le groupe) peut aussi être résolu via la recherche de période.
- **Processus :** On définit une fonction $f(x_1, x_2) = x_1 \times G + x_2 \times Q$. Trouver la période de cette fonction à 2 variables révlèle d'une relation $d \times x_1 + x_2 = 0$, ce qui permet de trouver d.

licensed under CC BY 4.0

Application de Short à ECDLP

- **Objectif :** Trouver q tel que $Q = d \times G$.
- Le problème du logarithme discret (quel que soit le groupe) peut aussi être résolu via la recherche de période.
- **Processus :** On définit une fonction $f(x_1, x_2) = x_1 \times G + x_2 \times Q$. Trouver la période de cette fonction à 2 variables révlèle d'une relation $d \times x_1 + x_2 = 0$, ce qui permet de trouver d.
- **Conclusion :** Shor résout ECDLP en temps polynomial brisant ECDSA, DH, ElGamal...

licensed under CC BY 4.0

Algorithme de Grover et la Cryptographie Symétrique

- **Problème résolu :** Recherche non structurée (liste non triée)
- **Principe :** "Amplification d'amplitude". Si un solution existe, Grover peut la trouver en $\mathcal{O}(\sqrt{N})$ requêtes à l'oracle, contre $\mathcal{O}(N)$ en classique.
- **Impacts sur la cryptographie symétrique (*e.g.* : AES)**
  - Une attaque par force brute sur une clé de k bits prend 2^k essais en classique.
  - Avec Grover cela prend $\mathcal{O}(\sqrt{2^k}) = \mathcal{O}(2^{k/2})$ opérations quantiques.
- **Conséquence :** Pour maintenir un niveau de sécurité de 128 bits contre un attaquant quantique, il faut utiliser des clés symétriques de 256 bits (*e.g.* : AES-256).
- **Bilan :** La cryptographie symétrique est **affaiblie**, mais pas **brisée**. On double la taille des clés et c'est réglé.

licensed under CC BY 4.0

D'une menace théorique à un Problème d'Ingénieurie

- **Nous avons établi que :**
  - **L'algorithme de Shor** brise la sécurité de **toute** la cryptographie asymétrique déployée aujourd'hui (RSA, ECDSA, DH, ...). Le problème mathématique sous-jacent est résolu en temps polynomial.
  - **L'algorithme de Grover** affaiblit la cryptgraphie symétrique, forçant une transition vers des clés plus longues (*e.g.* : AES-256), mais ne la brise pas fondamentalement.
- **Le débat n'est plus "si" mais "quand".**
  - La question passe du domaine de l'informatique théorique à celui de la physique et de l'ingénierie.
  - L'enjeu est de construire une machine capable d'exécuter ces algorithmes à une échelle pertinente.
- **Cette section explore la question la plus pressente :** Quel est le calendrier réel de cette menace et comment devons-nous l'interpréter en tant qu'architectes de la sécurité ?

licensed under CC BY 4.0

L'État de l'Art : Où en est l'Ordinateur Quantique ? (Hardware)

La course au qubits

- Les annonces de processeurs avec des centaines, voire plus de 1000 qubits, sont fréquentes (IBM, Google, etc.).
- **Mails le nombre de qubits est une métrique trompeuse s'il est pris isolément.**

licensed under CC BY 4.0

L'État de l'Art : Où en est l'Ordinateur Quantique ? (Hardware)

La course au qubits

- Les annonces de processeurs avec des centaines, voire plus de 1000 qubits, sont fréquentes (IBM, Google, etc.).
- **Mails le nombre de qubits est une métrique trompeuse s'il est pris isolément.**

Les Métriques qui comptent

- **Temps de Cohérence :** Durée pendant laquelle un qubit maintient son état quantique avant de "décohérer" à cause du bruit ambiant. Actuellement de l'ordre de quelques centaines de microsecondes pour les meilleurs supraconducteurs.
- **Fidélité des Portes Logiques :** Le taux d'erreur de chaque opération quantique. Doit être extrêmement faible $< 99.99\%$.
- **Connectivité :** Le nombre de qubits auxquels un qubit donné peut directement intéragir.
- **Quantum Volume (IBM) :** Une métrique qui tente de combiner ces facteurs pour mesurer la puissance réelle d'un processeur quantique.

licensed under CC BY 4.0

L'État de l'Art : Où en est l'Ordinateur Quantique ? (Hardware)

La course au qubits

- Les annonces de processeurs avec des centaines, voire plus de 1000 qubits, sont fréquentes (IBM, Google, etc.).
- **Mails le nombre de qubits est une métrique trompeuse s'il est pris isolément.**

Les Métriques qui comptent

- **Temps de Cohérence :** Durée pendant laquelle un qubit maintient son état quantique avant de "décohérer" à cause du bruit ambiant. Actuellement de l'ordre de quelques centaines de microsecondes pour les meilleurs supraconducteurs.
- **Fidélité des Portes Logiques :** Le taux d'erreur de chaque opération quantique. Doit être extrêmement faible $< 99.99\%$.
- **Connectivité :** Le nombre de qubits auxquels un qubit donné peut directement intéragir.
- **Quantum Volume (IBM) :** Une métrique qui tente de combiner ces facteurs pour mesurer la puissance réelle d'un processeur quantique.

Les Plateformes Dominantes

- **Qubits suprconducteurs :** Rapides mais sensibles au bruit et nécessitent des températuers cryogéniques.
- **Ions piégés :** Plus lents mais avec une meilleur fidélité et cohérence.

licensed under CC BY 4.0

Le Vrai Défi : Des Qubits Physiques aux Qubits Logiques

- **Les qubits actuels sont des "qubits physiques" :** Ils sont extrêmement bruités et instables. Un calcul un tant soit peu complexe échouse à cause de l'accumulation des erreurs.
- **L'algorithme de Shor a besoin de "qubits logiques" :** Des qubits parfaits, abstraits et sans erreur.
- **La Solution : La correction d'Erreurs QUantiques (QEC - Quantum Error Correction)**
  - **Principe :** Encoder l'information d'un qubit logique sur un grand nombre de qubits physiques redondants. Des algorithmes (codes de surface, etc.) détectent et corrigent en permanence les erreurs sur les qubits physiques, préservant ainsi l'état du qubit logique.
  - **Le Coût (Overhead) :** Les estimations acutelle suggèrent qu'il faudra entre **1000 et 100 000 qubits physiques** pour créer **un seul qubit logique** de haute qualité.
- **Calcul :** Pour briser RSA-2048, on estime qu'il faurait entre **20 millions de qubits physiques bruités** (étude de Google en 2019). C'est le principal goulot d'étranglement.

licensed under CC BY 4.0

L'Échéance : Quand le "YQ2" (Years to Quantum) ?

- **Il n'y a PAS de consensus, seulement des estimations basées sur la gestion du risque.**
- **Spectre des Opinations d'Experts :**
  - **Optimistes :** Certains acteurs du domaine (startups, etc.) prédisent une machine cryptographiquement pertinente d'ici **5 à 10 ans**.
  - **Consensus Prudent (Académique/Gouvernemental - NIST, ANSSI, ENISA) :** Le risque devient **significatif** dans une fenêtre de **10 à 20 ans**. Les politiques de migration sont basées sur cette fenêtre.
  - **Sceptiques :** Certains physiciens estiment que les obstacles de la correction d'erreur sont si fondamentaux que l'échéance pourrait être de **30 ans ou plus, voire jamais** à une échelle économique viable.
- **Le Point Clé :** Pour un cryptographe, une menace avec une probabilité non nulle dans 20 ans sur un sytème qui doit durer 30 ans est un **risque actuel** qui doit être mitigé **aujourd'hui**.

licensed under CC BY 4.0

Le théorème de Mosca : L'Urgence Cachée

Le Théorème de l'Urgence (Michele Mosca)

Si $x + y > z$, alors il faut s'inquiéter maintenant.

- x : Le temps nécessaire pour migrer nos systèmes vers une cryptographie post-quantique. (Estimé de 5-15 ans pour une migration globale).
- y : La durée de vie de nos secrets (*e.g.* : Secrets d'État : 50+ ans, secrets bancaires : 10+ ans, propriété intellectuelle : 20+ ans).
- z : Le temps qu'il faudra pour construire un ordinateur quantique pertinent. (Estimé à 10-20 ans).

Le Scénario d'Attaque "Harvest Now, Decrypt Later"

1. **Aujourd'hui :** Un adversaire intercepte et stocke massivement des données chiffrées (communications diplomatiques, données de santé, transactions financières...). Il ne peut pas les déchiffrer.
2. **Dans z années :** L'adversaire utilise son ordinateur quantique pour déchiffrer les données récoltées des années plus tôt.

Conclusion

Si la durée de vos secrets (y) est suffisamment longue, la migration (x) doit commencer bien avant l'arrivée de la machine (z). Pour les secrets d'État, nous sommes déjà dans la zone de danger.

licensed under CC BY 4.0

Thème de Discussion n°1 : Le Scénario de la Rupture

Le Scénario A : Le "Big Bang"

- Une entité (Google, un État, etc.) annonce publiquement avoir construit une machine capable de briser RSA-2048.
- Du jour au lendemain, la confiance dans l'infrastructure numérique mondiale s'effondre. C'est le chaos.
- Ce scénario est-il crédible ? Une telle avancée serait-elle annoncée publiquement plutôt que d'être gardée secrète ?

L'Érosion Progressive

- Une ou plusieurs agences de renseignement développent cette capacité en secret.
- Elles l'utilisent de manière ciblée, créant une asymétrie de l'information.
- On n'assiste pas à un effondrement, mais à une période de doute et de suspicion. Des "fuites" inexplicables se produisent. La confiance s'érode lentement.
- Ce scénario est-il plus probable ? Quelles seraient les conséquences géopolitiques d'une "suprématie quantique" secrète ?

licensed under CC BY 4.0

Thème de Discussion n°2 : L'Impact Économique et Sociétal

Imaginons un monde où ECDSA est cassé. Quelles sont les failles systémiques ?

licensed under CC BY 4.0

Imaginons un monde où ECDSA est cassé. Quelles sont les failles systémiques ?

- **Crypto-monnaise :** Toute adresse Bitcoin/Ethereum dont la clé publique est connée (*i.e.*toute adresse ayant déjà emis une transaction) peut être vidée par quiconque possédant un ordinateur quantique. Chute de la valeur des actifs.

licensed under CC BY 4.0

Imaginons un monde où ECDSA est cassé. Quelles sont les failles systémiques ?

- **Crypto-monnaise :** Toute adresse Bitcoin/Ethereum dont la clé publique est connée (*i.e.*toute adresse ayant déjà emis une transaction) peut être vidée par quiconque possédant un ordinateur quantique. Chute de la valeur des actifs.
- **Signature de Code :** Un attaquant pourrait signer des mises à jour malveillantes pour Windows, iOS, Android... qui seraient acceptées comme authentiques.

licensed under CC BY 4.0

Imaginons un monde où ECDSA est cassé. Quelles sont les failles systémiques ?

- **Crypto-monnaise :** Toute adresse Bitcoin/Ethereum dont la clé publique est connée (*i.e.*toute adresse ayant déjà emis une transaction) peut être vidée par quiconque possédant un ordinateur quantique. Chute de la valeur des actifs.
- **Signature de Code :** Un attaquant pourrait signer des mises à jour malveillantes pour Windows, iOS, Android... qui seraient acceptées comme authentiques.
- **Transactions Financières :** Les messages SWIFT, les ordres boursiers, pourraient être falsifiés.

licensed under CC BY 4.0

Thème de Discussion n°2 : L'Impact Économique et Sociétal

Imaginons un monde où ECDSA est cassé. Quelles sont les failles systémiques ?

- **Crypto-monnaise :** Toute adresse Bitcoin/Ethereum dont la clé publique est connée (*i.e.*toute adresse ayant déjà emis une transaction) peut être vidée par quiconque possédant un ordinateur quantique. Chute de la valeur des actifs.
- **Signature de Code :** Un attaquant pourrait signer des mises à jour malveillantes pour Windows, iOS, Android... qui seraient acceptées comme authentiques.
- **Transactions Financières :** Les messages SWIFT, les ordres boursiers, pourraient être falsifiés.
- **Identité Numérique et Gouvernement :** Les certificats des passeports biométriques, des cartes d'identité électroniques, déviendraint inutilisables. Les archives de l'État pourraient être compromises.

licensed under CC BY 4.0

Thème de Discussion n°2 : L'Impact Économique et Sociétal

Imaginons un monde où ECDSA est cassé. Quelles sont les failles systémiques ?

- **Crypto-monnaise :** Toute adresse Bitcoin/Ethereum dont la clé publique est connée (*i.e.*toute adresse ayant déjà emis une transaction) peut être vidée par quiconque possédant un ordinateur quantique. Chute de la valeur des actifs.
- **Signature de Code :** Un attaquant pourrait signer des mises à jour malveillantes pour Windows, iOS, Android... qui seraient acceptées comme authentiques.
- **Transactions Financières :** Les messages SWIFT, les ordres boursiers, pourraient être falsifiés.
- **Identité Numérique et Gouvernement :** Les certificats des passeports biométriques, des cartes d'identité électroniques, déviendraint inutilisables. Les archives de l'État pourraient être compromises.

Questions : Quels autres domaines critiques seraient touchées ?

licensed under CC BY 4.0

Synthèse de la Menace Quantique

Ce qu'il faut retenir :

- **La menace est théoriquement solide :** L'algorithme de Shor n'est pas une conjecture, ses mathématiques sont prouvées.
- **Le calendrier est incertain, mais la fenêtre de risque est ouverte.** Agir sur une base de 10-20 ans est une stratégie de gestion du risque prudente et largement acceptée.
- **L'attaque "Harvest Now, Decrypt Later" rend la menace immédiate** pour tutes les données à longue durée de vie.
- **Le principal obstacle est l'ingénierie,** notamment la correction d'erreurs quantiques pour passer des qubits physiques aux qubits logiques.
- **L'inaction n'est pas une option.** La migration vers la cryptographie post-quantique est un processur long et complexte qui doit être anticipé dès maintenant.

licensed under CC BY 4.0

Questions ??

4

Construction des standards PQC

Definition (Lattice)

Un réseau L est un sous-groupe discret de $\mathbb{R}^n$. C'est l'ensemble de toutes les combinaisons linéaires à coefficients **entiers** d'un ensemble de vecteurs de base linéairement indépendants $B = \{b_1, ..., b_n\}$.

$$L(B) = \{\sum x_i \times b_i | x_i \in \mathbb{Z}\}$$

Une "bonne" base a des vecteurs courts et quasi-orthogonaux. Une "mauvaise" base a des vecteurs longs et non-orthogonaux. Passer d'une mauvaise à une bonne base est difficile.

licensed under CC BY 4.0

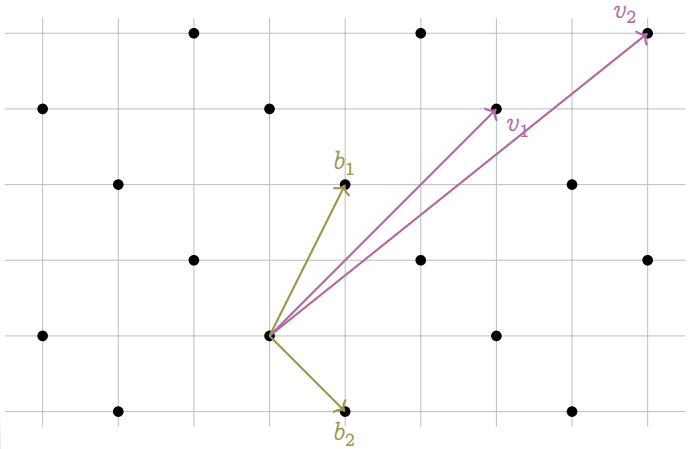


Figure 3 : Représentation des bases en 2 dimensions.

Problèmes Difficiles sur les Réseaux

- **SVP (*Shortest Vector Problem*) :** Trouver le vecteur non nul le plus court dans un réseau.
- **CVP (*Closest Vector Problem*) :** Étant donné un vecteur cible $t \notin L$, trouver le vecteur $v \in L$ le plus proche de t.
- **Problèmes de "Learning" :**
  - **SIS (*Short Integer Solution*) :** Étant donné une matrice A, trouver un vecteur non-nul z à petits coefficients entiers tel que : $A \times z \equiv 0 \mod q$.
  - **LWE (*Learning With Errors*) :** Étant donné une matrice A et un vecteur $b = A \times s + e$ (où s est un secret et e un vecteur d'erreurs "petit"), retrouver s.
- Ce sont ces problèmes (considérés quantiquement difficiles) qui sont à la base des chémas PQC les plus efficaces.

licensed under CC BY 4.0

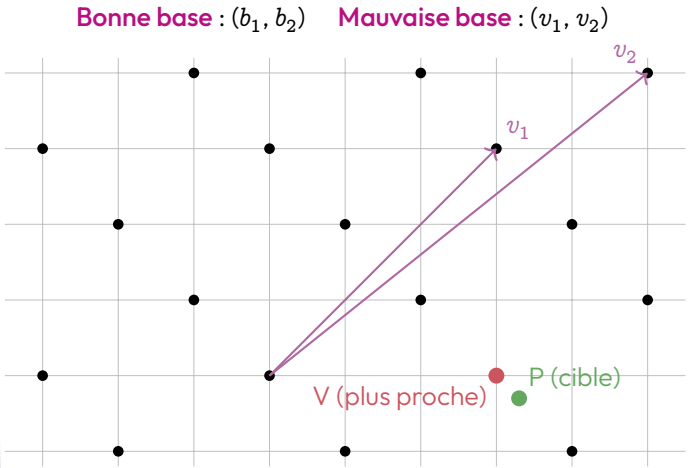


Figure 4 : Représentation du problème de vecteurs les plus proches (CVP).

Dilithiuem (ML-DSA) : Le Cheval de Bataille

Fondement

Modulel-LWE et Module-SIS sur des anneaux de polynômes : $R_q = \mathbb{Z}_q[X]/(X^n + 1)$.

- Travailler avec des polynomes permet d'obtenir des clés et des signatures beaucoup plus compactes que du LWE/SIS "standard".

Schéma de type "Fiat-Shamir with Aborts"

C'est la transformation d'un protocole d'identification interactif à 3 tours (Commit, Challenge, Response) en un schéma de signature non-interatif.

licensed under CC BY 4.0

Dilithiuem : Le Protocole Interactif Sous-Jacent

Données

- **Clé Privée :** (s_1, s_2).
- **Clé Publique :** $A, t = A \times s_1 + s_2$. Les coefficients s_i sont petits.

Protocole

1. **Prouveur → Vérificateur (Commit) :** Le prouveur choisit un y aléatoire à petit coefficients, calcule $w = A \times y$ et envoie w.
2. **Vérificateur → Prouveur (Challenge) :** Le vérificateur envoie un challenge c (polynôme avec peu de coefficients non nuls).
3. **Proveur → Vérificateur (Response) :** Le prouveur calcule et envoie $z = y + c \times s_1$.

Vérification interactive

Le vérificateur contrôle que les coefficients de z sont petits et que $A \times z - c \times t$ est "proche" de w.

licensed under CC BY 4.0

De l'Interactif au Non-Interactif (Fiat-Shamir)

L'astuce

Le prouveur peut simuler le challenge du vérificateur en le calculant lui-même.

licensed under CC BY 4.0

De l'Interactif au Non-Interactif (Fiat-Shamir)

L'astuce

Le prouveur peut simuler le challenge du vérificateur en le calculant lui-même.

Transformation

Le challenge c est généré en hachant la partie "commit" w avec le message à signer m.
$c = H(w, m)$.

licensed under CC BY 4.0

De l'Interactif au Non-Interactif (Fiat-Shamir)

L'astuce

Le prouveur peut simuler le challenge du vérificateur en le calculant lui-même.

Transformation

Le challenge c est généré en hachant la partie "commit" w avec le message à signer m.
$c = H(w, m)$.

Le signataire exécute les 3 étapes lui-même. La signature est (z, c).

licensed under CC BY 4.0

De l'Interactif au Non-Interactif (Fiat-Shamir)

L'astuce

Le prouveur peut simuler le challenge du vérificateur en le calculant lui-même.

Transformation

Le challenge c est généré en hachant la partie "commit" w avec le message à signer m.
$c = H(w, m)$.

Le signataire exécute les 3 étapes lui-même. La signature est (z, c).

Problème

Le z calculé peut avoir des coefficients trop grands, ce qui révélerait des informations sur la clé secrète.

licensed under CC BY 4.0

De l'Interactif au Non-Interactif (Fiat-Shamir)

L'astuce

Le prouveur peut simuler le challenge du vérificateur en le calculant lui-même.

Transformation

Le challenge c est généré en hachant la partie "commit" w avec le message à signer m.
$c = H(w, m)$.

Le signataire exécute les 3 étapes lui-même. La signature est (z, c).

Problème

Le z calculé peut avoir des coefficients trop grands, ce qui révélerait des informations sur la clé secrète.

Solution "with Aborts"

Le signataire vérifie lui-même la forme de z. Si elle est trop grande, il **rejette** cette tentative (abort) et recommance avec un nouveau y. Ce "rejection sampling" est crucial pour la sécurité.

licensed under CC BY 4.0

Fondement

Uniquement la sécurité des **fonctions de hachage**. Aucune hypothèse de complexité sur les réseaux, la factorisation, etc.

Construction ”bottom-up”

licensed under CC BY 4.0

Fondement

Uniquement la sécurité des **fonctions de hachage**. Aucune hypothèse de complexité sur les réseaux, la factorisation, etc.

Construction ”bottom-up”

1. **W-OTS+ (Winternitz One-Time Signature) :** Le bloc de base. Une clé permet de signer **un seul message**.

licensed under CC BY 4.0

Fondement

Uniquement la sécurité des **fonctions de hachage**. Aucune hypothèse de complexité sur les réseaux, la factorisation, etc.

Construction ”bottom-up”

1. **W-OTS+ (Winternitz One-Time Signature) :** Le bloc de base. Une clé permet de signer **un seul message**.
2. **HORST (HORS with Trees) :** Un schéma ”Few-Time” (quelques milliers de signatures).

licensed under CC BY 4.0

SPHINCS+ (SLH-DSA)

Fondement

Uniquement la sécurité des **fonctions de hachage**. Aucune hypothèse de complexité sur les réseaux, la factorisation, etc.

Construction ”bottom-up”

1. **W-OTS+ (Winternitz One-Time Signature) :** Le bloc de base. Une clé permet de signer **un seul message**.
2. **HORST (HORS with Trees) :** Un schéma ”Few-Time” (quelques milliers de signatures).
3. **Hypertree :** Une structure d’arbres de Merkle superposées pour signer un très grand nombres de messages.

licensed under CC BY 4.0

W-OTS+ : La Signature à Usage Unique

- **Clé privée :** l chaînes secrètes sk_i de n octets.
- **Clé publique :** On applique une fonction de chaînage f (basée sur un hachage) $w - 1$ fois à chaque sk_i pour obtenir des pk_i. La clé publique est le hash de la concaténation de tous les pk_i.
- **Signature :** Pour signer un hash H, on le découpe en l blocs. Chaque bloc h_i indique combient de fois il faut appliquer la fonction f à la clé secrète sk_i. La signature est la collection de ces résultats.
- **Vérification :** Le vérificateur applique f le nombre de fois manquant pour reconstruire les pk_i et vérifier la racine de la clé publique.

licensed under CC BY 4.0

L'Hypertree de SPHINCS+

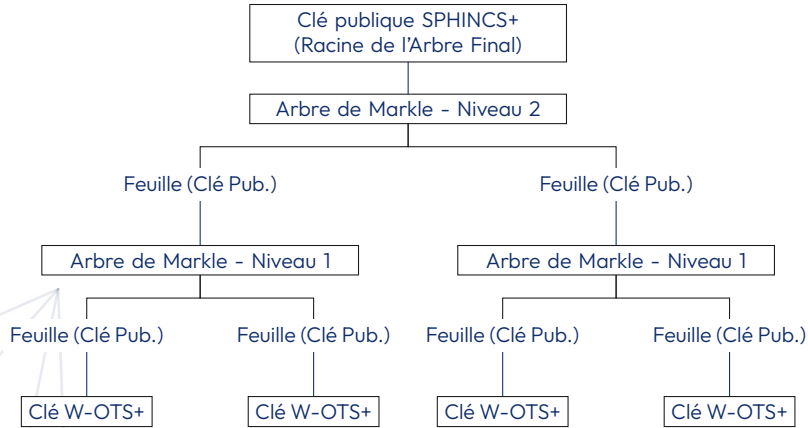


Figure 5 : SPHINCS Merkle Tree

licensed under CC BY 4.0

Comparaison Finale des Schémas

Algorithme	Fondement	Taille de Signature (niveau 3)	Vitesse signature	Confiance PQC
ECDSA P-256	ECDLP	64 B	Lente	**Nulle**
ML-DSA (Dilithium)	Module-LWE	3.3 kB	Très Rapide	Élevée
FN-DSA (Falcon)	NTRU Lattices	1.2 kB	Rapide	Élevée
SLH-DSA (SPHINCS+)	Fonctions de Hachages	16 kB	Très Lente	Maximale

Table 1 : Tableau comparatif des schémas de signatures Post-Quantique.

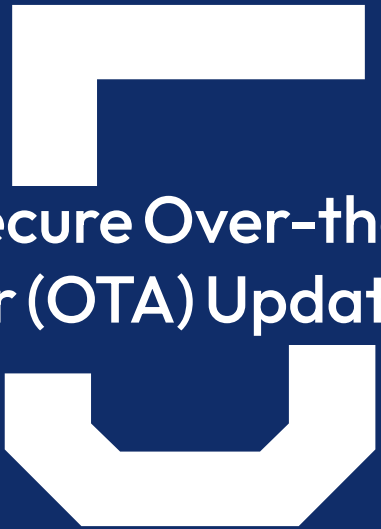
licensed under CC BY 4.0

Quelle signature pour quel usage ?

- **Dilithium :** Le successeur généraliste d'ECDSA/RSA pour la plupart des protocoles (TLS, etc.). Bon équilibre taille/performance.
- **Falcon :** Là où la taille de la signature est absolument critique, mais où la complexité de l'implémentation (arithmétique à virgule flottante) est acceptable.
- **SPHINCS+ :** Pour les applications de "haute assurance" où la confiance dans la sécurité à long terme prime sur la performance et la taille. (Ex : signature de firmware, archivage légal à très long terme).

licensed under CC BY 4.0

Questions ??



Secure Over-the-Air (OTA) Updates

Introduction au Secure OTA

- **Contexte :** Le monde de l'IoT, de l'automobile, des dispositifs médicaux... Des milliards d'appareils connectés, souvent sans surveillance physique (”unattended”).
- **Besoin Vital :** Mettre à jour le logiciel à distance pour :
  - Corriger des failles de sécurité critiques.
  - Ajouter des fonctionnalités.
  - Améliorer la performance.
- **Le risque :** Le mécanisme de mise à jour est un vecteur d'attaque extrêmement puissant. Un attaquant qui le compromet peut prendre le contrôle total d'une flotte entière d'appareils.

licensed under CC BY 4.0

Modèle de Menace pour les Mises à Jour OTA

Objectifs de l'attaquant

1. **Injection de Firmware Malveillant :** Installer un logiciel modifié pour "bricker" l'appareil, l'enrôler dans un botnet (ex : Mirai), ou exfiltrer des données. (Attaque contre l'Authenticité et l'Intégrité).
2. **Attaques par Retour en Arrière (Rollback Attack) :** Forcer un appareil à installer une version plus ancienne du logiciel, réintroduisant une faille de sécurité connue et déjà corrigée.
3. **Vol de Propriété Intellectuelle :**Intercepter la mise à jour pour faire du reverse engineering sur le firmware, qui peut contenir des secrets industriels. (Attaque contre la **Confidentialité**).
4. **Déni de Service :**Interrompre les mises à jour ou envoyer des mises à jour corrompues pour rendre les appareils inopérants.

licensed under CC BY 4.0

Architecture d'un Protocole Secure OTA

Composants Clés

- **Serveur de Build/Compilation :** Compile le code source en un binaire exécutable (l'image firmware `FW`).
- **Serveur de Signature :** Une infrastructure hautement sécurisée (souvent un **HSM – Hardware Security Module**) qui détient les clés privées de signature du fabricant. Son seul rôle est de signer des hash.
- **Serveur de Distribution (CDN) :** Stocke et distribue les images firmware signées aux appareils.
- **L'Appareil Cible (Device) :** Doit être capable de vérifier la mise à jour avant de l'installer. Possède une racine de confiance (Root of Trust).

licensed under CC BY 4.0

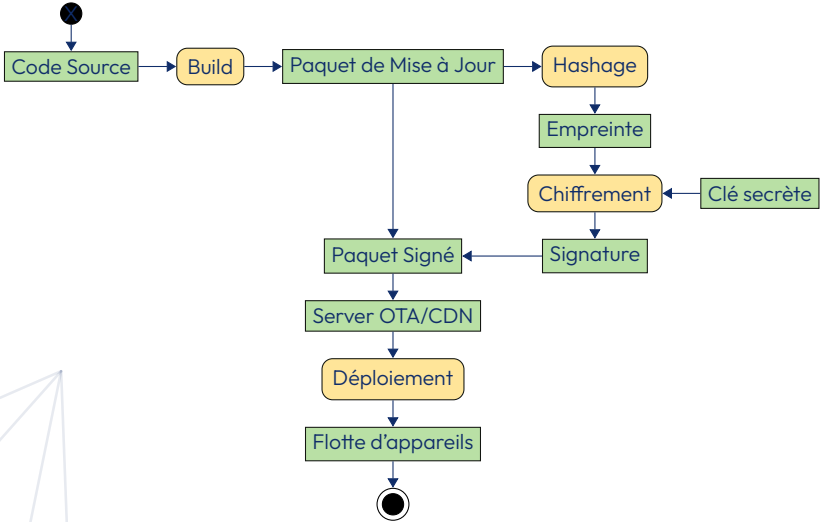


Figure 6 : Schéma d'un déploiement sécurisé OTA.

Le Workflow Cryptographique (Classique - ECDSA)

1. **Préparation**
   - Le serveur de build produit l'image **FW**.
   - Il créé un **manifeste**, un fichier de métadonnées contenant : `version_number`, `device_target`, `dependencies`, et surtout `H(FW)`.
2. **Signature**
   - La manifeste est envoyé aux HSM.
   - Le HSM signe le manifeste (ou directement le `H(FW)`) avec la clé privée du fabriquant `d_vendor`. $\sigma = \texttt{Sign}(d_{vendor}, \texttt{H}(manifest))$.
3. **Distribution**
   - Le serveur de distribution héberge le paquet de mise à jour : (`FW`, manifest, σ).
4. **Vérification sur l'appareil**
   - L'appareil télécharge le paquet.
   - **Étape 1 : Vérification de la signature.** Il calcule $\texttt{H}(manifest)$ et vérifie $sigma$ avec la clé publique Q_{vendor}. Si invalide → REJET.
   - **Étape 2 : Vérification du Rollback.** Il compare le `version_number` du manifeste à sa version actuelle. Si `version_new <= version_current` → REJET.
   - **Étape 3 : Vérification de l'intégrité.** Il calcule `H(FW)` et le compare au hash présent dans le manifeste. Si différent → REJET.
   - **Installation :** Uniquement si les 3 étapes réussissent.

licensed under CC BY 4.0

La Racine de Confiance (Root of Trust) Matérielle

Question : Comment faire confiance à la clé publique Q_{vendor} ?

licensed under CC BY 4.0

La Racine de Confiance (Root of Trust) Matérielle

Question : Comment faire confiance à la clé publique Q_{vendor} ?

Solution :

- La clé publique Q_{vendor} (ou hash de celle-ci) est gravée de manière **immuable** dans le hardware du microcontrôleur au moment de la fabrication (*e.g.* : fusibles OTP —One-Time Programmable—, mémoire flash protégée en écriture).
- Au démarrage, le code bas niveau inaltérable (**BootROM**) utilise cette clé pour vérifier la signature du premier logiciel mutable (**bootloader**).
- Le bootloader vérifie ensuite l'application principale, etc.

Ceci permet de créer une chaîne de confiance ininterrompue, du silicium jusqu'à l'application qui exécute la vérification OTA.

licensed under CC BY 4.0

Contraintes de l'Embarqué & Adéquation d'ECDSA

- **Bande passant limitée :** Souvent très limitée et coûteuse (LoRaWan, NB-IoT, 4G, etc.).
- **Énergie :** La plupart des équipement (I)IoT fonctionnent sur batterie. Chaque cycle CPU consomme de l'énergie.
- **Stockage :** La mémoire flash et la RAM sont limitées (quelques centaines de ko).
- **Pourquoi ECDSA est parfait ?**
  - **Signature Compacte (64 octets) :** Idéal pour les réseaux à faible bande passante.
  - **Clé publique Compacte (64 octets) :** Occupe peu de place dans la précieuse mémoire immuable.
  - **Vérification Rapide :** L'opération la plus fréquente $u_1 \times G + u_2 \times Q$ est efficace, ce qui préserve la batterie.

licensed under CC BY 4.0

Problème de l'OTA à l'Ère Post-Quantique

Le Scénario Catastrophe

1. Un attaquant disposant d'un ordinateur quantique observe les mises à jour publiques. Il dispose de Q_{vendor} et de nombreuses signatures (r, s).
2. Il utilise l'algorithme de Shor pour calculer d_{vendor} à partir de Q_{vendor}.
3. L'attaquant possède désormais la "clé maîtresse" du fabriquant. Il peut signer n'importe quel firmware malveillant.
4. Les appareils, ne voyant aucune différence cryptographique, accepteront et installeront ce firmware.

licensed under CC BY 4.0

Problème de l'OTA à l'Ère Post-Quantique

Le Scénario Catastrophe

1. Un attaquant disposant d'un ordinateur quantique observe les mises à jour publiques. Il dispose de Q_{vendor} et de nombreuses signatures (r, s).
2. Il utilise l'algorithme de Shor pour calculer d_{vendor} à partir de Q_{vendor}.
3. L'attaquant possède désormais la "clé maîtresse" du fabriquant. Il peut signer n'importe quel firmware malveillant.
4. Les appareils, ne voyant aucune différence cryptographique, accepteront et installeront ce firmware.

Le Facteur temps

Les appareils IoT et automobiles ont des cycles de vie de 10, 15, voire 20 ans. Un appareil vendu aujourd'hui doit être sécurisé contre une menace quantique qui se matérialisera peut-être dans 10 ans.

licensed under CC BY 4.0

Les Défis de la Migration PQC pour l'OTA

Le Choc des Tailles

- **Signature ECDSA P-256 :** 64 octets
- **Signature ML-DSA (Dilithium-2) :** 2.4 ko (**38x plus gros**)
- **Signature SLH-DSA (SPHINCS+-128s) :** 8 ko (**125x plus gros**)

Impact Direct

- **Réseau :** Explosion du temps de transmission et de la consommation d'énergie de la radio. Pour un réseau LoRaWAN, transmettre 8 ko peut prendre plusieurs minutes et consommer une part significative de la batterie.
- **Stockage :** L'appareil doit pouvoir stocker la nouvelle image ET la signature avant vérification.
- **RAM & CPU :** Les algorithmes PQC, en particulier la vérification SPHINCS+, nécessitent plus de RAM et de cycles CPU.

licensed under CC BY 4.0

Stratégie de Migration n°1 : Le Schéma Hybride

Principe

Ne pas abandonner l'algorithme classique immédiatement. On signe le manifeste deux fois.

$$\sigma_{total} = (\sigma_{ecdsa}, \sigma_{pqc})$$

Logique de l'appareil

- Il télécharge le paquet contenant les deux signatures.
- Il vérifie σ_{ecdsa} avec $Q_{ventor,ecdsa}$.
- Il vérifie σ_{pqc} avec $Q_{vendor,pqc}$.
- **La mise à jour est acceptée si les DEUX signatures sont valides.**

Avantages

- **"Don't break that works" :** La sécurité reste assurée par ECDSA contre les attaquants classiques, même si une faille est trouvée dans la nouvelle implémentation PQC.
- **Protection Quantique :** Assurée par la signature quantique.

Inconvénients

Le pire des deux mondes en termes de taille et de temps de vérification. **C'est une mesure de transition.**

licensed under CC BY 4.0

Stratégie de Migration n°2 : La Mise à Jour de la Racine de Confiance

Problème

Si Q_{vendor} est gravé dans les fusibles OTP, **on ne peut jamais changer d'algorithme**.

licensed under CC BY 4.0

Stratégie de Migration n°2 : La Mise à Jour de la Racine de Confiance

Problème

Si Q_{vendor} est gravé dans les fusibles OTP, **on ne peut jamais changer d'algorithme**.

Solution

Mettre en place un mécanisme pour mettre à jour la clé de vérification elle-même.

1. **Phase 1 :** L'appareil est fabriqué avec une RoT classique (*e.g.* : ECDSA).
2. **Phase 2 (transition) :** Le fabricant déploie une mise à jour spéciale et unique. Cette mise à jour contient une nouvelle clé publique PQC ($Q_{vendor,pqc}$) et le nouveau code de vérification.
3. Cette mise à jour spéciale est signée avec l'ancienne clé ECDSA. L'appareil la valide, l'installe, et "promouvoit" la nouvelle clé PQC comme la nouvelle racine de confiance pour les futures mises à jour OTA.
4. **Phase 3 :** Toutes les mises à jour suivantes sont signées uniquement avec la clé PQC.

licensed under CC BY 4.0

Stratégie de Migration n°2 : La Mise à Jour de la Racine de Confiance

Problème

Si Q_{vendor} est gravé dans les fusibles OTP, **on ne peut jamais changer d'algorithme**.

Solution

Mettre en place un mécanisme pour mettre à jour la clé de vérification elle-même.

1. **Phase 1 :** L'appareil est fabriqué avec une RoT classique (*e.g.* : ECDSA).
2. **Phase 2 (transition) :** Le fabricant déploie une mise à jour spéciale et unique. Cette mise à jour contient une nouvelle clé publique PQC ($Q_{vendor,pqc}$) et le nouveau code de vérification.
3. Cette mise à jour spéciale est signée avec l'ancienne clé ECDSA. L'appareil la valide, l'installe, et "promouvoit" la nouvelle clé PQC comme la nouvelle racine de confiance pour les futures mises à jour OTA.
4. **Phase 3 :** Toutes les mises à jour suivantes sont signées uniquement avec la clé PQC.

Risque

L'opération de mise à jour de la RoT est extrêmement sensible. Toute erreur peut "bricker" l'appareil de manière irréversible.

licensed under CC BY 4.0

SPHINCS+ : Le Choix de la Haute Assurance pour l'OTA

Scénario

Firmware de contrôleurs de réseaux électriques, calculateurs automobiles (ECU) critiques (freinage, direction), dispositifs médicaux implantables.

Pourquoi SPHINCS+ ?

- Sa sécurité ne repose que sur des hypothèses minimalistes (la sécurité des fonctions de hachage), qui sont très bien comprises.
- Pour des systèmes avec une espérance de vie de 20-30 ans, cette assurance à très long terme est plus importante que la performance.

Acceptation des contraintes

- Les ingénieurs doivent concevoir le protocole de communication pour gérer des signatures de plusieurs dizaines de ko (fragmentation, retransmission, etc.).
- Le surcoût en performance de vérification est jugé acceptable au vu de l'enjeu sécuritaire.

licensed under CC BY 4.0

Tableau Récapitulatif : Menaces OTA & Mitigations

Menace	Mitigation Classique (ECDSA)	Mitigation PQC	Défi de la Migration
Firmware falsifié	Signature ECDSA	Signature ML-DSA ou SLH-DSA	Taille, Performance
Rollback Attack	Compteur de version anti-rollback (sécurisé, monotone)	Identique	Aucun (orthogonal à l'algorithme)
Vol de PI [1]	Chiffrement du firmware (*e.g.* : AES-GCM) + KEM/ECDH	Chiffrement + KEM PQC (ML-KEM)	Taille du KEM et du chiffré
Compromission de la Clé Serveur	HSM pour d_{vendor}	HSM pour clé dérivée PQC	La clé privée PQC est plus grande.
Attaque Quantique	**Aucune**	L'algorithme PQC lui-même	

Table 2 : Tableau Récapitulatif : Menaces OTA & Mitigations

1. PI : Propriété Intellectuelle.

licensed under CC BY 4.0

Questions ??

Références i

[1] Dan Boneh, Richard A. DeMillo et Richard J. Lipton. “On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)”. In : *Advances in Cryptology - EUROCRYPT ’97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*. T. 1233. Lecture Notes in Computer Science. Springer, 1997, p. 37-51. doi : `10.1007/3-540-69053-0_4`.

[2] Kai Michaelis, Christopher Meyer et Jörg Schwenk. “Randomly Failed! The State of Randomness in Current Java Implementations”. In : *Topics in Cryptology – CT-RSA 2013*. Sous la dir. d’Ed Dawson. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013, p. 129-144. isbn : 978-3-642-36095-4.

[3] *Some SecureRandom Thoughts — android-developers.googleblog.com*. `https://android-developers.googleblog.com/2013/08/some-securerandom-thoughts.html`. [Accessed 25-09-2025].

licensed under CC BY 4.0