

Cryptographie Post-Quantique

Cours réalisé par Théo PEUCHLESTRADE et Julian RETY

Dans le cadre de la matière **Découverte et initiation à la recherche**
Sous la direction de **Pascal LAFOURCADE**

Mars 2023



Table des matières

- 1 Introduction
- 2 Lattice Based Cryptography
- 3 Code Based Cryptography
- 4 Hash Based Cryptography
- 5 Multivariate Polynomial Cryptography
- 6 Post-Quantum Cryptography on FPGA Based on Isogenies on Elliptic Curves
- 7 Bibliographie

Introduction

Définition : Cryptographie (Larousse)

La **cryptographie** est l'ensemble des techniques de chiffrement qui assurent l'inviolabilité de textes et, en informatique, de données.

Exemple : Chiffrement RSA, cryptographie sur les courbes elliptiques (ECC)

Définition : Cryptographie (Larousse)

La **cryptographie** est l'ensemble des techniques de chiffrement qui assurent l'inviolabilité de textes et, en informatique, de données.

Exemple : Chiffrement RSA, cryptographie sur les courbes elliptiques (ECC)

Problème : Les chiffrements RSA ou ECC sont rendus inefficaces à cause de la rapidité de calcul des ordinateurs quantiques.

Définition : Cryptographie (Larousse)

La **cryptographie** est l'ensemble des techniques de chiffrement qui assurent l'inviolabilité de textes et, en informatique, de données.

Exemple : Chiffrement RSA, cryptographie sur les courbes elliptiques (ECC)

Problème : Les chiffrements RSA ou ECC sont rendus inefficaces à cause de la rapidité de calcul des ordinateurs quantiques.

Solution : La cryptographie post-quantique

Définition : Cryptographie (Larousse)

La **cryptographie** est l'ensemble des techniques de chiffrement qui assurent l'inviolabilité de textes et, en informatique, de données.

Exemple : Chiffrement RSA, cryptographie sur les courbes elliptiques (ECC)

Problème : Les chiffrements RSA ou ECC sont rendus inefficaces à cause de la rapidité de calcul des ordinateurs quantiques.

Solution : La cryptographie post-quantique

Définition : Cryptographie Post-Quantique (Wikipédia)

La **cryptographie post-quantique** est une branche de la cryptographie visant à garantir la sécurité de l'information face à un attaquant disposant d'un calculateur quantique.

Ordinateur (calculateur) quantique



Ordinateur quantique D-Wave The Advantage™ avec plus de 5,000 Qubits.

ISIMA

Les principaux candidats

Les principaux candidats pour la cryptographie post-quantique sont les suivants :

- Lattice Based Cryptography

Les principaux candidats

Les principaux candidats pour la cryptographie post-quantique sont les suivants :

- Lattice Based Cryptography
- Code Based Cryptography

Les principaux candidats pour la cryptographie post-quantique sont les suivants :

- Lattice Based Cryptography
- Code Based Cryptography
- Hash Based Cryptography

Les principaux candidats pour la cryptographie post-quantique sont les suivants :

- Lattice Based Cryptography
- Code Based Cryptography
- Hash Based Cryptography
- Multivariate Polynomial Cryptography

Les principaux candidats pour la cryptographie post-quantique sont les suivants :

- Lattice Based Cryptography
- Code Based Cryptography
- Hash Based Cryptography
- Multivariate Polynomial Cryptography
- Post-Quantum Cryptography on FPGA Based on Isogenies on Elliptic Curves

Lattice Based Cryptography

Le calcul du **déterminant d'une matrice** carrée A =

$$\begin{pmatrix} a_{1;1} & \cdots & a_{1;n} \\ \vdots & \ddots & \vdots \\ a_{n;1} & \cdots & a_{n;n} \end{pmatrix}$$

est donné par la formule de Leibniz

$$\det(A) = \begin{vmatrix} a_{1;1} & \cdots & a_{1;n} \\ \vdots & \ddots & \vdots \\ a_{n;1} & \cdots & a_{n;n} \end{vmatrix} = \sum_{\sigma \in \mathfrak{S}_n} \varepsilon(\sigma) \prod_{i=1}^n a_{\sigma(i), i}$$

où \mathfrak{S}_n désigne l'ensemble des permutations de $\{1, \dots, n\}$ et $\varepsilon(\sigma)$ la signature de la permutation σ

Le **produit scalaire** de deux vecteur u et v est donné par la formule suivante :

$$\langle u, v \rangle = \|u\| \times \|v\| \times \cos(\widehat{u, v})$$

Qu'est-ce que la Lattice Based Cryptography ?

Définition : *Lattice Based Cryptography (LBC)*

La cryptographie à base de réseaux euclidiens (ou cryptographie à base de treillis) est une technique reposant sur des problèmes mathématiques liés aux réseaux euclidiens.

Qu'est-ce que la Lattice Based Cryptography ?

Définition : *Lattice Based Cryptography (LBC)*

La cryptographie à base de réseaux euclidiens (ou cryptographie à base de treillis) est une technique reposant sur des problèmes mathématiques liés aux réseaux euclidiens.

Définition : *Réseau*

Un réseau (ou treillis) est un ensemble discret de points dans un espace euclidien. Il est défini par une base, qui est un ensemble de vecteurs linéairement indépendants servant de points de référence pour générer l'ensemble des points du réseau.

Qu'est-ce que la Lattice Based Cryptography ?

Définition : *Lattice Based Cryptography (LBC)*

La cryptographie à base de réseaux euclidiens (ou cryptographie à base de treillis) est une technique reposant sur des problèmes mathématiques liés aux réseaux euclidiens.

Définition : *Réseau*

Un réseau (ou treillis) est un ensemble discret de points dans un espace euclidien. Il est défini par une base, qui est un ensemble de vecteurs linéairement indépendants servant de points de référence pour générer l'ensemble des points du réseau.

Remarque : Les problèmes algorithmiques sous-jacents sont considérés comme difficiles à résoudre, même pour les ordinateurs quantiques.

Qu'est-ce qu'un treilli ?

Définition : Treilli

Un **treillis m -dimensionnel** A est un sous-groupe discret de plein rang de \mathbb{R}^m .

Qu'est-ce qu'un treilli ?

Définition : Treilli

Un **treillis m -dimensionnel** A est un sous-groupe discret de plein rang de \mathbb{R}^m .

Définition : Base

Une base de A est un ensemble de vecteurs linéairement indépendants dont les combinaisons linéaires entières génèrent A .

Remarque : En cryptographie, on s'intéresse généralement aux treillis d'entiers, c'est-à-dire à ceux dont les points ont des coordonnées dans \mathbb{N}^m .

Parmi ces treillis, on trouve les treillis " q -aires" définis comme suit :

Pour tout entier $q \geq 2$ et tout $A \in \mathbb{N}_q^{n \times m}$, on définit :

$$:= e \in \mathbb{N}^m : A.e = 0 \text{ mod } q$$

Ces treillis sont d'un intérêt particulier en cryptographie. La distance minimale d'un treillis A est la longueur du vecteur non nul le plus court :

$$\lambda_1(A) = \min_{v \in A \setminus \{0\}} \| v \|$$

Ici, représente la norme euclidienne. En général, le i^{me} minima successif $\lambda_i(A)$ est le plus petit rayon r tel que A ait i vecteurs linéairement indépendants de norme au plus r .

Problèmes difficiles

Problèmes difficiles

Les cryptosystèmes LBC sont basés sur des problèmes difficiles sur les réseaux euclidiens.
Parmi ces problèmes nous pouvons citer :

- Le problème du plus court vecteur (SVP)

Problèmes difficiles

Les cryptosystèmes LBC sont basés sur des problèmes difficiles sur les réseaux euclidiens.
Parmi ces problèmes nous pouvons citer :

- Le problème du plus court vecteur (SVP)
- Le problème du vecteur le plus court approximatif ($SVP\gamma$)

Les cryptosystèmes LBC sont basés sur des problèmes difficiles sur les réseaux euclidiens.
Parmi ces problèmes nous pouvons citer :

- Le problème du plus court vecteur (SVP)
- Le problème du vecteur le plus court approximatif ($SVP\gamma$)
- Le problème de l'apprentissage avec erreurs (LWE)

Les cryptosystèmes LBC sont basés sur des problèmes difficiles sur les réseaux euclidiens.
Parmi ces problèmes nous pouvons citer :

- Le problème du plus court vecteur (SVP)
- Le problème du vecteur le plus court approximatif ($SVP\gamma$)
- Le problème de l'apprentissage avec erreurs (LWE)
- Le problème du module apprentissage avec erreurs (MLWE)

Les cryptosystèmes LBC sont basés sur des problèmes difficiles sur les réseaux euclidiens.
Parmi ces problèmes nous pouvons citer :

- Le problème du plus court vecteur (SVP)
- Le problème du vecteur le plus court approximatif ($SVP\gamma$)
- Le problème de l'apprentissage avec erreurs (LWE)
- Le problème du module apprentissage avec erreurs (MLWE)
- ...

Les cryptosystèmes LBC sont basés sur des problèmes difficiles sur les réseaux euclidiens. Parmi ces problèmes nous pouvons citer :

- Le problème du plus court vecteur (SVP)
- Le problème du vecteur le plus court approximatif ($SVP\gamma$)
- Le problème de l'apprentissage avec erreurs (LWE)
- Le problème du module apprentissage avec erreurs (MLWE)
- ...

Remarque : Il existe de nombreux autres problèmes tels que le $SVP\gamma$, le GapSVP, le SIVP ...

Shortest Vector Problem (SVP)

SVP :

Soit une base B d'un treillis $A = A(B)$, trouver un vecteur non nul $v \in A(B)$ tel que :

$$\|v\| = \lambda_1(A(B))$$

Nous notons qu'il y a une borne sur $\lambda_1(A(B))$ par le premier théorème de Minkowski, qui stipule que pour tout réseau de rang complet $A(B)$ de rang n :

$$\lambda_1(A(B)) \leq \sqrt{n}(\det(A(B)))^{1/n}$$

Approximate Shortest Vector Problem (SVP γ)

SVP_γ :

Soit $\gamma \geq 1$ un facteur d'approximation, soit une base B d'un treillis $A = A(B)$ à n dimensions, trouver un vecteur non nul $v \in A(B)$ tel que :

$$\|v\| = \gamma \cdot \lambda_1(A(B))$$

Learning with Errors (LWE)

LWE :

- extension du problème d'apprentissage de la parité avec le bruit, qui est lui-même considéré comme très difficile

LWE :

- extension du problème d'apprentissage de la parité avec le bruit, qui est lui-même considéré comme très difficile
- étroitement lié aux problèmes de décodage dans la théorie du codage, qui sont également considérés comme très difficiles

LWE :

- extension du problème d'apprentissage de la parité avec le bruit, qui est lui-même considéré comme très difficile
- étroitement lié aux problèmes de décodage dans la théorie du codage, qui sont également considérés comme très difficiles
- possède une connexion dans le pire des cas

LWE :

- extension du problème d'apprentissage de la parité avec le bruit, qui est lui-même considéré comme très difficile
- étroitement lié aux problèmes de décodage dans la théorie du codage, qui sont également considérés comme très difficiles
- possède une connexion dans le pire des cas

LWE et sa variante, **MLWE**, sont utilisés comme fondements pour de nombreux systèmes cryptographiques modernes. Ils permettent le chiffrement de clé et nombreuses autres choses essentielles à l'élaboration d'un cryptosystème solide.

LWE :

- extension du problème d'apprentissage de la parité avec le bruit, qui est lui-même considéré comme très difficile
- étroitement lié aux problèmes de décodage dans la théorie du codage, qui sont également considérés comme très difficiles
- possède une connexion dans le pire des cas

LWE et sa variante, **MLWE**, sont utilisés comme fondements pour de nombreux systèmes cryptographiques modernes. Ils permettent le chiffrement de clé et nombreuses autres choses essentielles à l'élaboration d'un cryptosystème solide.

Le problème **Module-LWE** utilise des anneaux plutôt que des espaces vectoriels.

Formulation mathématique du problème LWE :

Soit $q = q(n) \geq 2$ un entier et soit $\chi = \chi(n)$ une distribution sur \mathbb{Z} . Le problème LWE consiste à distinguer les deux distributions :

- ① Échantillonner (a_i, b_i) uniformément de \mathbb{Z}_q^{n+1} .
- ② Un premier tirage $s \leftarrow \mathbb{Z}_q^n$ uniformément, puis échantillonner $(a_i, b_i) \in \mathbb{Z}_q^{n+1}$ par échantillonnage de $a_i \leftarrow \mathbb{Z}_q^n$ uniformément, $e_i \leftarrow \chi$ et en définissant $b_i = \langle a_i, s \rangle + e_i$.

L'hypothèse LWE est que le problème LWE est **irréalisable**.

Module Learning with Errors (MLWE)

Formulation mathématique du problème Module-LWE :

Soit k un entier positif, R et R_q les anneaux $\mathbb{Z}[X]/(X^n + 1)$ et $\mathbb{Z}_q[X]/(X^n + 1)$, s (le secret, aussi appelé la clé privée) un vecteur (ou une matrice) d'éléments tirés de l'anneau de polynômes R_q , e (l'erreur) un vecteur (ou une matrice) d'éléments tirés du même anneau de polynômes R_q et a_i et b_i des vecteurs (ou des matrices) connus.

L'anneau R_q dans ce contexte est un anneau de polynômes à coefficients entiers modulo un nombre premier q . Il est défini comme suit:

$$R_q = \mathbb{Z}_q[X]/(f(X))$$

où :

- Z_q représente l'ensemble des entiers modulo q , avec q étant un nombre premier.
- X est la variable du polynôme.
- $f(X)$ est un polynôme irréductible de degré n , c'est-à-dire un polynôme qui ne peut pas être factorisé en polynômes de degré inférieur à n avec des coefficients dans Z_q .

L'erreur e est choisie aléatoirement selon une certaine distribution, généralement centrée autour de zéro et à faible variance. L'erreur est ajoutée aux équations Module-LWE pour rendre le problème plus difficile à résoudre.

Le problème difficile Module-LWE consiste à distinguer des échantillons homogènes $(a_i, b_i) \leftarrow R_q^k \times R_q$ depuis les échantillons (a_i, b_i) où $a_i \leftarrow R_q^k$ est uniforme et $b_i = a_i^T s + e_i$ avec $s \leftarrow \beta_\eta^k$ commun à tous les échantillons et $e_i \leftarrow \beta_\eta$ 'frais' pour chaque échantillon.

Plus précisément, pour un algorithme A , nous définissons :

$$\begin{aligned} \text{Adv}_{m,k,\eta}^{\text{mlwe}}(A) &= |\Pr[b' = 1 : A \leftarrow R_q^{m \times k}; (s, e) \leftarrow \beta_\eta^k \times \beta_\eta^m; b = As + e; b'(A, b)] \\ &\quad - \Pr[b' = 1 : A \leftarrow R_q^{m \times k}; b \leftarrow R_q^m; b'(A, b)]| \end{aligned}$$

La fonction d'avantage Adv représente la probabilité \Pr qu'un adversaire A puisse réussir à casser le schéma cryptographique, c'est-à-dire la différence entre la probabilité de l'adversaire de gagner un jeu de sécurité et la probabilité de gagner le jeu par pur hasard.

Avantages et inconvénients de la Lattice Based Cryptography

Avantages et inconvénients

Avantages :

- résistance avérée aux attaques quantiques

Avantages :

- résistance avérée aux attaques quantiques
- possibilité de concevoir des primitives cryptographiques avancées (chiffrement homomorphe)

Avantages :

- résistance avérée aux attaques quantiques
- possibilité de concevoir des primitives cryptographiques avancées (chiffrement homomorphe)
- technique la plus prometteuse

Avantages et inconvénients

Avantages :

- résistance avérée aux attaques quantiques
- possibilité de concevoir des primitives cryptographiques avancées (chiffrement homomorphe)
- technique la plus prometteuse

Inconvénients :

- elle n'est pas encore normalisée (NIST)

Avantages :

- résistance avérée aux attaques quantiques
- possibilité de concevoir des primitives cryptographiques avancées (chiffrement homomorphe)
- technique la plus prometteuse

Inconvénients :

- elle n'est pas encore normalisée (NIST)
- elle est très peu implémentée concrètement

KRISTAL-Kyber

Fonctions nécessaires à Kyber :

- *Sam*, une fonction de **sortie extensible**

Fonctions nécessaires à Kyber :

- *Sam*, une fonction de **sortie extensible**
- *Compress*, une fonction de **compression**

Fonctions nécessaires à Kyber :

- *Sam*, une fonction de **sortie extensible**
- *Compress*, une fonction de **compression**
- *Decompress*, une fonction de **décompression**

Compress - Decompress (1/3)

Nous définissons maintenant une fonction $\text{Compress}_q(x, d)$ qui prend un élément $x \in \mathbb{Z}_q$ et produit un entier dans $0, \dots, 2d - 1$, où $d < \lceil q^{2d+1} \rceil$.

Nous définissons la fonction $\text{Decompress}_q(x, d)$, telle que :

$$x' = \text{Decompress}_q(\text{Compress}_q(x, d), d) \quad (1)$$

x' est un élément proche de x ; plus précisément :

$$|x' - x \bmod^{\pm} q| \leq B_q := \lceil \frac{q}{2^{d+1}} \rceil.$$

Compress - Decompress (2/3)

Les fonctions satisfaisant ces exigences sont définies comme suit :

$$\text{Compress}_q(x, d) = \lceil \frac{2^d}{q} \cdot x \rceil \bmod 2^d,$$

$$\text{Decompress}_q(x, d) = \lceil \frac{q}{2^d} \cdot x \rceil$$

Compress - Decompress (3/3)

La principale raison de définir les fonctions Compress_q et Decompress_q est de pouvoir éliminer certains bits de poids faible dans la clé publique et le texte chiffré qui n'ont pas beaucoup d'effet sur la probabilité d'exactitude du décryptage, ce qui réduit les paramètres.

La fonction Compress_q est également utilisée à un autre endroit où son objectif intuitif n'est pas de "compresser". A un certain endroit de la procédure de décryptage (algorithme 3), la fonction est utilisée pour décrypter vers un 1 si $v - s^T u$ est plus proche de $\lceil q/2 \rceil$ que de 0, et décrypter vers un 0 dans le cas contraire.

Soit k, dt, du, dv des paramètres entiers positifs, et rappelons que $n = 256$.

Soit $M = \{0, 1\}^{256}$ l'espace des messages, où chaque message $m \in M$ peut être considéré comme un polynôme dans R avec des coefficients dans $0, 1$.

Considérons le schéma de chiffrement à clé publique $Kyber.CPA = (KeyGen, Enc, Dec)$ tel que décrit dans les algorithmes 1 à 3 ci-après. Notons que les cipheretxts sont de la forme :

$$(u, v) \in \{0, 1\}^{256-kd_u} \times \{0, 1\}^{256-d_v}$$

Algorithm 1 Kyber.CPA.KeyGen(): key generation

- 1: $\rho, \sigma \leftarrow \{0, 1\}^{256}$
 - 2: $\mathbf{A} \sim R_q^{k \times k} := \text{Sam}(\rho)$
 - 3: $(\mathbf{s}, \mathbf{e}) \sim \beta_\eta^k \times \beta_\eta^k := \text{Sam}(\sigma)$
 - 4: $\mathbf{t} := \text{Compress}_q(\mathbf{A}\mathbf{s} + \mathbf{e}, d_t)$
 - 5: **return** $(pk := (\mathbf{t}, \rho), sk := \mathbf{s})$
-

Pour générer une paire de clés, l'utilisateur génère d'abord une clé secrète, qui est un petit polynôme dans l'anneau R_q . Ensuite, l'utilisateur génère une clé publique en multipliant la clé secrète par un terme d'erreur LWE et en ajoutant un élément aléatoire uniforme de R_q .

Algorithm 2 Kyber.CPA.Enc($pk = (\mathbf{t}, \rho), m \in \mathcal{M}$): encryption

- 1: $r \leftarrow \{0, 1\}^{256}$
 - 2: $\mathbf{t} := \text{Decompress}_q(\mathbf{t}, d_t)$
 - 3: $\mathbf{A} \sim R_q^{k \times k} := \text{Sam}(\rho)$
 - 4: $(\mathbf{r}, \mathbf{e}_1, e_2) \sim \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \text{Sam}(r)$
 - 5: $\mathbf{u} := \text{Compress}_q(\mathbf{A}^T \mathbf{r} + \mathbf{e}_1, d_u)$
 - 6: $v := \text{Compress}_q(\mathbf{t}^T \mathbf{r} + e_2 + \lceil \frac{q}{2} \rceil \cdot m, d_v)$
 - 7: **return** $c := (\mathbf{u}, v)$
-

Pour encapsuler une clé, l'expéditeur génère d'abord un secret partagé aléatoire, qui sera utilisé comme clé symétrique pour le chiffrement des données. Ensuite, l'expéditeur chiffre le secret partagé en utilisant la clé publique du destinataire et ajoute un terme d'erreur LWE au cryptogramme résultant. Ce processus garantit que le cryptogramme est indiscernable du bruit aléatoire.

Algorithm 3 Kyber.CPA.Dec($sk = \mathbf{s}, c = (\mathbf{u}, v)$): decryption

- 1: $\mathbf{u} := \text{Decompress}_q(\mathbf{u}, d_u)$
 - 2: $v := \text{Decompress}_q(v, d_v)$
 - 3: **return** $\text{Compress}_q(v - \mathbf{s}^T \mathbf{u}, 1)$
-

Pour récupérer le secret partagé, le destinataire utilise sa clé secrète pour déchiffrer la clé encapsulée. En raison des propriétés du problème LWE, le processus de déchiffrement est essentiellement une étape de réduction de bruit, permettant au destinataire de récupérer le secret partagé original avec une probabilité élevée.

Code Based Cryptography

Code Based Cryptography:

Nous allons maintenant nous intéresser aux systèmes cryptographiques basés sur des codes.

Nous ne considérerons que les systèmes cryptographiques dans lesquels la primitive algorithmique utilise un code correcteur d'erreurs C . Cette primitive peut consister à ajouter une erreur à un mot de C ou à calculer un syndrome relativement à une matrice de contrôle de parité de C .

Un peu d'histoire:

Le premier système de cryptage à clé publique basé sur ce principe a été proposé par Robert J. McEliece en 1978.

La clé privée est un code Goppa irréductible binaire aléatoire et la clé publique est une matrice génératrice aléatoire d'une version permutée aléatoirement de ce code.

Le texte chiffré est un mot codé auquel des erreurs ont été ajoutées, et seul le propriétaire de la clé privée (le code Goppa) peut supprimer ces erreurs.

Trois décennies plus tard, quelques ajustements de paramètres ont été nécessaires, mais aucune attaque n'est connue pour représenter une menace sérieuse pour le système, même sur un ordinateur quantique.

En résumé

En résumé:

- compromis entre sécurité et efficacité

En résumé

En résumé:

- compromis entre sécurité et efficacité
- aucune application pratique de la cryptographie basée sur les codes, cela peut être dû en partie à la taille importante de la clé publique.

En résumé:

- compromis entre sécurité et efficacité
- aucune application pratique de la cryptographie basée sur les codes, cela peut être dû en partie à la taille importante de la clé publique.
- le système de cryptage de McEliece présente de nombreuses caractéristiques solides

Le problème de McEliece

Algorithm 2.1 The McEliece PKC

- **System Parameters:** $n, t \in \mathbb{N}$, where $t \ll n$.
- **Key Generation:** Given the parameters n, t generate the following matrices:

$\mathbf{G} : k \times n$ generator matrix of a code \mathcal{G} over \mathbb{F} of dimension k and minimum distance $d \geq 2t + 1$. (A binary irreducible Goppa code in the original proposal.)

$\mathbf{S} : k \times k$ random binary non-singular matrix

$\mathbf{P} : n \times n$ random permutation matrix

Then, compute the $k \times n$ matrix $\mathbf{G}^{\text{pub}} = \mathbf{SGP}$.

- **Public Key:** $(\mathbf{G}^{\text{pub}}, t)$
- **Private Key:** $(\mathbf{S}, D_{\mathcal{G}}, \mathbf{P})$, where $D_{\mathcal{G}}$ is an efficient decoding algorithm for \mathcal{G} .
- **Encryption ($E_{(\mathbf{G}^{\text{pub}}, t)}$):** To encrypt a plaintext $\mathbf{m} \in \mathbb{F}^k$ choose a vector $\mathbf{z} \in \mathbb{F}^n$ of weight t randomly and compute the ciphertext \mathbf{c} as follows:

$$\mathbf{c} = \mathbf{m}\mathbf{G}^{\text{pub}} \oplus \mathbf{z}.$$

- **Decryption ($D_{(\mathbf{S}, D_{\mathcal{G}}, \mathbf{P})}$):** To decrypt a ciphertext \mathbf{c} calculate

$$\mathbf{c}\mathbf{P}^{-1} = (\mathbf{m}\mathbf{S})\mathbf{G} \oplus \mathbf{z}\mathbf{P}^{-1}$$

first, and apply the decoding algorithm $D_{\mathbf{G}^{\text{pub}}}$ for \mathcal{G} to it. Since $\mathbf{c}\mathbf{P}^{-1}$ has a hamming distance of t to \mathcal{G} we obtain the codeword

$$\mathbf{m}\mathbf{S}\mathbf{G} = D_{\mathcal{G}}(\mathbf{c}\mathbf{P}^{-1}).$$

Let $J \subseteq \{1, \dots, n\}$ be a set, such that $\mathbf{G}_{\cdot J}$ is invertible, then we can compute the plaintext $\mathbf{m} = (\mathbf{m}\mathbf{S}\mathbf{G})_J (\mathbf{G}_{\cdot J})^{-1} S^{-1}$

Formulation mathématique du problème de McEliece :

Soit $F = \{0, 1\}$ et G un code Goppa irréductible binaire selon l'algorithme précédent.

- Etant donné une clé publique de McEliece (G^{pub}, t) où $G^{pub} \in \{0, 1\}^{k \times n}$ et un texte chiffré $c \in \{0, 1\}^n$,
- Trouvez le message (unique) $m \in \{0, 1\}^k$ s.t. $\text{wt } mG^{pub} - c = t$.

Nous ne pouvons pas supposer que le problème de McEliece est NP-difficile. La résolution du problème de McEliece ne résoudrait le problème général de décodage que pour une certaine classe de codes et non pour tous les codes.

La variante de Niederreiter

Formulation mathématique de la variante de Niederreiter :

La variante double du PKC de McEliece est un cryptosystème de type knapsack et est appelée PKC de Niederreiter. Contrairement au cryptosystème de McEliece, au lieu de représenter le message sous la forme d'un mot codé, Niederreiter a proposé de l'encoder dans le vecteur d'erreur par une fonction $\phi_{n,t}$:

$$\phi_{n,t} : \{0, 1\}^l \rightarrow W_{n,t} \quad (1)$$

où $W_{n,t} = \{e \in F_2^n \mid \text{wt}(e) = t\}$ et $l = \lfloor \log_2 |W_{n,t}| \rfloor$.

Formulation mathématique de la variante de Niederreiter :

La variante double du PKC de McEliece est un cryptosystème de type knapsack et est appelée PKC de Niederreiter. Contrairement au cryptosystème de McEliece, au lieu de représenter le message sous la forme d'un mot codé, Niederreiter a proposé de l'encoder dans le vecteur d'erreur par une fonction $\phi_{n,t}$:

$$\phi_{n,t} : \{0, 1\}^l \rightarrow W_{n,t} \quad (1)$$

où $W_{n,t} = \{e \in F_2^n \mid \text{wt}(e) = t\}$ et $l = \lfloor \log_2 |W_{n,t}| \rfloor$.

Une telle correspondance est résumée dans l'algorithme suivant.

Algorithme

Algorithm 2.2 $\phi_{n,t}$: Mapping bit strings to constant weight codewords

Input: $\mathbf{x} \in \{0,1\}^\ell$

Output: a word $\mathbf{e} = (e_1, e_2, \dots, e_n)$ of weight w and length n .

$$c \leftarrow \binom{n}{w}, c' \leftarrow 0, j \leftarrow n.$$

$i \leftarrow$ Index of \mathbf{x} in the lexicographic order (an integer).

while $j > 0$ **do**

$$c' \leftarrow c \cdot \frac{j-w}{j}$$

if $i \leq c'$ **then**

$$e_j \leftarrow 0, c \leftarrow c'$$

else

$$e_j \leftarrow 1, i \leftarrow i - c', c \leftarrow c \cdot \frac{w}{n}$$

$$j \leftarrow j - 1$$

Cet algorithme est assez inefficace et a une complexité $O(n^2 \cdot \log_2 n)$. Son inverse est facile à définir :

$$\phi_{n,t}^{-1}(e) = \sum_{i=1}^n e_i \cdot \binom{i}{\sum_{j=0}^i e_j}$$

Dans l'algorithme ci-dessous nous représentons le message par le vecteur d'erreur, nous obtenons la variante duale du cryptosystème de McEliece. La sécurité du PKC de Niederreiter et celle du PKC de McEliece sont équivalentes. Un attaquant capable de casser l'un est capable de casser l'autre et vice versa.

Algorithm 2.3 Niederreiter's PKC

- **System Parameters:** $n, t \in \mathbb{N}$, where $t \ll n$.
- **Key Generation:** Given the parameters n, t generate the following matrices:

H : $(n - k) \times n$ check matrix of a code \mathcal{G} which can correct up to t errors.

P : $n \times n$ random permutation matrix

Then, compute the systematic $n \times (n - k)$ matrix $H^{\text{pub}} = MHP$, whose columns span the column space of HP , i.e. $H_{\{1, \dots, n-k\}}^{\text{pub}} = \text{Id}_{(n-k)}$.

- **Public Key:** (H^{pub}, t)
- **Private Key:** (P, D_g, M) , where D_g is an efficient syndrome decoding algorithm¹ for \mathcal{G} .
- **Encryption:** A message \mathbf{m} is represented as a vector $\mathbf{e} \in \{0, 1\}^n$ of weight t , called plaintext. To encrypt it, we compute the syndrome

$$\mathbf{s} = H^{\text{pub}} \mathbf{e}^\top.$$

- **Decryption:** To decrypt a ciphertext \mathbf{s} calculate

$$M^{-1}\mathbf{s} = HPe^\top$$

first, and apply the syndrome decoding algorithm D_g for \mathcal{G} to it in order to recover $P\mathbf{e}^\top$. Now, we can obtain the plaintext $\mathbf{e}^\top = P^{-1}P\mathbf{e}^\top$

¹ A syndrome decoding algorithm takes as input a syndrome – not a codeword. Each syndrome decoding algorithm leads immediately to an decoding algorithm and vice versa.

Avantage

- taille réduite de la clé publique

Avantage et inconvénient

Avantage

- taille réduite de la clé publique

Inconvénient

- le mappage $\phi_{n,t}$ ralentit l'encryptage et le décryptage

Remarque : Dans un contexte où nous voulons envoyer uniquement des chaînes aléatoires, l'inconvénient disparaît car nous pouvons prendre $h(e)$ comme chaîne aléatoire, où h est une fonction de hachage sécurisée.

Modifications de la trappe du PKC de McEliece

Modifications :

À partir du schéma de McEliece, on peut facilement dériver un schéma avec une trappe différente en remplaçant simplement les codes binaires irréductibles de Goppa par une autre classe de codes.

Toutefois, ces tentatives se sont souvent révélées vulnérables aux attaques structurelles.

Pour prévenir les attaques structurelles, il existe non seulement la proposition de McEliece, mais aussi d'autres solutions. L'énumération suivante donne un aperçu des principales modifications.

Propositions de McEliece pour y remédier :

- **Brouilleur de lignes** : Multiplier G avec une matrice inversible aléatoire $S \in F^{k \times k}$ à partir de la droite. Comme $G = \langle SG \rangle$, on peut utiliser l'algorithme de correction d'erreur connu.

Propositions de McEliece pour y remédier :

- **Brouilleur de lignes** : Multiplier G avec une matrice inversible aléatoire $S \in F^{k \times k}$ à partir de la droite. Comme $G = \langle SG \rangle$, on peut utiliser l'algorithme de correction d'erreur connu.
- **Brouilleur de colonnes / Isométrie** : Multiplier G avec une matrice inversible aléatoire $T \in F^{n \times n}$ à partir de la gauche, où T préserve la norme.

Propositions de McEliece pour y remédier :

- **Brouilleur de lignes** : Multiplier G avec une matrice inversible aléatoire $S \in F^{k \times k}$ à partir de la droite. Comme $G = \langle SG \rangle$, on peut utiliser l'algorithme de correction d'erreur connu.
- **Brouilleur de colonnes / Isométrie** : Multiplier G avec une matrice inversible aléatoire $T \in F^{n \times n}$ à partir de la gauche, où T préserve la norme.
- **Sous-code** : Soit $0 < l < k$. Multiplier G par une matrice aléatoire $S \in F^{l \times k}$ de plein rang à partir de la droite. Comme $\langle SG \rangle \subseteq \langle G \rangle$, l'algorithme de correction d'erreur connu peut être utilisé.

Propositions de McEliece pour y remédier :

- **Brouilleur de lignes** : Multiplier G avec une matrice inversible aléatoire $S \in F^{k \times k}$ à partir de la droite. Comme $G = \langle SG \rangle$, on peut utiliser l'algorithme de correction d'erreur connu.
- **Brouilleur de colonnes / Isométrie** : Multiplier G avec une matrice inversible aléatoire $T \in F^{n \times n}$ à partir de la gauche, où T préserve la norme.
- **Sous-code** : Soit $0 < l < k$. Multiplier G par une matrice aléatoire $S \in F^{l \times k}$ de plein rang à partir de la droite. Comme $\langle SG \rangle \subseteq \langle G \rangle$, l'algorithme de correction d'erreur connu peut être utilisé.
- **Sous-code de sous-champ** : Prendre le sous-code F_{SUB} -sous-champ du code secret pour un sous-champ F_{SUB} de F .

Propositions de McEliece pour y remédier :

- **Brouilleur de lignes** : Multiplier G avec une matrice inversible aléatoire $S \in F^{k \times k}$ à partir de la droite. Comme $G = \langle SG \rangle$, on peut utiliser l'algorithme de correction d'erreur connu.
- **Brouilleur de colonnes / Isométrie** : Multiplier G avec une matrice inversible aléatoire $T \in F^{n \times n}$ à partir de la gauche, où T préserve la norme.
- **Sous-code** : Soit $0 < l < k$. Multiplier G par une matrice aléatoire $S \in F^{l \times k}$ de plein rang à partir de la droite. Comme $\langle SG \rangle \subseteq \langle G \rangle$, l'algorithme de correction d'erreur connu peut être utilisé.
- **Sous-code de sous-champ** : Prendre le sous-code F_{SUB} -sous-champ du code secret pour un sous-champ F_{SUB} de F .
- **Redondance aléatoire** : Ajouter un nombre l de colonnes aléatoires au côté gauche de la matrice G .

Propositions de McEliece pour y remédier :

- **Brouilleur de lignes** : Multiplier G avec une matrice inversible aléatoire $S \in F^{k \times k}$ à partir de la droite. Comme $G = \langle SG \rangle$, on peut utiliser l'algorithme de correction d'erreur connu.
- **Brouilleur de colonnes / Isométrie** : Multiplier G avec une matrice inversible aléatoire $T \in F^{n \times n}$ à partir de la gauche, où T préserve la norme.
- **Sous-code** : Soit $0 < l < k$. Multiplier G par une matrice aléatoire $S \in F^{l \times k}$ de plein rang à partir de la droite. Comme $\langle SG \rangle \subseteq \langle G \rangle$, l'algorithme de correction d'erreur connu peut être utilisé.
- **Sous-code de sous-champ** : Prendre le sous-code F_{SUB} -sous-champ du code secret pour un sous-champ F_{SUB} de F .
- **Redondance aléatoire** : Ajouter un nombre l de colonnes aléatoires au côté gauche de la matrice G .
- **Erreurs artificielles** : On peut choisir de modifier la matrice G à un petit nombre de positions.

Propositions de McEliece pour y remédier :

- **Brouilleur de lignes** : Multiplier G avec une matrice inversible aléatoire $S \in F^{k \times k}$ à partir de la droite. Comme $G = \langle SG \rangle$, on peut utiliser l'algorithme de correction d'erreur connu.
- **Brouilleur de colonnes / Isométrie** : Multiplier G avec une matrice inversible aléatoire $T \in F^{n \times n}$ à partir de la gauche, où T préserve la norme.
- **Sous-code** : Soit $0 < l < k$. Multiplier G par une matrice aléatoire $S \in F^{l \times k}$ de plein rang à partir de la droite. Comme $\langle SG \rangle \subseteq \langle G \rangle$, l'algorithme de correction d'erreur connu peut être utilisé.
- **Sous-code de sous-champ** : Prendre le sous-code F_{SUB} -sous-champ du code secret pour un sous-champ F_{SUB} de F .
- **Redondance aléatoire** : Ajouter un nombre l de colonnes aléatoires au côté gauche de la matrice G .
- **Erreurs artificielles** : On peut choisir de modifier la matrice G à un petit nombre de positions.
- **Codes réductibles** : Choisir quelques matrices $Y \in F^{k \times k}$ et $S \in F^{l \times k}$ avec $l \leq k$.

Déchiffrement

Déchiffrement :

Le décodage des ensembles d'informations est sans aucun doute la technique qui a le plus attiré l'attention des cryptographes. Les attaques de décodage les plus connues contre McEliece et Niederreiter en sont toutes dérivées.

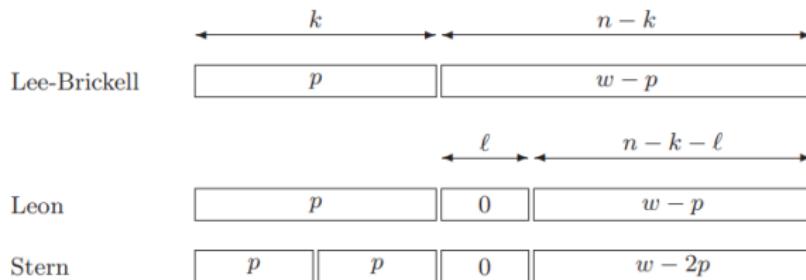
L'algorithme suivant présente une version généralisée du décodage des ensembles d'informations. Lee et Brickell ont été les premiers à l'utiliser pour analyser la sécurité du PKC de McEliece.

Algorithm 3.1 Information set decoding (for parameter p)

- **Input:** a $k \times n$ matrix G , an integer w
 - **Output:** a non-zero codeword of weight $\leq w$
 - **Repeat**
 - Pick a $n \times n$ permutation matrix P .
 - Compute $G' = UGP = (\text{Id} \mid R)$ (w.l.o.g. we assume the first k positions form an information set).
 - Compute all the sum of p rows or less of G' , if one of those sums has weight $\leq w$ then stop and return it.
-

Déchiffrement (2/3)

Dans un autre contexte, le calcul de la distance minimale d'un code, Leon a proposé une amélioration en recherchant les mots de code contenant des zéros dans une fenêtre de taille dans la partie redondante (droite) du mot de code. Stern l'a encore optimisée en divisant l'ensemble d'informations en deux parties, ce qui a permis d'accélérer la recherche de mots de code contenant des zéros dans la fenêtre grâce à une technique d'attaque par anniversaire.



Déchiffrement (3/3)

Dans la figure précédente, nous présentons les différents profils de poids correspondant à un succès, la probabilité de succès d'une itération donnée est respectivement

$$P_{LB} = \frac{\binom{k}{p} \binom{n-k}{w-p}}{\binom{n}{w}}, P_L = \frac{\binom{k}{p} \binom{n-k-1}{w-p}}{\binom{n}{w}}, P_S = \frac{\left(\frac{k}{2}\right)^2 \binom{n-k-1}{w-2p}}{\binom{n}{w}} \quad (2)$$

Le coût total de l'algorithme est généralement exprimé sous la forme d'un facteur de travail binaire. Il est égal au coût (en opération binaire) d'une itération divisée par la probabilité ci-dessus (c'est-à-dire multiplié par le nombre attendu d'itérations).

Hash Based Cryptography

Hash Based Cryptography :

Les systèmes de signature numérique basés sur le hachage offrent une alternative très intéressante aux systèmes de signature numérique traditionnels tels que RSA ou DSA.

L'essentiel :

- ils utilisent une fonction de hachage cryptographique

Hash Based Cryptography :

Les systèmes de signature numérique basés sur le hachage offrent une alternative très intéressante aux systèmes de signature numérique traditionnels tels que RSA ou DSA.

L'essentiel :

- ils utilisent une fonction de hachage cryptographique
- leur sécurité repose sur la résistance aux collisions de cette fonction de hachage

Propriété

Un système de signature numérique basé sur le hachage est sûr, si et seulement si, la fonction de hachage sous-jacente est résistante aux collisions.

Exigence minimale : pour un système de signature numérique capable de signer de nombreux documents avec une seule clé privée, il faut que fonction de hachage soit résistante aux collisions.

Définition

Une collision se produit lorsqu'il est possible de construire deux documents avec la même signature numérique.

Si collisions → le système de signature ne peut plus être considéré comme sûr.

Chaque nouvelle fonction de hachage cryptographique produit un nouveau schéma de signature.

La construction de systèmes de signature sécurisés est donc indépendante des problèmes algorithmiques difficiles de la théorie des nombres ou de l'algèbre. Des constructions issues de la cryptographie symétrique suffisent.

Système de signature numérique :

Un système de signature numérique est capable de signer plusieurs documents à l'aide d'une seule clé privée.

Cela implique :

- des exigences de sécurité minimale

Système de signature numérique :

Un système de signature numérique est capable de signer plusieurs documents à l'aide d'une seule clé privée.

Cela implique :

- des exigences de sécurité minimale
- La construction de systèmes de signature sécurisés est donc indépendante des problèmes algorithmiques difficiles de la théorie des nombres ou de l'algèbre.

Système de signature numérique :

Avantage :

Système de signature numérique :

Avantage :

- la fonction de hachage sous-jacente peut être choisie en fonction des ressources matérielles et logicielles disponibles

Exemple :

Si le système de signature doit être mis en œuvre sur une puce qui implémente déjà AES, une fonction de hachage basée sur AES peut être utilisée, réduisant ainsi la taille du code du système de signature et optimisant son temps d'exécution.

Système de signature numérique :

Avantage :

- la fonction de hachage sous-jacente peut être choisie en fonction des ressources matérielles et logicielles disponibles

Exemple :

Si le système de signature doit être mis en œuvre sur une puce qui implémente déjà AES, une fonction de hachage basée sur AES peut être utilisée, réduisant ainsi la taille du code du système de signature et optimisant son temps d'exécution.

Inconvénient :

- une paire de clés, composée d'une clé de signature secrète et d'une clé de vérification publique, ne peut être utilisée que pour signer et vérifier un seul document.

Système de signature numérique :

Idée : utiliser un arbre de hachage qui réduit la validité de nombreuses clés de vérification unique (les feuilles de l'arbre de hachage) à la validité d'une clé publique (la racine de l'arbre de hachage).

Système de signature numérique :

Idée : utiliser un arbre de hachage qui réduit la validité de nombreuses clés de vérification unique (les feuilles de l'arbre de hachage) à la validité d'une clé publique (la racine de l'arbre de hachage).

Remarque : Les systèmes de signature basés sur le hachage doivent pouvoir résister aux collisions mais aussi au préimage.

Définition : Préimage

Une attaque de préimage est une attaque sur une fonction de hachage cryptographique qui essaie de trouver un message qui a une valeur spécifique de hachage.

Définition : Préimage

Une attaque de préimage est une attaque sur une fonction de hachage cryptographique qui essaie de trouver un message qui a une valeur spécifique de hachage.

Définition : Attaque de préimage

Dans l'attaque de préimage (classique), pour une valeur de sortie spécifiée, un attaquant tente de trouver une entrée qui produit cette valeur en sortie, c'est-à-dire, pour un y donné, il tente de trouver un x tel que $h(x) = y$.

Définition : Préimage

Une attaque de préimage est une attaque sur une fonction de hachage cryptographique qui essaie de trouver un message qui a une valeur spécifique de hachage.

Définition : Attaque de préimage

Dans l'attaque de préimage (classique), pour une valeur de sortie spécifiée, un attaquant tente de trouver une entrée qui produit cette valeur en sortie, c'est-à-dire, pour un y donné, il tente de trouver un x tel que $h(x) = y$.

Définition : Attaque de second préimage

Dans l'attaque de seconde préimage, l'attaquant tente de trouver une seconde entrée qui a la même valeur de hachage qu'une entrée spécifiée. Pour un x donné, il tente de trouver une deuxième préimage $x' \neq x$ tel que $h(x) \neq h(x')$.

Problème difficile : les arbres de Merkle

Introduction :

En 1979, Ralph Merkle propose d'utiliser un arbre de hachage binaire complet pour réduire la validité d'un nombre arbitraire mais fixe de clés de vérification unique, à la validité d'une seule clé publique, la racine de l'arbre de hachage.

Le schéma de signature de Merkle (MSS) fonctionne avec n'importe quelle fonction de hachage cryptographique et n'importe quel schéma de signature à usage unique.

Pour les besoins de l'explication, nous supposons que $g : \{0, 1\}^* \rightarrow \{0, 1\}^n$ est une fonction de hachage cryptographique. Nous supposons également qu'un schéma de signature à usage unique a été sélectionné.

Génération de paires de clés MSS

Le signataire choisit $H \in \mathbb{N}$, $H \leq 2$. Il génère 2^H paires de clés à usage unique $(X_j, Y_j), 0 \geq j < 2^{H-h}$. X_j est la clé de signature et Y_j la clé de vérification. Il s'agit de chaînes de bits.

Les feuilles de l'arbre de Merkle sont les condensés $g(Y_j)$, $0 \geq j < 2^H$.

Les nœuds internes de l'arbre de Merkle sont calculés selon la règle de construction suivante : un nœud parent est la valeur de hachage de la concaténation de ses enfants de gauche et de droite.

La clé publique du MSS est la racine de l'arbre de Merkle.

La clé privée du MSS est la séquence des $2H$ clés de signature à usage unique. Pour être plus précis, désignons les nœuds de l'arbre de Merkle par $\nu_h[j]$, $0 \geq j < 2^{H-h}$ où $h \in \{0, \dots, H\}$ est la hauteur du nœud. Alors :

$$\nu_h[j] = g(\nu_{h-1}[2j] \parallel \nu_{h-1}[2j + 1]), 1 \leq h \leq H, 0 \leq j < 2^{H-h}$$

Un exemple

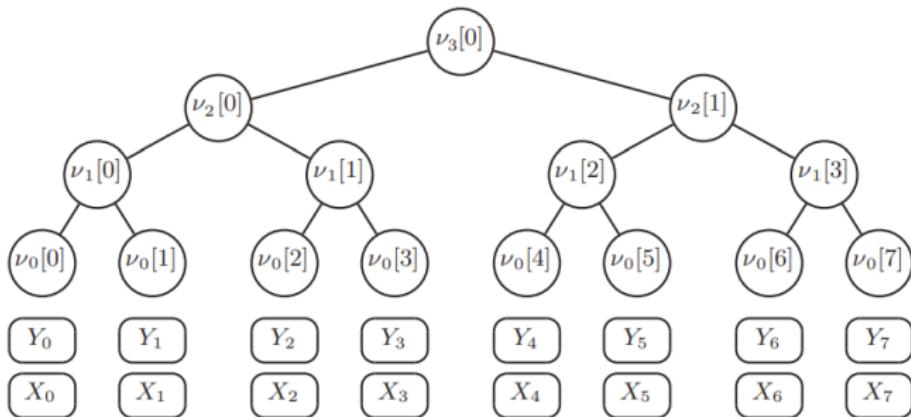


Fig. 1. Un arbre de Merkle de hauteur $H = 3$

Pour signer un message M , le signataire calcule d'abord le condensé de n bits $d = g(M)$. Il génère ensuite la signature unique σ_{OTS} du condensé à l'aide de la s ème clé de signature unique X_s , $s \in \{0, \dots, 2^H - 1\}$.

La signature de Merkle contiendra cette signature à usage unique et la clé de vérification à usage unique correspondante Y_s .

Pour prouver l'authenticité de Y_s au vérificateur, le signataire inclut également l'index s ainsi qu'un chemin d'authentification pour la clé de vérification Y_s qui est une séquence $A_s = (a_0, \dots, a_{H-1})$ de nœuds dans l'arbre de Merkle.

Cet index et le chemin d'authentification permettent au vérificateur de construire un chemin depuis la feuille $g(Y_s)$ jusqu'à la racine de l'arbre de Merkle.

Le nœud h du chemin d'authentification est le frère du nœud de hauteur h sur le chemin allant de la feuille $g(Y_s)$ à la racine de l'arbre de Merkle :

$$a_h = \nu_h[s/2^h - 1], \text{ si } \lceil s/2^h \rceil \equiv 1 \pmod{2}$$

ou

$$a_h = \nu_h[s/2^h + 1], \text{ si } \lceil s/2^h \rceil \equiv 0 \pmod{2}$$

pour $h = 0, \dots, H - 1$ La s -ième signature de Merkle est donc :

$$\sigma_s = (s, \sigma_{OTS}, Y_s, (a_0, \dots, a_{H-1}))$$

Exemple

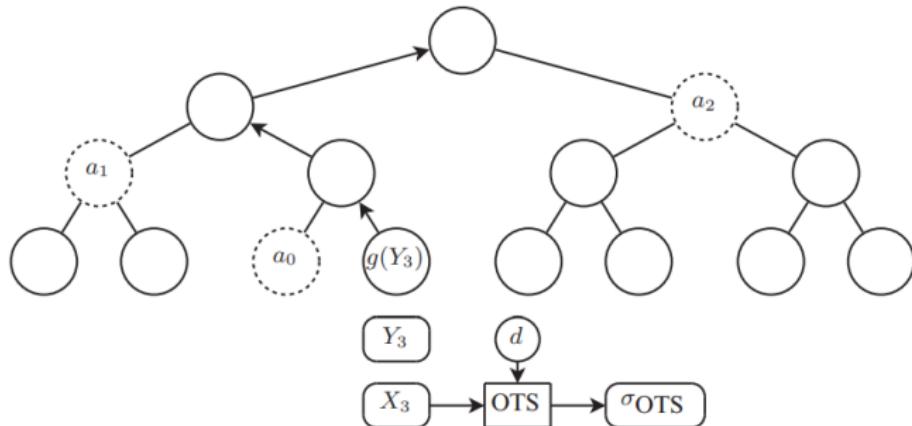


Fig. 2. Génération de signature Merkle pour $s = 3$. Les nœuds en pointillés indiquent le chemin d'authentification de la feuille $g(Y_3)$. Les flèches indiquent le chemin de la feuille $g(Y_3)$ à la racine.

MSS signature verification

La vérification de la signature de Merkle décrite dans la section précédente se fait en deux étapes :

La vérification de la signature de Merkle décrite dans la section précédente se fait en deux étapes :

- Première étape : le vérificateur utilise la clé de vérification unique Y_s pour vérifier la signature unique σ_{OTS} du condensé d au moyen de l'algorithme de vérification du système de signature unique correspondant.

La vérification de la signature de Merkle décrite dans la section précédente se fait en deux étapes :

- Première étape : le vérificateur utilise la clé de vérification unique Y_s pour vérifier la signature unique σ_{OTS} du condensé d au moyen de l'algorithme de vérification du système de signature unique correspondant.
- Seconde étape : le vérificateur valide l'authenticité de la clé de vérification unique Y_s en construisant le chemin (p_0, \dots, p_H) de la énième feuille $g(Y_s)$ à la racine de l'arbre de Merkle.

MSS signature verification

Il utilise l'index s et le chemin d'authentification (a_0, \dots, a_{H-1}) et applique la construction suivante :

$$p_h = g(a_{h-1} \parallel p_{h-1}), \text{ si } \lceil s/2^{h-1} \rceil \equiv 1 \pmod{2}$$

ou

$$p_h = g(p_{h-1} \parallel a_{h-1}), \text{ si } \lceil s/2^h \rceil \equiv 0 \pmod{2}$$

pour $h = 1, \dots, H$ et $p_0 = g(Y_s)$.

L'indice s est utilisé pour décider dans quel ordre les nœuds du chemin d'authentification et les nœuds sur le chemin de la feuille $g(Y_s)$ à la racine de l'arbre de Merkle doivent être concaténés. L'authentification de la clé de vérification unique Y_s est réussie si et seulement si p_H est égal à la clé publique

Cryptosystème : SPHINCS

SPHINCS :

Il introduit deux nouvelles idées qui, ensemble, réduisent considérablement la taille des signatures :

SPHINCS :

Il introduit deux nouvelles idées qui, ensemble, réduisent considérablement la taille des signatures :

- SPHINCS remplace l'OTS de la feuille par un schéma de signature en quelque temps basé sur le hachage (FTS).

SPHINCS :

Il introduit deux nouvelles idées qui, ensemble, réduisent considérablement la taille des signatures :

- SPHINCS remplace l'OTS de la feuille par un schéma de signature en quelque temps basé sur le hachage (FTS).
- SPHINCS considère la construction de Goldreich comme une construction d'hyper-arbre avec h couches d'arbres de hauteur 1, et la généralise à un hyper-arbre avec d couches d'arbres de hauteur h/d .

Few Time Signature :

Un FTS est un schéma de signature conçu pour **signer quelques messages**.

Dans le contexte de SPHINCS, cela permet quelques **collisions d'index**, ce qui permet à son tour de **réduire la hauteur de l'arbre** pour le même niveau de sécurité.

Pour notre FTS, la probabilité d'une falsification après γ signatures augmente progressivement avec γ , tandis que la probabilité que le signataire utilise la même clé FTS γ fois diminue progressivement avec γ .

Nous choisissons des paramètres pour nous assurer que le produit de ces probabilités est suffisamment petit pour tous les $\gamma \in \mathbb{N}$.

Par exemple, SPHINCS-256 réduit la hauteur totale de l'arbre de 256 à seulement 60 tout en conservant une sécurité de 2^{128} contre les attaques sur ordinateurs quantiques.

Construction de Goldreich :

Cette construction introduit un compromis entre la taille de la signature et le temps contrôlé par le nombre de couches d .

La taille de la signature est $|\sigma| \approx d |\sigma_{OTS}| + hn$ en supposant une fonction de hachage avec des sorties de n -bits.

Construction de SPHINX :

SPHINCS utilise un hyper-arbre (un arbre d'arbres) d'une hauteur totale $h \in \mathbb{N}$, où h est un multiple de d et l'hyper-arbre consiste en d couches d'arbres, chacune ayant une hauteur h/d .

Les composants de SPHINCS ont des paramètres supplémentaires qui influencent les performances et la taille de la signature et des clés :

- la signature unique de Winternitz WOTS permet naturellement un compromis espace-temps en utilisant le paramètre de Winternitz $w \in \mathbb{N}$, $w > 1$;

Construction de SPHINX :

SPHINCS utilise un hyper-arbre (un arbre d'arbres) d'une hauteur totale $h \in \mathbb{N}$, où h est un multiple de d et l'hyper-arbre consiste en d couches d'arbres, chacune ayant une hauteur h/d .

Les composants de SPHINCS ont des paramètres supplémentaires qui influencent les performances et la taille de la signature et des clés :

- la signature unique de Winternitz WOTS permet naturellement un compromis espace-temps en utilisant le paramètre de Winternitz $w \in \mathbb{N}$, $w > 1$;
- le schéma de signature à plusieurs temps basé sur des arbres HORST a un compromis espace-temps qui est contrôlé par deux paramètres $k \in \mathbb{N}$ et $t = 2^\tau$ avec $\tau \in \mathbb{N}$ et $k = m$.

WOTS :

Nous décrivons maintenant la signature unique de Winternitz (WOTS+). Étant donné $n \in \mathbb{N}$ et $w \in \mathbb{N}$, nous définissons :

$$l_1 = \lceil \frac{n}{\log(w)} \rceil,$$

$$l_1 = \lceil \frac{\log(l_1(w - 1))}{\log(w)} + 1 \rceil,$$

$$l_1 + l_2$$

Génération de clés :

$(sk, pk \leftarrow WOTS.kg(S, r))$: Sur la base de la graine $S \in \{0, 1\}^n$ et des masques de bits $r \in \{0, 1\}^n \times (w - 1)$, l'algorithme de génération de clés calcule la clé secrète interne sous la forme $sk = (sk_1, \dots, sk_l) \leftarrow G_l(S)$, c'est-à-dire que la graine de n bits est étendue à des valeurs de n bits. La clé publique pk est calculée comme suit :

$$pk = (pk_1, \dots, pk_l) = (c^{w-1}(sk_1, r), \dots, c^{w-1}(sk_l, r))$$

Signature :

$(\sigma \leftarrow WOTS.sign(M, S, r))$: À partir d'un message M de n bits, d'une semence S et des masques de bits r , l'algorithme de signature calcule d'abord une représentation de base w de M :

$$M = (M_1, \dots, M_l), M_i \in \{0, \dots, w - 1\}$$

En d'autres termes, M est traité comme la représentation binaire d'un nombre naturel x , puis la représentation w -aire de x est calculée.

Il calcule ensuite la somme de contrôle $C = \sum_{i=1}^{l_1} (w - 1 - M_i)$ et sa représentation de base $wC = (C_1, \dots, C_{l_2})$. La longueur de la représentation de la base w de C est au plus égale à l_2 puisque $C_1(w - 1)$.

Nous définissons $B = (b_1, \dots, b_l) = M \parallel C$, la concaténation des représentations de la base w de M et de C .

La **clé secrète interne** est ensuite générée à l'aide de $G_l(S)$ de la même manière que lors de la génération des clés. La **signature** est calculée comme suit :

$$\sigma = (\sigma_1, \dots, \sigma_l) = (c^{b_1}(sk_1, r), \dots, c^{b_l}(sk_l, r))$$

Verification :

$(pk' \leftarrow WOTS.vf(M, \sigma, r))$: En entrée d'un message M de n bits, d'une signature σ et de masques de bits r , l'algorithme de vérification calcule d'abord les b_i , $1 \leq i \leq l$ comme décrit ci-dessus. Ensuite, il renvoie :

$$pk' = (pk'_1, \dots, pk'_l) = (c^{w-1-b_1}(\sigma_1, r_{b_1+1, w-1}), \dots, c^{w-1-b_l}(\sigma_l, r_{b_{l+1}, w-1}))$$

HORST :

HORST signe des messages de longueur m et utilise les paramètres k et $t = 2$ avec $k = m$ (les valeurs typiques utilisées dans SPHINCS-256 sont $t = 2^{16}$, $k = 32$).

HORST améliore HORS en utilisant un arbre de hachage binaire pour réduire la taille de la clé publique de tn bits à n bits et la taille combinée de la signature et de la clé publique de tn bits à $(k(\tau - x + 1) + 2^x)n$ bits pour un certain $x \in \mathbb{N} \setminus \{0\}$.

La valeur x est déterminée en fonction de t et de k de telle sorte que $k(\tau - x + 1) + 2^x$ soit minimale.

Il peut arriver que l'expression prenne son minimum pour deux valeurs successives. Dans ce cas, la plus grande valeur est utilisée. Pour SPHINCS-256, cela donne $x = 6$.

Contrairement à un système de signature unique comme WOTS, HORST peut être utilisé pour signer plus d'un message avec la même paire de clés. Cependant, la sécurité diminue avec chaque signature. Comme pour $WOTS^+$, notre description inclut la génération de clés pseudo-aléatoires.

Génération de clés :

$(pk \leftarrow HORST.kg(S, Q))$: Sur l'entrée de la graine $S \in \{0,1\}^n$ et des masques de bits $Q \in \{0,1\}^{2n \times \log t}$, l'algorithme de génération de clés calcule d'abord la clé secrète interne $sk = (sk_1, \dots, sk_t) \leftarrow G_t(S)$.

Les feuilles de l'arbre sont calculées comme $L_i = F(sk_i)$ pour $i[t - 1]$ et l'arbre est construit en utilisant les masques de bits Q . La clé publique pk est calculée comme le nœud racine d'un arbre binaire de hauteur $\log t$.

Signature :

$(\sigma \leftarrow HORST.sign(M, S, r))$: Sur l'entrée d'un message $M \in \{0, 1\}^m$, d'une graine $S \in \{0, 1\}^n$, et de masques de bits $Q \in \{0, 1\}^{2n \times \log t}$, la clé secrète interne sk est d'abord calculée comme décrit ci-dessus. Ensuite, $M = (M_0, \dots, M_{k-1})$ désigne les k nombres obtenus en divisant M en k chaînes de longueur $\log t$ bits chacune et en interprétant chacune d'elles comme un entier non signé.

La signature $\sigma = (\sigma_0, \dots, \sigma_{k-1}, \sigma k)$ se compose de k blocs $= (sk_{Mi}, Auth_{Mi})$ pour $i \in [k - 1]$ contenant la M_i ième clé secrète et les $\tau - x$ éléments inférieurs du chemin d'authentification de la feuille correspondante $(A_0, \dots, A_{\tau-1-x})$.

Le bloc σ_k contient tous les 2^x nœuds de l'arbre binaire au niveau $\tau - x$ $(N_{0,\tau-x}, \dots, N_{2^x-1,\tau-x})$. En plus de la signature, $HORST.sign$ produit également la clé publique.

Verification :

$(pk' \leftarrow HORST.vf(M, \sigma, Q))$: Sur l'entrée d'un message $M \in \{0, 1\}^m$, d'une signature σ , et de masques de bits $Q \in \{0, 1\}^{2n \times \log t}$, l'algorithme de vérification calcule d'abord le M_i , comme décrit ci-dessus. Ensuite, pour $i \in [k - 1]$, $y_i = \lfloor M_i / 2^\tau - x \rfloor$, il calcule $N'_{y_i, \tau-x}$ en utilisant l'algorithme 1 avec l'indice M_i , $L_{M_i} = F(\sigma_i^1)$, et $Auth_{M_i} = \sigma_i^2$. Il vérifie ensuite que $\forall i \in [k - 1] : N'_{y_i, \tau-x} = Ny_i, \tau - x'$, c'est-à-dire que les nœuds calculés correspondent à ceux de σ_k .

Si toutes les comparaisons sont valables, il utilise σ_k pour calculer et renvoie $Root_0$, sinon il renvoie *fail*.

SPHINCS :

SPHINCS travaille sur un hyper-arbre de hauteur h qui consiste en d couches d'arbres de hauteur h/d . Chacun de ces arbres se présente comme suit :

Les feuilles d'un arbre sont des nœuds racine de $2^{h/d}$ $L - Trees$ qui compriment chacun la clé publique d'une paire de clés $WOTS^+$. Par conséquent, un arbre peut être considéré comme une paire de clés qui peut être utilisée pour signer $2^{h/d}$ messages.

L'hyper-arbre :

L'hyper-arbre est structuré en d couches. La couche $d - 1$ comporte un seul arbre. Sur la couche $d - 2$, il y a $2^{h/d}$ arbres. Les racines de ces arbres sont signées à l'aide des paires de clés WOTS+ de l'arbre de la couche $d - 1$.

En général, la couche i se compose de $2^{(d-1-i)(h/d)}$ arbres et les racines de ces arbres sont signées à l'aide des paires de clés WOTS+ des arbres de la couche $i - 1$.

Enfin, sur la couche 0, chaque paire de clés $WOTS^+$ est utilisée pour signer une clé publique HORST. Nous parlons d'une structure "virtuelle" car toutes les valeurs qu'elle contient sont déterminées en choisissant une graine et les masques de bits, et parce que la structure complète n'est jamais calculée. La graine fait partie de la clé secrète et est utilisée pour la génération de clés pseudo-aléatoires

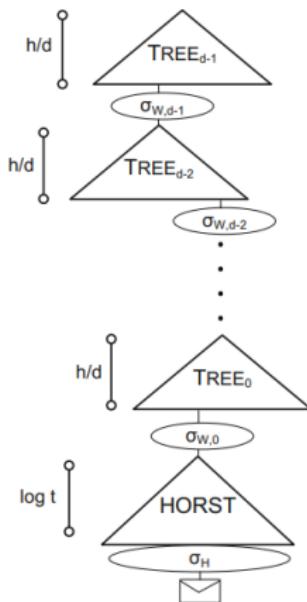


Fig. 3. Structure virtuelle d'une signature SPHINCS

Vous pouvez voir ci-dessus un schéma montrant la structure virtuelle d'une signature SPHINCS, c'est-à-dire d'un chemin à l'intérieur de l'hyper-arbre.

Elle contient d arbres $TREE_i; i \in [d - 1]$ (chacun consistant en un arbre de hachage binaire qui authentifie les nœuds racine de $2^{h/d} L - Trees$ qui, à leur tour, ont chacun les nœuds de clé publique d'une paire de clés $WOTS^+$ comme feuilles).

Chaque arbre authentifie l'arbre inférieur à l'aide d'une signature $WOTS^+ \sigma_{W,i}$. La seule exception est l'arbre $TREE_0$ qui authentifie une clé publique HORST à l'aide d'une signature $WOTS^+$.

Enfin, la paire de clés HORST est utilisée pour signer le message. Les arbres utilisés dans l'hyperarbre (qui déterminent à leur tour les paires de clés σ_H utilisées pour la signature) et la paire de clés HORST sont déterminés par l'index généré de manière pseudo-aléatoire qui n'est pas montré ici.

Nous utilisons un schéma d'adressage simple pour la génération de clés pseudo-aléatoires.
Une adresse est une chaîne de bits de longueur :

$$a = \lceil \log(d + 1) \rceil + (d - 1)(h/d) + (h/d) = \lceil \log(d + 1) \rceil + h$$

L'adresse d'une paire de clés $WOTS^+$ est obtenue en codant la couche de l'arbre à laquelle elle appartient sous la forme d'une chaîne de $\log(d + 1)$ -bits (en utilisant $d + 1$ pour la couche supérieure avec un seul arbre).

Ensuite, on ajoute l'index de l'arbre dans la couche codée sous la forme d'une chaîne de $(d - 1)(h/d)$ -bits (nous numérotions les arbres de gauche à droite, en commençant par 0 pour l'arbre le plus à gauche).

Enfin, l'index de la paire de clés $WOTS^+$ dans l'arbre est ajouté et codé sous la forme d'une chaîne de (h/d) -bits (la numérotation se fait également de gauche à droite, en commençant par 0).

L'adresse de la paire de clés HORST est obtenue en utilisant l'adresse de la paire de clés $WOTS^+$ utilisée pour signer sa clé publique et en plaçant d comme valeur de couche dans la chaîne d'adresse, codée sous forme de chaîne de bits $\lceil \log(d + 1) \rceil$. A titre d'exemple, en SPHINCS-256, une adresse nécessite 64 bits.

Génération de clés :

$((SK, PK)kg(1^n))$: L'algorithme de génération de clés échantillonne d'abord deux valeurs de clés secrètes $(SK_1, SK_2) \in \{0, 1\}^n \times \{0, 1\}^n$.

La valeur SK_1 est utilisée pour la génération de clés pseudo-aléatoires. La valeur SK_2 est utilisée pour générer un index imprévisible en signe et des valeurs pseudo-aléatoires pour randomiser le hachage du message en signe. En outre, p valeurs uniformément aléatoires de n -bits $Q \leftarrow \{0, 1\}^{p \times n}$ sont échantillonnées en tant que masques de bits où $p = \max\{w - 1, 2(h + \lceil \log l \rceil), 2\log t\}$.

Ces masques de bits sont utilisés pour toutes les instances $WOTS^+$ et HORST ainsi que pour les arbres. Dans ce qui suit, nous utilisons $QWOTS^+$ pour les $w - 1$ premiers masques de bits (de longueur n) dans Q , $QHORTS$ pour le premier $2\log t$, QL -Tree pour le premier $2\lceil \log l \rceil$, et $QTree$ pour les $2h$ chaînes de longueur n dans Q qui suivent $QTree$.

Algorithme de génération de clés

La partie restante du kg consiste à générer le nœud racine de l'arbre sur la couche $d - 1$. À cette fin, les paires de clés $WOTS^+$ pour l'arbre unique de la couche $d - 1$ sont générées. La graine de la paire de clés avec l'adresse $A = (d - 1 \parallel 0 \parallel i)$ où $i \in [2^{h/d} - 1]$ est calculée comme $S_a \leftarrow F_a(A, SK_1)$, en évaluant la PRF sur l'entrée A avec la clé SK_1 .

En général, la graine d'une paire de clés $WOTS^+$ avec l'adresse A est calculée comme $S_a \leftarrow F_a(A, SK_1)$ et nous supposerons à partir de maintenant que ces graines sont connues de tout algorithme qui connaît SK_1 .

La clé publique $WOTS^+$ est calculée comme suit : $pk_a \leftarrow WOTS.kg(S_a, Q_{WOTS^+})$. La i ème feuille L_i de l'arbre est la racine d'un arbre L qui compresse pk_a à l'aide de masques de bits $QL - Tree$. Enfin, un arbre de hachage binaire est construit à l'aide des feuilles construites et son nœud racine devient pk_1 .

La clé secrète SPHINCS est : $SK = (SK_1, SK_2, Q)$, la clé publique est $PK = (PK_1, Q)$. kg renvoie la paire de clés $((SK_1, SK_2, Q), (PK_1, Q))$.

Signature :

$(\Sigma \leftarrow sign(M, SK))$: Sur l'entrée d'un message $M \in \{0, 1\}^*$ et d'une clé secrète $SK = (SK_1, SK_2, Q)$, $sign$ calcule un condensé de message aléatoire $D \in \{0, 1\}^m$:

Tout d'abord, un pseudo-aléatoire $R = (R_1, R_2) \in \{0, 1\}^n \times \{0, 1\}^n$ est calculé comme $R \leftarrow F(M, SK_2)$.

Ensuite, $D \leftarrow H(R_1, M)$ est calculé comme le hachage aléatoire de M en utilisant les n premiers bits de R comme aléatoire. Les n derniers bits de R sont utilisés pour sélectionner une paire de clés HORST, en calculant un index de h bits $i \leftarrow Chop(R_2, h)$ comme les h premiers bits de R_2 .

Algorithme de signature

Il convient de noter que la signature est déterministe, c'est-à-dire que nous n'avons pas besoin d'un véritable hasard, car tout le "hasard" nécessaire est généré de manière pseudo-aléatoire à l'aide de la *PRFF*.

Étant donné l'index i , la paire de clés HORST avec l'adresse

$A_{HORST} = (d \parallel i(0, (d - 1)h/d) \parallel i((d - 1)h/d, h/d))$ est utilisée pour signer le condensé de message D , c'est-à-dire que les premiers $(d - 1)h/d$ bits de i sont utilisés comme index de l'arbre et les bits restants pour l'index à l'intérieur de l'arbre.

La signature HORST et la clé publique $(\sigma_H, pk_H) \leftarrow (D, S_{A_{HORST}}, Q_{HORST})$ sont calculées à l'aide des masques de bits HORST et de la graine

$S_{A_{HORST}} \leftarrow F_a(A_{HORST}, SK_1)$.

Algorithme de signature

La signature SPHINCS $\Sigma = (i, R_1, \sigma_H, \sigma_{W,0}, Auth_{A_0}, \dots, \sigma_{W,d-1}, Auth_{A_{d-1}})$ contient, outre l'indice i , l'aléa R_1 et la signature HORST σ_H , une signature WOTS⁺ et un chemin d'authentification $\sigma_{W,i}, Auth_{A_i}$, $i \in [d - 2]$ par couche. Ils sont calculés comme suit :

La paire de clés WOTS+ avec l'adresse A_0 est utilisée pour signer pk_H , où A_0 est l'adresse obtenue en prenant A_{HORST} et en mettant les premiers $\lceil \log(d + 1) \rceil$ bits à zéro.

Cette opération est réalisée en exécutant $\sigma_{W,1} \leftarrow (pk_H, S_{A_0}, Q_{WOTS^+})$ à l'aide des masques de bits WOTS+. Le chemin d'authentification $Auth_{i((d-1)h/d, h/d)}$ de la paire de clés WOTS+ utilisée est ensuite calculée.

Ensuite, la clé publique WOTS+ $pk_{W,0}$ est calculée en exécutant $pk_{W,0} \leftarrow WOTS.vf(pk_H, \sigma_{W,0}, Q_{WOTS^+})$.

Algorithme de signature

Le nœud racine Root_0 de l'arbre est calculé en compressant d'abord $pk_{W,0}$ à l'aide d'un $L - Tree$. Le premier algorithme est ensuite appliqué en utilisant l'index de la paire de clés $WOTS^+$ dans l'arbre, la racine de l'arbre L et $\text{Auth}_{i((d-1)h/d, h/d)}$.

Cette procédure est répétée pour les couches 1 à $d - 1$ avec les deux différences suivantes:

Sur la couche $1 \leq j < d$, $WOTS^+$ est utilisé pour signer Root_{j-1} , la racine calculée à la fin de l'itération précédente. L'adresse de la paire de clés $WOTS^+$ utilisée sur la couche j est calculée comme $A_j = (j \parallel i(0, (d-1-j)h/d) \parallel i((d-1-j)h/d, h/d))$, c'est-à-dire que sur chaque couche, les derniers (h/d) bits de l'adresse de l'arbre deviennent la nouvelle adresse de la feuille et les bits restants de l'ancienne adresse de l'arbre deviennent la nouvelle adresse de l'arbre.

Enfin, $sign$ a pour sorties : $\Sigma = (i, R_1, \sigma_H, \sigma_W, 0, \text{Auth}_{A_0}, \dots, \sigma_{W,d-1}, \text{Auth}_{A_{d-1}})$.

Verification :

$(b \leftarrow vf(M, \Sigma, PK))$: Avec un message $M \in \{0,1\}^*$, une signature Σ et une clé publique PK , l'algorithme calcule le condensé de message $D \leftarrow H(R_1, M)$ en utilisant l'aléa R_1 contenu dans la signature.

Le condensé de message D et les masques binaires $HORST$ $QHORST$ de PK sont utilisés pour calculer la clé publique $HORST$ $pk_H \leftarrow HORST.vf(D, \sigma_H, QHORST)$ à partir de la signature $HORST$.

Si $HORST.vf$ renvoie un échec, la vérification renvoie faux. La clé publique $HORST$ est à son tour utilisée avec les masques de bits $WOTS+$ et la signature $WOTS+$ pour calculer la première clé publique $WOTS+$ $pk_{W,0} \leftarrow WOTS.vf(pk_H, \sigma_{W,0}, QWOTS^+)$. Un $L - Tree$ est utilisé pour calculer $L_{i((d-1)h/d, h/d)}$, la feuille correspondant à $pk_{W,0}$.

Ensuite, la racine $Root_0$ de l'arbre correspondant est calculée à l'aide de l'algorithme 1 avec l'index $i((d-1)h/d, h/d)$, la feuille $L_{i((d-1)h/d, h/d)}$ et le chemin d'authentification $Auth_0$.

Algorithme de vérification

Cette procédure est ensuite répétée pour les couches 1 à $d - 1$ avec les deux différences suivantes :

- Premièrement, sur la couche $1 \leq j < d$, la racine de l'arbre $\text{Root}\{j - 1$ précédemment traité est utilisée pour calculer la clé publique $\text{WOTS+ } pk_{W,j}$.

Cette procédure est ensuite répétée pour les couches 1 à $d - 1$ avec les deux différences suivantes :

- Premièrement, sur la couche $1 \leq j < d$, la racine de l'arbre $\text{Root}\{j - 1$ précédemment traité est utilisée pour calculer la clé publique $\text{WOTS+ } pk_{W,j}$.
- Deuxièmement, la feuille calculée à partir de $pk_{W,j}$ à l'aide d'un $L - Tree$ est $L_{i((d-1)h/d, h/d)}$, c'est-à-dire que l'indice de la feuille dans l'arbre peut être calculé en coupant les derniers $j(h/d)$ bits de i et en utilisant ensuite les derniers (h/d) bits de la chaîne de bits résultante.

Le résultat de la dernière répétition sur la couche $d - 1$ est une valeur Root_{d-1} pour le nœud racine de l'arbre unique de la couche supérieure. Cette valeur est comparée au premier élément de la clé publique, c'est-à-dire $PK_1 = ? \text{Root}_{d-1}$. Si la comparaison est valable, vf renvoie vrai, sinon il renvoie faux.

Le cryptosystème SPHINCS+

SPHINCS+ :

SPHINCS+ est un **cadre de signature sans état basé sur le hachage**, qui améliore SPHINCS en termes de **vitesse et de taille de signature**. Il introduit plusieurs améliorations qui renforcent la **sécurité du système** et permettent ainsi d'utiliser des **paramètres plus petits**.

Il introduit un **cadre de signature** plutôt qu'un système de signature spécifique. La raison principale en est la grande **flexibilité** offerte par les nombreuses options de paramètres. Cela permet aux utilisateurs de faire des **compromis** très spécifiques à leur application en ce qui concerne la taille de la signature, la vitesse de signature, le nombre requis de signatures et le niveau de sécurité souhaité.

Comme SPHINCS+ ressemble à SPHINCS sur de nombreux points, nous nous abstenons de donner une description détaillée du système complet dans le présent document. Une spécification formelle complète de SPHINCS+ est disponible dans la soumission officielle au NIST.

Multivariate Polynomial Cryptography

Multivariate Polynomial Cryptography :

L'idée de la cryptographie intitulée Multivariate Polynomial Cryptography (MPKC) a vu le jour dans les années 1980 et de nombreux cryptographes de premier plan (Ong, Schnorr, Matsumoto, Imai, Harashima, Diffie, Fell, Miyagawa, Tsujii, Kurosawa, Fujioka et d'autres) ont essayé de construire différents types de MPKC.

Pour un MPKC, la clé publique est donnée par un ensemble de polynômes multivariés non linéaires sur un corps fini. En général, la clé publique repose sur un système de polynômes quadratiques multivariés.

Système de polynômes quadratiques multivariés

Système de polynômes quadratiques multivariés :

$$p^{(1)}(x_1, \dots, x_n) = \sum_{i=1}^n \sum_{j=1}^n p_{ij}^{(1)} \cdot x_i x_j + \sum_{i=1}^n p_i^{(1)} x_i + p_0^{(1)}$$

$$p^{(2)}(x_1, \dots, x_n) = \sum_{i=1}^n \sum_{j=1}^n p_{ij}^{(2)} \cdot x_i x_j + \sum_{i=1}^n p_i^{(2)} x_i + p_0^{(2)}$$

.

.

.

$$p^{(m)}(x_1, \dots, x_n) = \sum_{i=1}^n \sum_{j=1}^n p_{ij}^{(m)} \cdot x_i x_j + \sum_{i=1}^n p_i^{(m)} x_i + p_0^{(m)}$$

Ici, tous les coefficients et variables sont issus de F_q , un corps fini à q éléments.
L'ensemble des polynômes publics

$$P(x_1, \dots, x_n) = (p^{(1)}(x_1, \dots, x_n), \dots, p^{(m)}(x_1, \dots, x_n)),$$

est mathématiquement une carte de F_q^n à F_q^m .

Les opérations publiques de cryptage d'un message ou de vérification d'une signature correspondent à une simple évaluation de $P(x_1, \dots, x_n)$. Le processus de décryptage d'un texte chiffré ou de génération d'une signature correspond à une "inversion" de la carte $P(x_1, \dots, x_n)$.

Le problème MQ

Définition

Étant donné un système P de m polynômes quadratiques multivariés $p^{(1)}(x), \dots, p^{(m)}(x)$ en n variables (x_1, \dots, x_n) comme indiqué ci-dessus, trouver un vecteur $\tilde{x} = (\tilde{x}_1, \dots, \tilde{x}_n)$ tel que

$$P(\tilde{x}_1, \dots, \tilde{x}_n) = (0, \dots, 0),$$

ou

$$p^{(1)}(x) = \dots = p^{(m)}(x) = 0.$$

Il est prouvé que le problème général MQ est NP -difficile sur tout corps fini $GF(q)$.

La preuve est particulièrement simple et directe pour $q = 2$. Les instances les plus difficiles de MQ sont généralement obtenues lorsque m et n sont du même ordre de grandeur (lorsque m est beaucoup plus grand que n , ou lorsque n est beaucoup plus grand que m , certains algorithmes efficaces sont connus).

Il est également intéressant de noter que très souvent, les meilleurs algorithmes connus sur MQ ont une **complexité similaire pour les cas les plus défavorables que pour les cas aléatoires**. Il en va de **même pour les algorithmes quantiques**. À l'heure actuelle, il n'existe aucun algorithme quantique capable de résoudre efficacement ce problème MQ , et puisque MQ est NP -difficile, nous avons des raisons de penser que ce sera encore le cas à l'avenir.

Afin de permettre un décryptage et une génération de signature efficaces, il est nécessaire de construire une trappe secrète pour la carte $P(x_1, \dots, x_n)$.

Pour la construction standard d'un cryptosystème à clé publique multivariée, on choisit un système F de m polynômes quadratiques en n variables qui peuvent être facilement inversés (carte centrale).

Ensuite, on choisit deux cartes affines inversibles S et T pour cacher la structure de la carte centrale F dans la clé publique.

La clé publique du cryptosystème est la carte quadratique composée $P = S \circ F \circ T$ qui est supposée être difficile à inverser.

La clé privée se compose de S , F et T et permet donc d'inverser P .

Les cartes S et T sont utilisées pour protéger la carte F . Le processus standard de cryptage et de décryptage ou de génération et de vérification de signature fonctionne comme suit :

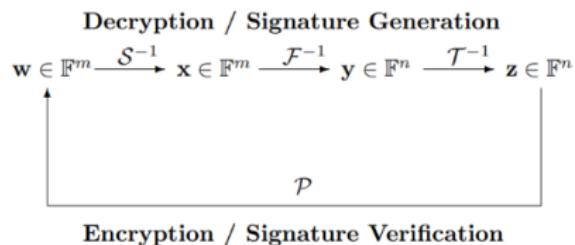


Figure Flux de travail général pour les schémas multivariés

Chiffrement :

Pour chiffrer un message $z \in F^n$, il suffit de calculer $w = P(z)$. Le texte chiffré du message z est wF^m

Décryptage :

Pour déchiffrer le texte chiffré $w \in F^m$, on calcule séquentiellement $x = S^{-1}(w)$, $y = F^{-1}(x)$ et $z = T^{-1}(y)$. $z \in F_n$ est le texte en clair correspondant au texte chiffré w . Puisque $m \leq n$, la préimage de x sous F et donc le texte en clair résultant est unique.

Génération de signatures

Pour signer un document d , nous utilisons une fonction de hachage $H : \{0,1\}^* \rightarrow F^m$ pour calculer la valeur $w = H(d) \in F^m$. Ensuite, nous calculons $x = S^{-1}(h)$, $y = F^{-1}(x)$ et $z = T^{-1}(y)$. La signature du document d est $z \in F^n$. Ici, $F^{-1}(x)$ signifie trouver une (parmi les nombreuses possibles) pré-images de x sous la carte centrale F . Puisque $n \leq m$, nous pouvons être sûrs qu'une telle pré-image existe. Par conséquent, chaque message a une signature.

Vérification

Pour vérifier l'authenticité d'une signature z , il suffit de calculer $w' = P(z)$ et la valeur de hachage $w = H(d)$ du document. Si $w' = w$ est valide, la signature est acceptée, sinon elle est rejetée.

Cryptosystème : Rainbow

Rainbow :

L'histoire du système de signature Rainbow remonte à l'attaque des équations de linéarisation de Patarin contre le cryptosystème Matsumoto-Imai en 1995. Un an plus tard, en 1996, Patarin a dérivé de cette attaque le schéma de signature Oil and Vinegar (OV).

Après que la version équilibrée de ce système a été cassée par une attaque de sous-espace invariant en 1998, Kipnis, Patarin et Goubin ont proposé en 1999 le système de signature Huile et Vinaigre non équilibré (UOV), qui est toujours considéré comme sûr.

Pour améliorer l'efficacité de ce système, Ding et Schmidt ont proposé en 2005 le système de signature Rainbow, qui peut être considéré comme une version multicouche de l'UOV.

Le système de signature Rainbow avec u couches peut être décrit comme suit. Soit F^q un corps fini à q éléments et $v_1 < v_2 < \dots < v_u < v_{u+1} = n$ des nombres entiers.

Nous fixons $V_i = 1, \dots, v_i$, $o_i = v_{i+1} - v_i$ et $O_i = \{v_i, \dots, v_{i+1}\}$ ($i = 1, \dots, u$). On obtient donc $|V_i| = v_i$ et $|O_i| = o_i$ ($i = 1, \dots, u$).

La carte centrale F de Rainbow consiste en $m = n + v_1$ polynômes quadratiques multivariés $f^{(v_1+1)}, \dots, f^{(n)}$ de la forme

$$f^{(k)}(x) = \sum_{i,j \in V_l} \alpha_{ij}^{(k)} x_i x_j + \sum_{i \in V_l, j \in O_l} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V_l O_l} \gamma_i^{(k)} x_i + \eta^{(k)}, \quad (3)$$

où $l \in \{1, \dots, u\}$ est le seul entier tel que $k \in O_l$.

Notons que, dans tout polynôme $f(k)$ avec $k \in O_I$, il n'y a pas de terme quadratique $x_i x_j$ où i et j sont tous deux dans O_I . Ce fait sera utilisé plus tard pour générer la signature.

De tels polynômes ont été appelés polynômes Oil Vinegar lorsque les schémas *OV* ont été proposés.

Pour cacher la structure de F dans la clé publique, on la compose avec deux cartes affines ou linéaires inversibles $S : F^m \rightarrow F^m$ et $T : F^n \rightarrow F^n$.

Ainsi, la clé publique de Rainbow a la forme $P = S \circ F \circ T : F^n \rightarrow F^m$, la clé privée se compose des trois cartes S , F et T et permet donc l'inversion de la carte de la clé publique.

Génération de signatures

Pour générer une signature pour un message (ou une valeur de hachage) $w \in F^m$, il faut suivre les trois étapes suivantes.

Pour générer une signature pour un message (ou une valeur de hachage) $w \in F^m$, il faut suivre les trois étapes suivantes.

- Calculer $x = S^{-1}(w) \in F^m$.

Pour générer une signature pour un message (ou une valeur de hachage) $w \in F^m$, il faut suivre les trois étapes suivantes.

- Calculer $x = S^{-1}(w) \in F^m$.
- Calculer une pré-image y de x sous la carte centrale F en utilisant l'algorithme d'inversion de la carte centrale ci-dessous.

Pour générer une signature pour un message (ou une valeur de hachage) $w \in F^m$, il faut suivre les trois étapes suivantes.

- Calculer $x = S^{-1}(w) \in F^n$.
- Calculer une pré-image y de x sous la carte centrale F en utilisant l'algorithme d'inversion de la carte centrale ci-dessous.
- Calculer la signature $z \in F^n$ par $z = T^{-1}(y)$.

Algorithm Inversion of the Rainbow central map

Input: Rainbow central map $\mathcal{F} = (f^{(v_1+1)}, \dots, f^{(n)})$, vector $\mathbf{x} \in \mathbb{F}^m$.

Output: vector $\mathbf{y} \in \mathbb{F}^n$ with $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

- 1: Choose random values for the variables y_1, \dots, y_{v_1} and substitute these values into the polynomials $f^{(i)}$ ($i = v_1 + 1, \dots, n$).
 - 2: **for** $\ell = 1$ to u **do**
 - 3: Perform Gaussian Elimination on the polynomials $f^{(i)}$ ($i \in O_\ell$) to get the values of the variables y_i ($i \in O_\ell$).
 - 4: Substitute the values of y_i ($i \in O_\ell$) into the polynomials $f^{(i)}$ ($i = v_{\ell+1} + 1, \dots, n$).
 - 5: **end for**
 - 6: **return** $\mathbf{y} = (y_1, \dots, y_n)$
-

Vérification de la signature

Pour vérifier si $z \in F^n$ est une signature valide pour un message $w \in F^n$, il suffit de calculer $w' \in P(z)$. Si $w' = w$ est valide, la signature est acceptée, sinon elle est rejetée.

La présente soumission décrit essentiellement le système de signature Rainbow tel qu'il a été proposé dans l'article original de 2005 avec deux couches de systèmes OV (c.-à-d. $u = 2$). Nous n'utilisons qu'un petit nombre de modifications :

Vérification de la signature

Pour vérifier si $z \in F^n$ est une signature valide pour un message $w \in F^n$, il suffit de calculer $w' \in P(z)$. Si $w' = w$ est valide, la signature est acceptée, sinon elle est rejetée.

La présente soumission décrit essentiellement le système de signature Rainbow tel qu'il a été proposé dans l'article original de 2005 avec deux couches de systèmes OV (c.-à-d. $u = 2$). Nous n'utilisons qu'un petit nombre de modifications :

- En raison de certaines attaques raffinées, nous adaptons légèrement les paramètres de Rainbow.

Pour vérifier si $z \in F^n$ est une signature valide pour un message $w \in F^n$, il suffit de calculer $w' \in P(z)$. Si $w' = w$ est valide, la signature est acceptée, sinon elle est rejetée.

La présente soumission décrit essentiellement le système de signature Rainbow tel qu'il a été proposé dans l'article original de 2005 avec deux couches de systèmes OV (c.-à-d. $u = 2$). Nous n'utilisons qu'un petit nombre de modifications :

- En raison de certaines attaques raffinées, nous adaptons légèrement les paramètres de Rainbow.
- Nous proposons en option une variante de Rainbow avec une taille de clé publique réduite basée sur des techniques, que nous appelons désormais Rainbow circumzenithal (CZ), ainsi qu'une variante compressée qui réduit en outre la taille de la clé privée.

Pour vérifier si $z \in F^n$ est une signature valide pour un message $w \in F^n$, il suffit de calculer $w' \in P(z)$. Si $w' = w$ est valide, la signature est acceptée, sinon elle est rejetée.

La présente soumission décrit essentiellement le système de signature Rainbow tel qu'il a été proposé dans l'article original de 2005 avec deux couches de systèmes OV (c.-à-d. $u = 2$). Nous n'utilisons qu'un petit nombre de modifications :

- En raison de certaines attaques raffinées, nous adaptons légèrement les paramètres de Rainbow.
- Nous proposons en option une variante de Rainbow avec une taille de clé publique réduite basée sur des techniques, que nous appelons désormais Rainbow circumzenithal (CZ), ainsi qu'une variante compressée qui réduit en outre la taille de la clé privée.
- Pour accélérer le processus de génération des clés, nous utilisons une nouvelle technique proposée par Petzoldt

Ce cryptosystème a été **cassé** après sa soumission au NIST.

En effet, une récupération de clés contre l'ensemble des paramètres SL 1 de la soumission du deuxième tour de Rainbow peut être réalisée en pratique par toute personne disposant d'un ordinateur portable décent et d'un peu de patience (ou de chance) et celle du troisième tour de Rainbow devrait être plus coûteuse d'un facteur 2^8 seulement.

Post-Quantum Cryptography on FPGA Based on Isogenies on Elliptic Curves

Introduction :

La cryptographie post-quantique sur FPGA basée sur les isogénies des courbes elliptiques est une autre approche visant à sécuriser les communications contre les attaques des ordinateurs quantiques.

Définition

Les isogénies des courbes elliptiques sont une approche prometteuse. Elles reposent sur l'étude des transformations entre différentes courbes elliptiques, appelées isogénies, qui préservent certaines propriétés algébriques.

La sécurité des systèmes cryptographiques basés sur les isogénies repose sur la difficulté de résoudre le problème du calcul des isogénies entre deux courbes elliptiques, un problème qui est considéré comme difficile même pour les ordinateurs quantiques.

Définition

Le FPGA (Field-Programmable Gate Array) est un dispositif électronique programmable qui permet de mettre en œuvre des circuits logiques personnalisés.

L'implémentation de la cryptographie post-quantique basée sur les isogénies des courbes elliptiques sur FPGA présente plusieurs avantages :

Définition

Le FPGA (Field-Programmable Gate Array) est un dispositif électronique programmable qui permet de mettre en œuvre des circuits logiques personnalisés.

L'implémentation de la cryptographie post-quantique basée sur les isogénies des courbes elliptiques sur FPGA présente plusieurs avantages :

- Tout d'abord, les FPGA sont reprogrammables, ce qui permet de mettre à jour les algorithmes cryptographiques au fur et à mesure que les recherches avancent et que de nouvelles techniques de cryptographie post-quantique sont développées.

Définition

Le FPGA (Field-Programmable Gate Array) est un dispositif électronique programmable qui permet de mettre en œuvre des circuits logiques personnalisés.

L'implémentation de la cryptographie post-quantique basée sur les isogénies des courbes elliptiques sur FPGA présente plusieurs avantages :

- Tout d'abord, les FPGA sont reprogrammables, ce qui permet de mettre à jour les algorithmes cryptographiques au fur et à mesure que les recherches avancent et que de nouvelles techniques de cryptographie post-quantique sont développées.
- En outre, les FPGA peuvent être optimisés pour des performances élevées et une faible consommation d'énergie, ce qui est essentiel pour les applications de sécurité en temps réel et les dispositifs à faible consommation d'énergie.

L'implémentation de la cryptographie post-quantique basée sur les isogénies des courbes elliptiques sur FPGA présente cependant plusieurs défis :

L'implémentation de la cryptographie post-quantique basée sur les isogénies des courbes elliptiques sur FPGA présente cependant plusieurs défis :

- Les algorithmes basés sur les isogénies nécessitent des opérations arithmétiques complexes et des structures de données spécifiques pour gérer les courbes elliptiques et les isogénies.

L'implémentation de la cryptographie post-quantique basée sur les isogénies des courbes elliptiques sur FPGA présente cependant plusieurs défis :

- Les algorithmes basés sur les isogénies nécessitent des opérations arithmétiques complexes et des structures de données spécifiques pour gérer les courbes elliptiques et les isogénies.
- Les concepteurs de FPGA doivent optimiser ces opérations et structures de données pour garantir une performance maximale tout en minimisant la consommation d'énergie et l'utilisation des ressources matérielles.

Problème difficile : isogénies sur courbes elliptiques supersingulaires

Isogénies sur courbes elliptiques supersingulaires :

Nous allons définir le problème difficile sur lequel est basé le schéma SIDH que nous détaillerons plus tard.

La forme la plus populaire de cryptographie à clé publique pour les applications actuelles est la **cryptographie à courbe elliptique (ECC)**.

L'ECC définit des points sur une courbe elliptique et des formules spécifiques de doublement et d'addition de points pour passer d'un point à l'autre. La multiplication scalaire de points utilise une séquence de doublages et d'additions de points pour évaluer efficacement les multiplications de points $Q = kP = P + P + \dots + P$.

Les cryptosystèmes basés sur l'ECC reposent sur la difficulté de **résoudre le problème du logarithme discret de la courbe elliptique (ECDL)**, de sorte qu'étant donné Q et P dans l'équation précédente, il est impossible de déterminer le multiple scalaire k pour les courbes elliptiques avec des points d'un grand ordre.

Toutefois, avec l'émergence des ordinateurs quantiques dans un avenir proche, **ces cryptosystèmes qui reposent sur l'ECDL ne seront plus sûrs**, car le multiple scalaire peut être facilement récupéré à l'aide de l'algorithme de Shor sur un ordinateur quantique.

Par conséquent, l'ECC standard n'est plus applicable pour la sécurité à long terme et d'autres systèmes résilients au niveau quantique ont été proposés.

La cryptographie basée sur l'isogénie utilise également des points sur une courbe elliptique, mais elle est basée sur la difficulté de calculer les isogénies entre les courbes elliptiques.

Une **isogénie** peut être considérée comme une **cartographie algébrique unique entre deux courbes elliptiques** qui satisfait à la loi des groupes.

Childs et al ont présenté un algorithme permettant de calculer les isogénies sur les courbes ordinaires en temps sous-exponentiel, ce qui rend l'utilisation des cryptosystèmes basés sur les isogénies sur les courbes ordinaires peu sûre en présence d'ordinateurs quantiques.

Cependant, il n'existe pas d'algorithme connu pour calculer les isogénies sur les courbes supersingulaires en temps sous-exponentiel.

L'anneau d'endomorphisme d'une courbe est défini comme l'anneau formé par l'ensemble des endomorphismes d'une courbe elliptique ainsi que par la carte nulle sous l'effet de l'addition de points et de la composition fonctionnelle.

Les **courbes supersingulières** ont un anneau d'endomorphisme avec un rang \mathbb{N} égal à 4. Ces courbes peuvent être définies sur l'ensemble F_p ou l'ensemble F_{p^2} .

Ainsi, toutes les courbes supersingulières peuvent être représentées dans F_{p^2} . Plus précisément, les courbes supersingulières ont la propriété que pour chaque nombre premier $l \neq p$, il existe $l + 1$ isogénies de degré l à partir d'une courbe de base.

Nous calculons une isogénie entre les courbes en utilisant un noyau, k , tel que $\phi : E \rightarrow E/\langle \kappa \rangle$. En outre, nous ramenons des points de la courbe originale à la courbe isogène en évaluant l'isogénie à ces points.

Le j -invariant est un discriminant basé sur les coefficients de la courbe elliptique.

Calcul des isogénies de grand degré

Le degré d'une isogénie est son degré en tant que carte algébrique. Les calculs d'isogénie peuvent être effectués de manière itérative. Étant donné une courbe elliptique E et un point R d'ordre l^e , nous calculons $\phi : E \rightarrow E/\langle R \rangle$ en décomposant ϕ en une chaîne d'isogénies de degré l , $\phi = \phi_{e-1} \circ \dots \circ \phi_0$, comme suit :

Fixez $E_0 = E$ et $R_0 = R$, et définissez :

$$E_{i+1} = E_i / \langle l^{e-i-1} R_i \rangle \phi_i : E_i \rightarrow E_{i+1} R_{i+1} = \phi_i(R_i)$$

Calcul des isogénies de grand degré

Essentiellement, des additions de points sont utilisées pour calculer le noyau à chaque itération et les formules de Vélu sont utilisées pour calculer ϕ_i et E_{i+1} . Cette méthode s'applique spécifiquement aux isogénies et à la construction de courbes.

Une stratégie optimale pour calculer ces isogénies consiste à parcourir un grand graphe acyclique dirigé en forme de triangle jusqu'aux feuilles, comme le montre la figure ci-dessous :

Calcul des isogénies de grand degré

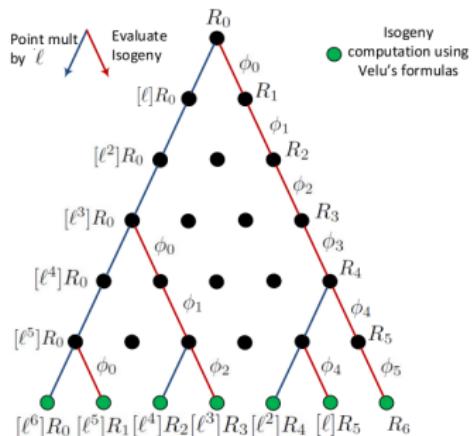


Fig. 1 Structure de calcul des isogénies de grand degré.

Pour ce graphe, effectuer une multiplication par ℓ revient à marcher vers la gauche et évaluer une relation d'isogénie revient à marcher vers la droite. Le calcul d'une isogénie à chacune des feuilles est utilisé pour calculer la cartographie isogène complète.

ISIMA

Schéma cryptographique : SIDH (Supersingular Isogeny Diffie-Hellman)s

Introduction

Jao et De Feo ont proposé un échange de clés basé sur les isogénies des courbes elliptiques supersingulaires.

Le schéma de SIDH ressemble au schéma standard de Diffie-Hellman sur courbe elliptique (ECDH), mais il va plus loin en calculant les isogénies sur de grands degrés.

Alice et Bob veulent échanger une clé secrète sur un canal non sécurisé.

Ils choisissent une isogénie lisse première p de la forme $l_A^a l_B^b \cdot f \pm 1$ où A et B sont de petits nombres premiers, a et b sont des entiers positifs et f est un petit cofacteur pour rendre le nombre premier.

Ils définissent une courbe elliptique supersingulière, E_0 (dans l'ensemble F_q) où $q = p^2$.

Enfin, ils choisissent quatre points sur la courbe qui forment les bases P_A, Q_A et P_B, Q_B , qui agissent comme générateurs pour $E_0[l_A^a]$ et $E_0[l_B^b]$, respectivement.

Dans un graphe d'isogénies supersingulaires où les sommets représentent des courbes isomorphes, et les arêtes des isogénies de degré l , l'impossibilité de découvrir un chemin reliant deux sommets particuliers assure la sécurité de ce protocole.

Essentiellement, chaque partie effectue des promenades apparemment aléatoires dans les graphes des isogénies de degré l_A^a et l_B^b pour arriver à une courbe ayant la même j -invariance.

Alice choisit deux clés privées $m_A, n_A \in \mathbb{N}/I_A^a\mathbb{N}$ avec la stipulation qu'elles ne sont pas toutes deux divisibles par I_A^a .

De l'autre côté, Bob choisit deux clés privées $m_B, n_B \in \mathbb{N}/I_B^b\mathbb{N}$, où les deux clés privées ne sont pas divisibles par I_B^b . À partir de là, le protocole d'échange de clés peut être décomposé en deux tours de la manière suivante :

Alice choisit deux clés privées $m_A, n_A \in \mathbb{N}/I_A^a\mathbb{N}$ avec la stipulation qu'elles ne sont pas toutes deux divisibles par I_A^a .

De l'autre côté, Bob choisit deux clés privées $m_B, n_B \in \mathbb{N}/I_B^b\mathbb{N}$, où les deux clés privées ne sont pas divisibles par I_B^b . À partir de là, le protocole d'échange de clés peut être décomposé en deux tours de la manière suivante :

- Calculer $R = <[m]P + [n]Q>$ pour les points P, Q .

Alice choisit deux clés privées $m_A, n_A \in \mathbb{N}/I_A^a\mathbb{N}$ avec la stipulation qu'elles ne sont pas toutes deux divisibles par I_A^a .

De l'autre côté, Bob choisit deux clés privées $m_B, n_B \in \mathbb{N}/I_B^b\mathbb{N}$, où les deux clés privées ne sont pas divisibles par I_B^b . À partir de là, le protocole d'échange de clés peut être décomposé en deux tours de la manière suivante :

- Calculer $R = <[m]P + [n]Q>$ pour les points P, Q .
- Calculer l'isogénie $\phi : E \rightarrow E/<R>$ pour une courbe supersingulière E .

Alice choisit deux clés privées $m_A, n_A \in \mathbb{N}/I_A^a\mathbb{N}$ avec la stipulation qu'elles ne sont pas toutes deux divisibles par I_A^a .

De l'autre côté, Bob choisit deux clés privées $m_B, n_B \in \mathbb{N}/I_B^b\mathbb{N}$, où les deux clés privées ne sont pas divisibles par I_B^b . À partir de là, le protocole d'échange de clés peut être décomposé en deux tours de la manière suivante :

- Calculer $R = <[m]P + [n]Q>$ pour les points P, Q .
- Calculer l'isogénie $\phi : E \rightarrow E/<R>$ pour une courbe supersingulière E .
- Calculer les images $\phi(R)$ et $\phi(S)$, où R et S sont les bases du parti opposé, uniquement pour le premier tour.

Protocole d'échange de clés

Voici le protocole d'échange de clés :

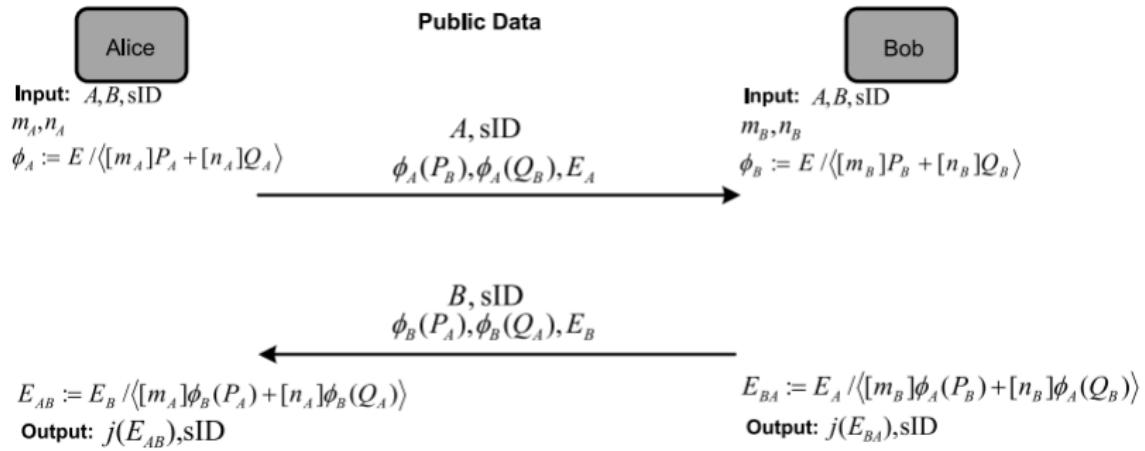


Fig. 2 Échange de clés Diffie-Hellman à isogénie supersingulière. "sID" est l'identifiant unique de la session.

Alice effectue la multiplication double point avec ses clés privées pour obtenir un noyau, $R_A = <[m_A]P_A + [n_A]Q_A>$ et calcule une isogénie
 $\phi_A : E_0 \rightarrow E_A = E_0 / <[m_A]P_A + [n_A]Q_A>.$

En calculant l'isogénie, elle calcule également la projection $\phi_A(P_B), \phi_A(Q_B) \subset E_A$ de la base $\{P_B, Q_B\}$ pour $E_0[I_B^b]$ sous son isogénie secrète ϕ_A , ce qui peut être fait efficacement en poussant les points P_B et Q_B à travers l'isogénie à chaque plus petite isogénie.

Sur un canal public, elle envoie ces points et la courbe E_A à Bob.

Bob effectue les mêmes calculs et envoie la courbe E_B et les points $\phi_B(P_A)$ et $\phi_B(Q_A)$ à **Alice**.

Pour le second tour, **Alice** effectue la double multiplication de points pour trouver un second noyau, $R_{AB} = <[m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A)>$, afin de calculer une seconde isogénie $\phi'_A : E_B \rightarrow E_{AB} = E_B / <[m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A)>$.

Bob effectue également une multiplication à deux points et calcule une seconde isogénie $\phi'_B : E_A \rightarrow E_{BA} = E_A / <[m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B)>$.

Alice et **Bob** ont maintenant des courbes isomorphes et peuvent utiliser la j -invariante commune comme clé secrète partagée.

The end.

Fin !

ISIMA

Bibliographie (1/3)

-  Agrawal S. (2019). Post Quantum Cryptography: An Introduction. IIT Madras. [Lien](#)
-  Institute for Quantum Computing. (2014, 23 octobre). Vinod Vaikuntanathan - Lattices and Cryptography : A Match Made in Heaven [Vidéo]. YouTube. [Lien](#)
-  Bos, J. W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., Stehlé, D. (2018). CRYSTALS - Kyber : A CCA-Secure Module-Lattice-Based KEM. HAL (Le Centre pour la Communication Scientifique Directe). [Lien](#)
-  Bernstein, D. J., Buchmann, J., Dahmen, E. (2009). Lattice-based Cryptography. Post-Quantum Cryptography. Springer Science Business Media (pp. 147-186)

Bibliographie (2/3)

Bibliographie (2/3)

-  Bernstein, D. J., Buchmann, J., Dahmen, E. (2009). Code-based cryptography. Post-Quantum Cryptography. Springer Science Business Media (pp. 95-145)
-  Bernstein, D. J., Buchmann, J., Dahmen, E. (2009). Hash-based Digital Signature Schemes. Post-Quantum Cryptography. Springer Science Business Media (pp. 35-91)
-  Contributeurs aux projets Wikimedia. (2021, août 18). Attaque de préimage.
-  Bernstein, D., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z. (2015). SPHINCS : Practical Stateless Hash-Based Signatures. Lecture Notes in Computer Science, 368-397. [Lien](#)

Bibliographie (3/3)

Bibliographie (3/3)

-  Bernstein, D., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P. (2019). The SPHINCS + Signature Framework. Computer and Communications Security. [Lien](#)
-  Kelsey, J., [COSIC - Computer Security and Industrial Cryptography]. (2019, 3 octobre). COSIC seminar « Introduction to Hash Based Signatures » (John Kelsey, KU Leuven NIST) [Vidéo]. YouTube. A partir de 1:48:00, à l'adresse [Lien](#)
-  Bernstein, D. J., Buchmann, J., Dahmen, E. (2009). Multivariate Public Key Cryptography. Post-Quantum Cryptography. Springer Science Business Media (pp. 193-229)
-  Ding, J., Chen, M.-S., Kannwischer, M., Patarin, J., Petzoldt, A., Schmidt, D., Yang, B.-Y. (2021). Rainbow. [Lien](#)
-  Koziel, B., Azarderakhsh, R., Kermani, M. M., Jao, D. (2017). Post-Quantum Cryptography on FPGA Based on Isogenies on Elliptic Curves. IEEE Transactions on Circuits and Systems I-regular Papers, 64(1), 86-99. [Lien](#)