Schéma k-sur-n sous Tails : formulation mathématique rigoureuse

F2F 60C4 1DAF 4B4A E829 2D4C 7C0C 2DF7 6AE4 D7F5

27 novembre 2025

Résumé

On formalise ici, dans un cadre entièrement mathématique, un schéma k-sur-n pour la gestion d'un secret maître, basé sur :

- le partage de secret de Shamir sur un corps fini ;
- la dérivation de clés par HKDF et PBKDF2 ;
- une signature Ed25519 (courbe elliptique sur un corps fini) ;
- un chiffrement hybride RSA–AES avec authentification d'intégrité (AES-GCM).

AES-GCM garantit la confidentialité et l'intégrité des données. L'authenticité est assurée différemment selon le contexte :

- **Parts de secret** : authentifiées par les mots de passe individuels (AES-GCM avec clé dérivée du mot de passe)
- **Signatures** : authentifiées cryptographiquement par Ed25519
- **Fichiers chiffrés hybrides RSA-AES** : intégrité garantie par AES-GCM, mais authenticité de la source non garantie
- **Clés publiques** : doivent être authentifiées après distribution publique par canaux sécurisés

Les objets sont définis dans leur structure algébrique naturelle, et les liens avec les scripts informatiques sont donnés en fin de document.

Table des matières

Table des notations

$\{0,1\}^t$	Mots binaires de longueur t
$\{0,1\}^*$	Mots binaires de longueur finie (union sur $t \geq 0$)
$\|x\|$	Longueur en bits de $x \in \{0,1\}^*$
$x \parallel y$	Concaténation de mots binaires
$x \oplus y$	XOR bit-à-bit (x, y de même longueur)
$\mathrm{val}_{\mathrm{be}}(b)$	Encodage big-endian $b \in \{0,1\}^t \to \{0,\ldots,2^t-1\}$
$\mathrm{val}_{\mathrm{le}}(b)$	Encodage little-endian $b \in \{0,1\}^t \to \{0,\ldots,2^t-1\}$
$S \in \{0,1\}^{256}$	Secret maître (32 octets)
s_0	Entier $\mathrm{val}_{\mathrm{be}}(S) \in \{0,\ldots,2^{256}-1\}$
$P = 2^{521}-1$	Premier de Mersenne, cardinal de $\mathbb{F}_P$
$\mathbb{F}_P$	Corps fini à P éléments
$\mathbb{F}_P[X]$	Polynômes à coefficients dans $\mathbb{F}_P$
$\mathcal{P}_{<k}$	Polynômes de degré $< k$ dans $\mathbb{F}_P[X]$
(x_i, y_i)	Part de Shamir du participant i ($\in \mathbb{F}_P^2$)
$q = 2^{255}-19$	Cardinal du corps de base de Ed25519
$E(\mathbb{F}_q)$	Groupe des points de la courbe elliptique
$B \in E(\mathbb{F}_q)$	Point de base d'ordre premier ℓ
ℓ	Ordre premier de B ($\approx 2^{252}$)
$\mathbb{Z}_\ell$	Anneau $\{0,\ldots,\ell-1\}$ modulo ℓ
$a \in \mathbb{Z}_\ell$	Clé privée Ed25519
$A = aB$	Clé publique Ed25519
(n, e, d)	Paramètres RSA : module, exposants public/privé
h	Fonction de hachage $h : \{0,1\}^* \to \{0,1\}^d$
HMAC_h	MAC HMAC basé sur h
HKDF_h	Dérivation de clé HKDF (RFC 5869)
$\mathrm{PBKDF2}_h$	Dérivation par mot de passe (PBKDF2, RFC 8018)
$\mathrm{MGF1}_h$	Génération de masque MGF1 (RFC 8017)
RSA-OAEP	RSA avec rembourrage OAEP (RFC 8017)
$\mathrm{AES\text{-}GCM}_K$	AES-GCM sous clé symétrique K
IKM	Input Keying Material (ici S)
saltEd	Sel HKDF constant pour K_{Ed}
W	Sel HKDF aléatoire par cérémonie pour K_{RSA}
$\mathrm{info}_{\mathrm{Ed}}$	Contexte HKDF pour Ed25519
$\mathrm{info}_{\mathrm{RSA}}$	Contexte HKDF pour RSA
P_i	Mot de passe du participant i
S_i	Sel PBKDF2 individuel
K_i	Clé symétrique pour chiffrer la part i
K_{Ed}	Graine Ed25519 dérivée de S
K_{RSA}	Clé pour chiffrer la clé privée RSA
K_{AES}	Clé de session AES
N, N_i	Nonces AES-GCM
$\perp$	Symbole d'échec/rejet

1 Cadre algébrique de base

1.1 Mots binaires et encodage

Définition 1.1 (Mots binaires)**.** Pour $t \in \mathbb{N}$, on note

$$\{0,1\}^t = \{0,1\}^t$$

l'ensemble des suites $(b_{t-1}, \ldots, b_0)$ de bits, avec la convention que $\{0,1\}^0 = \{\varepsilon\}$ contient le mot vide. On pose

$$\{0,1\}^* = \bigcup_{t \geq 0} \{0,1\}^t,$$

et pour $x \in \{0,1\}^*$, on note $\|x\|$ sa longueur (en bits).

Définition 1.2 (Concaténation et XOR)**.** La concaténation est l'application

$$\| : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^{a+b}$$

qui à (x, y) associe la suite obtenue en juxtaposant x et y.

Le XOR est l'application

$$\oplus : \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^t$$

définie par $(x, y) \mapsto (x_0 \oplus y_0, \ldots, x_{t-1} \oplus y_{t-1})$, où $\oplus$ désigne l'addition dans $\mathbb{F}_2$.

Définition 1.3 (Encodage big-endian)**.** Pour $t \in \mathbb{N}$ et $b = (b_{t-1}, \ldots, b_0) \in \{0,1\}^t$, on définit

$$\mathrm{val}_{\mathrm{be}}(b) = \sum_{j=0}^{t-1} b_j \, 2^{t-1-j} \in \{0, \ldots, 2^t - 1\}.$$

Cet encodage fournit une bijection canonique entre $\{0,1\}^t$ et $\{0, \ldots, 2^t - 1\}$.

Définition 1.4 (Encodage little-endian)**.** Pour $t \in \mathbb{N}$ et $b = (b_{t-1}, \ldots, b_0) \in \{0,1\}^t$, on définit

$$\mathrm{val}_{\mathrm{le}}(b) = \sum_{j=0}^{t-1} b_j \, 2^j \in \{0, \ldots, 2^t - 1\}.$$

Cet encodage est utilisé dans les standards Ed25519.

Remarque 1.5 (Convention d'écriture)**.** Dans la suite du document :

- L'encodage big-endian est utilisé pour le partage de Shamir et les représentations internes dans $\mathbb{F}_P$
- L'encodage little-endian est utilisé pour Ed25519 (scalaires et points)
- Le contexte déterminera clairement quel encodage est employé

1.2 Corps finis et polynômes

Définition 1.6 (Corps fini et polynômes)**.** Soit P un nombre premier. On note

$$\mathbb{F}_P = \mathbb{Z}/P\mathbb{Z}$$

le corps fini à P éléments, et

$$\mathbb{F}_P[X]$$

l'anneau des polynômes à coefficients dans $\mathbb{F}_P$. On pose

$$\mathcal{P}_{<k} = \{f \in \mathbb{F}_P[X] \mid \deg(f) < k\}.$$

Lemme 1.7 (Nombre de racines)**.** Soit $f \in \mathbb{F}_P[X]$ un polynôme non nul de degré $d \geq 0$. Alors f admet au plus d racines dans $\mathbb{F}_P$.

Démonstration. Par récurrence sur d. Le cas $d = 0$ est immédiat (un polynôme constant non nul n'a aucune racine). Pour $d \geq 1$, si f a une racine $a \in \mathbb{F}_P$, alors par division euclidienne dans $\mathbb{F}_P[X]$, on peut écrire $f(X) = (X - a)g(X)$ avec $g \in \mathbb{F}_P[X]$ de degré $d - 1$. En effet, la division de f par $(X - a)$ donne un quotient g et un reste r de degré < 1, donc constant. En évaluant en $X = a$, on obtient $f(a) = 0 = 0 \cdot g(a) + r$, donc $r = 0$.

Soit maintenant $b \neq a$ une autre racine de f. Alors $0 = f(b) = (b - a)g(b)$. Comme $b - a \neq 0$ et que $\mathbb{F}_P$ est un corps (donc intègre), on en déduit $g(b) = 0$. Ainsi, toutes les racines de f distinctes de a sont des racines de g. Par hypothèse de récurrence, g a au plus $d - 1$ racines, donc f a au plus d racines. □

1.3 Théorèmes arithmétiques fondamentaux

Définition 1.8 (Indicatrice d'Euler)**.** L'indicatrice d'Euler $\varphi(n)$ est définie pour tout entier $n \geq 1$ comme le nombre d'entiers compris entre 1 et n qui sont premiers avec n :

$$\varphi(n) = \#\{k \in \{1, 2, \ldots, n\} \mid \gcd(k, n) = 1\}$$

Théorème 1.9 (Théorème d'Euler)**.** Si a et n sont premiers entre eux, alors :

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

Preuve utilisant la théorie des groupes. L'ensemble des éléments inversibles de l'anneau $\mathbb{Z}/n\mathbb{Z}$ est noté $(\mathbb{Z}/n\mathbb{Z})^\times$. Par l'identité de Bézout, un élément $a \in \{0, 1, \ldots, n-1\}$ est inversible modulo n si et seulement si $\gcd(a, n) = 1$. Ainsi, le groupe multiplicatif $(\mathbb{Z}/n\mathbb{Z})^\times$ a pour ordre $\varphi(n)$. Par le théorème de Lagrange, l'ordre de tout élément du groupe divise l'ordre du groupe. Donc, pour tout a inversible modulo n, on a $a^{\varphi(n)} \equiv 1 \pmod{n}$. □

Théorème 1.10 (Petit théorème de Fermat)**.** Si p est premier et a n'est pas divisible par p, alors :

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

Démonstration. C'est un cas particulier du théorème d'Euler, car pour p premier, $\varphi(p) = p - 1$. □

Théorème 1.11 (Théorème de Gauss (lemme d'Euclide))**.** Soient a, b, c des entiers. Si a et b sont premiers entre eux et a divise bc, alors a divise c.

Démonstration. Puisque $\gcd(a, b) = 1$, par l'identité de Bézout, il existe des entiers u, v tels que :
$$au + bv = 1$$
Multiplions cette égalité par c :
$$auc + bvc = c$$
Par hypothèse, a divise bc, donc a divise bvc. De plus, a divise évidemment auc. Donc a divise la somme $auc + bvc = c$. $\square$

2 Secret maître et schéma de Shamir

2.1 Secret maître

Définition 2.1 (Secret maître)**.** On fixe un entier
$$P = 2^{521} - 1$$
et le corps $\mathbb{F}_P$. Le secret maître est un mot binaire
$$S \in \{0, 1\}^{256}$$
tiré uniformément. On définit
$$s_0 = \mathrm{val}_{\mathrm{be}}(S) \in \{0, \ldots, 2^{256} - 1\}, \qquad s = s_0 \bmod P \in \mathbb{F}_P.$$
Comme $P > 2^{256} - 1$, on a $s = s_0$ dans $\mathbb{F}_P$, ce qui garantit que l'application $S \mapsto s$ est une injection de $\{0, 1\}^{256}$ dans $\mathbb{F}_P$.

Ainsi, S est la représentation binaire utilisée pour la dérivation de clés, et $s \in \mathbb{F}_P$ est la représentation dans le corps fini utilisée pour le partage de secret.

2.2 Schéma (k, n) de Shamir

Définition 2.2 (Partage de Shamir)**.** Soient des entiers n, k vérifiant $2 \leq k \leq n$ et $n < P$. On définit le schéma (k, n) de Shamir sur $\mathbb{F}_P$ comme suit.

- À partir de $s \in \mathbb{F}_P$, on choisit indépendamment et uniformément $a_1, \ldots, a_{k-1} \in \mathbb{F}_P$.
- On définit
$$f(X) = s + a_1 X + a_2 X^2 + \cdots + a_{k-1} X^{k-1} \in \mathcal{P}_{<k}.$$
- Pour $i \in \{1, \ldots, n\}$, on fixe $x_i = i \in \mathbb{F}_P$ (en identifiant l'entier i à son image dans $\mathbb{F}_P$) et on pose
$$y_i = f(x_i) \in \mathbb{F}_P.$$
- La part i-ème est le couple $(x_i, y_i) \in \mathbb{F}_P^2$.

Définition 2.3 (Algorithmes Share et Reconstruct)**.** On définit formellement les applications :

- Share : $\mathbb{F}_P \to (\mathbb{F}_P^2)^n$ qui à s associe la famille $((x_i, y_i))_{1 \leq i \leq n}$.
- Reconstruct : $(\mathbb{F}_P^2)^k \to \mathbb{F}_P$ qui, à k points distincts (x_{i_j}, y_{i_j}), associe $s = f(0)$ obtenu par interpolation de Lagrange.

Théorème 2.4 (Unicité de l'interpolation)**.** Soient $x_1, \dots, x_k \in \mathbb{F}_P$ deux à deux distincts et $y_1, \dots, y_k \in \mathbb{F}_P$. Il existe un unique $f \in \mathcal{P}_{<k}$ tel que $f(x_j) = y_j$ pour tout $j \in \{1, \dots, k\}$.

Démonstration. Par le lemme sur le nombre de racines, deux polynômes de degré $< k$ coïncidant en k points distincts sont égaux. L'existence s'obtient par la formule d'interpolation de Lagrange :

$$f(X) = \sum_{j=1}^{k} y_j \ell_j(X), \qquad \ell_j(X) = \prod_{\substack{m=1\\ m\neq j}}^{k} \frac{X - x_m}{x_j - x_m} \in \mathbb{F}_P[X].$$

Les dénominateurs $x_j - x_m$ sont inversibles dans $\mathbb{F}_P$ car les x_j sont distincts. On vérifie $\ell_j(x_i) = \delta_{ij}$, d'où $f(x_j) = y_j$. $\square$

Propriété 2.5 (Reconstruction du secret)**.** Soit $I = \{i_1, \dots, i_k\}$ avec $i_1 < \cdots < i_k$. Soit f le polynôme défini par le schéma de Shamir. Alors

$$s = f(0) = \sum_{j=1}^{k} y_{i_j} \lambda_j, \qquad \lambda_j = \ell_j(0) = \prod_{\substack{m=1\\ m\neq j}}^{k} \frac{-x_{i_m}}{x_{i_j} - x_{i_m}} \in \mathbb{F}_P.$$

Démonstration. On applique le théorème 2.4 avec $(x_j, y_j) = (x_{i_j}, y_{i_j})$. On obtient

$$f(X) = \sum_{j=1}^{k} y_{i_j} \ell_j(X).$$

Comme $f(X) = s + a_1X + \cdots + a_{k-1}X^{k-1}$, on a $f(0) = s$. En évaluant en $X = 0$, on obtient

$$s = f(0) = \sum_{j=1}^{k} y_{i_j} \ell_j(0),$$

avec $\lambda_j = \ell_j(0)$. $\square$

Proposition 2.6 (Confidentialité parfaite du schéma de Shamir)**.** Soit $t < k$ et $I = \{i_1, \dots, i_t\} \subset \{1, \dots, n\}$. On suppose que $(s, a_1, \dots, a_{k-1})$ est uniforme dans $\mathbb{F}_P^k$. Alors, pour toute réalisation fixée des parts $\big((x_{i_j}, y_{i_j})\big)_{1\le j\le t}$ et pour tous $s_0, s_1 \in \mathbb{F}_P$,

$$\Pr[s = s_0 \mid (x_{i_j}, y_{i_j})] = \Pr[s = s_1 \mid (x_{i_j}, y_{i_j})].$$

En particulier, la loi a posteriori de s conditionnellement à $t < k$ parts reste uniforme sur $\mathbb{F}_P$.

Démonstration. Les t équations $f(x_{i_j}) = y_{i_j}$ forment un système linéaire de rang t (la matrice de Vandermonde partielle est de rang plein car les x_{i_j} sont distincts). Ce système impose t contraintes indépendantes sur les k variables $(s, a_1, \dots, a_{k-1})$.

Pour toute valeur fixée $s_0 \in \mathbb{F}_P$, le système restreint aux $(a_1, \dots, a_{k-1})$ a t équations indépendantes sur $k-1$ variables, donc admet exactement P^{k-1-t} solutions. Ce nombre est indépendant de s_0.

Par Bayes et l'uniformité a priori sur $\mathbb{F}_P^k$:

$$\Pr[s = s_0 \mid \text{parts}] = \frac{\Pr[\text{parts} \mid s = s_0] \cdot \Pr[s = s_0]}{\Pr[\text{parts}]} = \frac{P^{-t} \cdot P^{-1}}{P^{-t}} = \frac{1}{P}.$$

Ainsi, la loi conditionnelle reste uniforme sur $\mathbb{F}_P$. $\square$

3 Fonctions de hachage, HMAC, HKDF, PBKDF2

3.1 Fonction de hachage

Définition 3.1 (Fonction de hachage)**.** Une fonction de hachage est une application

$$h : \{0,1\}^* \to \{0,1\}^d,$$

pour un d fixé (par exemple $d = 256$ pour SHA-256). Dans ce document, h désigne une telle fonction fixée une fois pour toutes.

3.2 HMAC

Définition 3.2 (Longueur de bloc et masques internes)**.** On fixe un entier $k_{\text{blk}} \in \mathbb{N}^*$, appelé longueur de bloc de HMAC. On se donne deux mots binaires

$$\mathsf{ipad}, \mathsf{opad} \in \{0,1\}^{k_{\text{blk}}}$$

appelés respectivement masque interne et masque externe. Dans les spécifications usuelles, ce sont les octets $0x36$ (pour ipad) et $0x5C$ (pour opad) répétés de façon à obtenir k_{blk} bits.

Définition 3.3 (Normalisation de clé pour HMAC)**.** On définit une application

$$\text{NormKey} : \{0,1\}^* \to \{0,1\}^{k_{\text{blk}}}$$

qui, pour toute clé $K \in \{0,1\}^*$,
— si $\|K\| > k_{\text{blk}}$, alors $K_{\text{blk}} = \text{troncature}(h(K))$ aux k_{blk} premiers bits ;
— si $\|K\| < k_{\text{blk}}$, alors $K_{\text{blk}} = K \parallel 0^{k_{\text{blk}} - \|K\|}$;
— si $\|K\| = k_{\text{blk}}$, alors $K_{\text{blk}} = K$.

Ainsi, pour tout $K \in \{0,1\}^*$, la valeur $\text{NormKey}(K)$ est un mot binaire de longueur exactement k_{blk}.

Définition 3.4 (HMAC basé sur h)**.** On fixe une fonction de hachage

$$h : \{0,1\}^* \to \{0,1\}^d.$$

On définit la fonction

$$\text{HMAC}_h : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^d$$

par la formule :

$$\text{HMAC}_h(K, M) = h\big((\text{NormKey}(K) \oplus \mathsf{opad}) \parallel h\big((\text{NormKey}(K) \oplus \mathsf{ipad}) \parallel M\big)\big).$$

3.3 Fonction d'encodage d'entier

Définition 3.5 (Fonction INT_4)**.** On définit l'application

$$\text{INT}_4 : \{0, \ldots, 2^{32} - 1\} \to \{0,1\}^{32}$$

qui à un entier i associe son encodage big-endian sur 4 octets (32 bits) :

$$\text{INT}_4(i) = (b_{31} b_{30} \ldots b_0) \text{ où } i = \sum_{j=0}^{31} b_j 2^{31-j}.$$

3.4 HKDF

Définition 3.6 (HKDF)**.** Soit $h : \{0,1\}^* \to \{0,1\}^d$ une fonction de hachage. On définit la fonction de dérivation de clé HKDF selon RFC 5869 :

$$\mathrm{HKDF}_h : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \times \mathbb{N} \to \{0,1\}^L$$

comme suit. Soient IKM $\in \{0,1\}^*$ (input keying material), salt $\in \{0,1\}^*$, info $\in \{0,1\}^*$, et $L \in \mathbb{N}$ (longueur de sortie en octets). On pose :

- PRK $= \mathrm{HMAC}_h(\mathrm{salt}, \mathrm{IKM})$ (extraction)
- On définit itérativement pour $i = 1, 2, \ldots$ jusqu'à obtenir L octets :

$$\begin{aligned} T_1 &= \mathrm{HMAC}_h(\mathrm{PRK}, \mathrm{info} \,\|\, \mathrm{INT}_4(1)) \\ T_2 &= \mathrm{HMAC}_h(\mathrm{PRK}, T_1 \,\|\, \mathrm{info} \,\|\, \mathrm{INT}_4(2)) \\ &\vdots \\ T_i &= \mathrm{HMAC}_h(\mathrm{PRK}, T_{i-1} \,\|\, \mathrm{info} \,\|\, \mathrm{INT}_4(i)) \end{aligned}$$

- La sortie est la troncature aux L octets de $T_1 \,\|\, T_2 \,\|\, \ldots$

3.5 PBKDF2

Définition 3.7 (PBKDF2)**.** On fixe un paramètre d'itération $c \in \mathbb{N}^*$ et un hachage h. On définit

$$\mathrm{PBKDF2}_h : \{0,1\}^* \times \{0,1\}^* \times \mathbb{N}^* \times \mathbb{N} \to \{0,1\}^{8L}$$

où L est le nombre d'octets souhaités. Pour un mot de passe $P \in \{0,1\}^*$, un sel $S \in \{0,1\}^*$ et un entier $L \geq 1$, on définit, pour $i \geq 1$:

$$U_1 = \mathrm{HMAC}_h(P,\ S \,\|\, \mathrm{INT}_4(i)),$$

$$U_j = \mathrm{HMAC}_h(P,\ U_{j-1}) \quad (2 \leq j \leq c),$$

puis

$$F(P, S, c, i) = U_1 \oplus U_2 \oplus \ldots \oplus U_c.$$

La sortie de PBKDF2 est le préfixe aux $8L$ bits de la concaténation

$$F(P, S, c, 1) \,\|\, F(P, S, c, 2) \,\|\, \ldots$$

4 Structure de groupe et signature Ed25519

4.1 Courbe elliptique Ed25519

Définition 4.1 (Courbe elliptique Ed25519)**.** Soit $q = 2^{255} - 19$ et $\mathbb{F}_q$ le corps fini correspondant. La courbe elliptique Ed25519 est définie par l'équation de Twisted Edwards :

$$E : -x^2 + y^2 = 1 + dx^2y^2 \quad \text{sur } \mathbb{F}_q$$

où $d = -\frac{121665}{121666} \in \mathbb{F}_q$, avec la division interprétée comme la multiplication par l'inverse modulo q.

Définition 4.2 (Loi de groupe sur $E(\mathbb{F}_q)$)**.** Soient $P_1 = (x_1, y_1)$ et $P_2 = (x_2, y_2)$ deux points de $E(\mathbb{F}_q)$. L'addition est définie par :

$$P_1 + P_2 = (x_3, y_3) = \left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 + x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right)$$

où toutes les opérations arithmétiques sont effectuées dans $\mathbb{F}_q$. L'élément neutre est le point $\mathcal{O} = (0, 1)$.

Proposition 4.3 (Cardinal du groupe)**.** Le groupe $E(\mathbb{F}_q)$ est abélien fini de cardinal :

$$\#E(\mathbb{F}_q) = 8\ell$$

où ℓ est le nombre premier $\ell = 2^{252} + 27742317777372353535851937790883648493$.

Remarque 4.4 (Admission du cardinal)**.** Le calcul du cardinal de la courbe Ed25519 est admis. Il peut être obtenu par l'algorithme de Schoof-Elkies-Atkin (SEA) et a été vérifié de manière indépendante par plusieurs implémentations.

Définition 4.5 (Point de base standard Ed25519)**.** On fixe le point de base standard $B \in E(\mathbb{F}_q)$ d'ordre ℓ, dont les coordonnées affines sont :

$$y_B = 46316835694926478169428394003475163141307993866256225615783033603165251855960$$
$$x_B = 15112221349535400772501151409588531511454012693041857206046113283949847762202$$

Ces valeurs satisfont l'équation de la courbe.

Remarque 4.6 (Admission de l'ordre du point de base)**.** L'ordre premier ℓ du point de base B est admis. Cette propriété essentielle pour la sécurité cryptographique a été vérifiée par la communauté.

Définition 4.7 (Sous-groupe cyclique principal)**.** Le sous-groupe cyclique d'ordre ℓ est :

$$\langle B \rangle = \{aB \mid a \in \mathbb{Z}_\ell\}$$

qui est isomorphe à $\mathbb{Z}_\ell$.

Remarque 4.8 (Utilisation cryptographique)**.** Seuls les points du sous-groupe $\langle B \rangle$ d'ordre premier ℓ sont utilisés pour la cryptographie. Le clampage dans Ed25519 garantit que les scalaires appartiennent à $\mathbb{Z}_\ell$.

4.2 Encodage des points et scalaires

Définition 4.9 (Encodage des points et scalaires)**.** L'encodage canonique d'un point $P = (x, y) \in E(\mathbb{F}_q)$ est défini par :

$$\mathrm{enc}(P) = \mathrm{bytes}_{\mathrm{le}}(y) \,\|\, p \in \{0, 1\}^{256}$$

où $\mathrm{bytes}_{\mathrm{le}}(y)$ est la représentation little-endian de y sur 255 bits (32 octets, le bit de poids fort ignoré), et p est le bit de parité de x, c'est-à-dire le bit le moins significatif de x.

L'encodage d'un scalaire $s \in \mathbb{Z}_\ell$ est sa représentation little-endian sur 32 octets.

Remarque 4.10 (Encodage compressé)**.** L'encodage utilisé pour Ed25519 est l'encodage compressé standard où seul le coordonnée y est stockée avec un bit de parité pour permettre la reconstruction de x. L'encodage est en little-endian conformément au standard.

4.3 Décodage des points Ed25519

Définition 4.11 (Symbole de Legendre)**.** Soit p un nombre premier impair et a un entier. Le symbole de Legendre $\left(\frac{a}{p}\right)$ est défini par :

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{si } p \mid a \\ 1 & \text{si } a \text{ est un carré modulo } p \\ -1 & \text{sinon} \end{cases}$$

Théorème 4.12 (Critère d'Euler)**.** Soit p un nombre premier impair et a un entier non divisible par p. Alors :

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \pmod p$$

En particulier, $a^{(p-1)/2} \equiv 1 \pmod p$ si et seulement si a est un carré modulo p, et $a^{(p-1)/2} \equiv -1 \pmod p$ si et seulement si a n'est pas un carré modulo p.

Démonstration. Si a est un carré modulo p, alors $a \equiv b^2 \pmod p$ pour un b non divisible par p, donc $a^{(p-1)/2} \equiv b^{p-1} \equiv 1 \pmod p$ par le théorème de Fermat.

Réciproquement, dans $\mathbb{F}_p^*$ qui est cyclique d'ordre $p-1$, considérons l'homomorphisme $\phi : x \mapsto x^2$. Son noyau est $\{\pm 1\}$ d'ordre 2, donc son image (les carrés) est d'ordre $(p-1)/2$. L'équation $x^{(p-1)/2} = 1$ a au plus $(p-1)/2$ solutions, et tous les carrés la vérifient. Donc les non-carrés vérifient $a^{(p-1)/2} = -1$. □

Lemme 4.13 (Lemme de Gauss pour les résidus quadratiques)**.** Soit p un nombre premier impair et a un entier non divisible por p. Considérons l'ensemble :

$$A = \{a, 2a, 3a, \dots, \tfrac{p-1}{2}a\}$$

Pour chaque élément $ka \in A$, on considère son *résidu modulo* p, c'est-à-dire l'unique entier r_k tel que $1 \le r_k \le p-1$ et $r_k \equiv ka \pmod p$.

On sépare ces résidus en deux groupes :

— Les résidus $r_1, \dots, r_k$ qui sont inférieurs ou égaux à $p/2$

— Les résidus $s_1, \dots, s_\mu$ qui sont supérieurs à $p/2$

où k est le nombre de résidus dans le premier groupe et μ est le nombre de résidus dans le second groupe. Alors :

$$\left(\frac{a}{p}\right) = (-1)^\mu$$

Démonstration. **Étape 1 : Les nombres $r_1, \dots, r_k, p - s_1, \dots, p - s_\mu$ sont distincts**

Les r_i sont par définition dans $\{1, \dots, \lfloor p/2 \rfloor\}$. Pour les s_j, comme $s_j > p/2$, on a $p - s_j \in \{1, \dots, \lfloor p/2 \rfloor\}$.

Supposons par l'absurde qu'il existe i, j tels que $r_i = p - s_j$. Alors :

$$r_i + s_j = p$$

Mais $r_i \equiv \alpha a \pmod p$ et $s_j \equiv \beta a \pmod p$ pour certains $\alpha, \beta \in \{1, \dots, \frac{p-1}{2}\}$. Donc :

$$\alpha a + \beta a \equiv 0 \pmod p \quad \Rightarrow \quad (\alpha + \beta) a \equiv 0 \pmod p$$

Puisque $p \nmid a$, on doit avoir $p \mid (\alpha + \beta)$. Mais $2 \le \alpha + \beta \le p - 1$, contradiction.

Ainsi, ces $k+\mu = \frac{p-1}{2}$ nombres forment une permutation de $\{1, 2, \dots, \frac{p-1}{2}\}$.
Étape 2 : Relations produits
Le produit de tous les éléments de $\{1, 2, \dots, \frac{p-1}{2}\}$ est $\left(\frac{p-1}{2}\right)!$. Donc :

$$r_1 \cdots r_k (p-s_1)\cdots(p-s_\mu) = \left(\frac{p-1}{2}\right)!$$

Modulo p, on a $p - s_j \equiv -s_j \pmod p$, donc :

$$r_1 \cdots r_k (p-s_1)\cdots(p-s_\mu) \equiv r_1 \cdots r_k (-s_1)\cdots(-s_\mu) = (-1)^\mu r_1 \cdots r_k s_1 \cdots s_\mu \pmod p$$

Ainsi :

$$(-1)^\mu r_1 \cdots r_k s_1 \cdots s_\mu \equiv \left(\frac{p-1}{2}\right)! \pmod p \tag{1}$$

D'autre part, le produit des éléments de A est :

$$a \cdot 2a \cdot 3a \cdots \tfrac{p-1}{2} a = a^{(p-1)/2} \left(\frac{p-1}{2}\right)!$$

Modulo p, ce produit est congru au produit des résidus des éléments de A, c'est-à-dire $r_1 \cdots r_k s_1 \cdots s_\mu$. Donc :

$$a^{(p-1)/2} \left(\frac{p-1}{2}\right)! \equiv r_1 \cdots r_k s_1 \cdots s_\mu \pmod p \tag{2}$$

Étape 3 : Conclusion
En substituant (2) dans (1), on obtient :

$$(-1)^\mu a^{(p-1)/2} \left(\frac{p-1}{2}\right)! \equiv \left(\frac{p-1}{2}\right)! \pmod p$$

Puisque $\left(\frac{p-1}{2}\right)!$ n'est pas divisible par p, on peut simplifier :

$$(-1)^\mu a^{(p-1)/2} \equiv 1 \pmod p \quad \Rightarrow \quad a^{(p-1)/2} \equiv (-1)^\mu \pmod p$$

Par le critère d'Euler (Théorème 4.12), on a $a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \pmod p$, donc :

$$\left(\frac{a}{p}\right) = (-1)^\mu$$

□

Lemme 4.14 (Caractère quadratique de 2)**.** Soit q un nombre premier impair. Alors :

$$\left(\frac{2}{q}\right) = (-1)^{\frac{q^2-1}{8}}$$

En particulier, pour $q \equiv 5 \pmod 8$, on a $\left(\frac{2}{q}\right) = -1$.

Démonstration. Nous appliquons le lemme 4.13 avec $a = 2$. Soit

$$A = \{2, 4, 6, \ldots, 2 \cdot \frac{q-1}{2}\} = \{2, 4, \ldots, q-1\}$$

Soit μ le nombre d'éléments de A dont le reste modulo q est supérieur à $q/2$. Alors par le lemme de Gauss :

$$\left(\frac{2}{q}\right) = (-1)^{\mu}$$

Les éléments de A sont tous dans l'intervalle $[2, q-1]$. Un élément $2k$ (avec $1 \leq k \leq \frac{q-1}{2}$) a un reste modulo q supérieur à $q/2$ si et seulement si $2k > q/2$, c'est-à-dire $k > q/4$.

Le nombre de tels k est donc :

$$\mu = \left\lfloor \frac{q-1}{2} \right\rfloor - \left\lfloor \frac{q}{4} \right\rfloor$$

Écrivons $q = 8m + r$ avec $r \in \{1, 3, 5, 7\}$. Alors :

$$\frac{q^2-1}{8} = \frac{(8m+r)^2-1}{8} = 8m^2 + 2mr + \frac{r^2-1}{8}$$

D'autre part, calculons μ pour chaque cas :

— Si $q = 8m+1$, alors $\mu = 4m - 2m = 2m$ (pair)

— Si $q = 8m+3$, alors $\mu = 4m + 1 - 2m = 2m + 1$ (impair)

— Si $q = 8m+5$, alors $\mu = 4m + 2 - (2m+1) = 2m + 1$ (impair)

— Si $q = 8m+7$, alors $\mu = 4m + 3 - (2m+1) = 2m + 2$ (pair)

Donc $\left(\frac{2}{q}\right) = (-1)^{\mu} = (-1)^{\frac{q^2-1}{8}}$.

Pour $q \equiv 5 \pmod 8$, on a $q = 8m + 5$, donc $\frac{q^2-1}{8} = 8m^2 + 10m + 3$ qui est impair, donc $\left(\frac{2}{q}\right) = -1$. □

Théorème 4.15 (Structure des racines carrées dans $\mathbb{F}_q$ pour $q \equiv 5 \pmod 8$)**.** Soit $q = 2^{255} - 19$ et $a \in \mathbb{F}_q$ un élément non nul. Si a est un carré dans $\mathbb{F}_q$, alors l'ensemble de ses racines carrées est $\{x, -x\}$ où :

$$x = a^{(q+3)/8} \quad \text{ou} \quad x = a^{(q+3)/8} \cdot 2^{(q-1)/4}$$

De plus, ces deux racines carrées ont des bits de parité opposés.

Démonstration. Soit $q = 2^{255} - 19 \equiv 5 \pmod 8$. On peut écrire $q = 8k + 5$ avec $k \in \mathbb{Z}$, donc $(q+3)/8 = k + 1$ est un entier.

Supposons que a est un carré dans $\mathbb{F}_q$. Calculons $x_0 = a^{(q+3)/8}$:

$$x_0^2 = \left(a^{(q+3)/8}\right)^2 = a^{(q+3)/4} = a \cdot a^{(q-1)/4}$$

Soit $b = a^{(q-1)/4}$. Puisque a est un carré, le critère d'Euler (Théorème 4.12) donne $a^{(q-1)/2} = 1$, donc $b^2 = a^{(q-1)/2} = 1$, ce qui implique $b = \pm 1$.

On distingue deux cas :

— Si $b = 1$, alors $x_0^2 = a \cdot 1 = a$, donc x_0 est une racine carrée de a.

— Si $b = -1$, alors $x_0^2 = a \cdot (-1) = -a$. Soit $i = 2^{(q-1)/4}$. Alors :

$$i^2 = (2^{(q-1)/4})^2 = 2^{(q-1)/2}$$

Par le lemme précédent, pour $q \equiv 5 \pmod 8$, on a $\left(\frac{2}{q}\right) = -1$, donc par le critère d'Euler (Théorème 4.12), $2^{(q-1)/2} = -1$. Ainsi $i^2 = -1$ et $x = x_0 \cdot i$ vérifie $x^2 = x_0^2 \cdot i^2 = (-a) \cdot (-1) = a$, donc x est une racine carrée de a.

Dans les deux cas, on obtient une racine carrée x de a. L'autre racine carrée est $-x$. Puisque la caractéristique de $\mathbb{F}_q$ est différente de 2, on a $x \neq -x$. En effet, si $x = -x$, alors $2x = 0$, ce qui impliquerait $x = 0$, mais $a = x^2 = 0$, contradiction avec a non nul.

De plus, les bits de parité de x et $-x$ sont opposés. En effet, si on représente les éléments de $\mathbb{F}_q$ par des entiers entre 0 et $q-1$, alors $-x$ est représenté par $q - x$. Comme q est impair, $q - x$ a la parité opposée à x. □

Définition 4.16 (Décodage des points Ed25519)**.** Le décodage d'un point compressé est l'application :

$$\mathrm{dec} : \{0,1\}^{256} \to E(\mathbb{F}_q) \cup \{\perp\}$$

définie comme suit. Soit $P_{\text{enc}} \in \{0,1\}^{256}$ un point encodé :

1. Interpréter P_{enc} comme un entier en little-endian $0 \leq N < 2^{256}$. Noter p le bit de poids fort de N (bit d'indice 255), et y l'entier obtenu en annulant ce bit, c'est-à-dire en prenant les 255 bits de poids faible. On identifie alors y avec un élément de $\{0,1\}^{255}$ et $p \in \{0,1\}$.
2. Interpréter y (les 255 bits de poids faible) comme un élément de $\mathbb{F}_q$ (en little-endian) et le noter y_P.
3. Calculer $x_P^2 = \frac{y_P^2 - 1}{dy_P^2 + 1}$ dans $\mathbb{F}_q$.
4. Si x_P^2 n'est pas un carré dans $\mathbb{F}_q$, retourner $\perp$.
5. Calculer une racine carrée x_P de x_P^2 en utilisant le théorème précédent.
6. Parmi les deux racines carrées $\{x_P, -x_P\}$, choisir celle dont le bit de parité (bit le moins significatif) est égal à p.
7. Retourner le point $P = (x_P, y_P)$.

Remarque 4.17 (Implémentation efficace de la racine carrée)**.** En pratique, on calcule d'abord :

$$x = a^{(q+3)/8}$$

puis on vérifie :

— Si $x^2 = a$, alors x est une racine carrée de a.

— Si $x^2 = -a$, alors $x \cdot 2^{(q-1)/4}$ est une racine carrée de a.

— Sinon, a n'est pas un carré dans $\mathbb{F}_q$.

On ajuste ensuite la parité en choisissant entre la racine obtenue et son opposée selon le bit p stocké dans l'encodage.

4.4 Clés Ed25519

Définition 4.18 (Clampage Ed25519)**.** Soit $H_1 \in \{0,1\}^{256}$, que l'on interprète comme une suite de 32 octets $(h_0, h_1, \ldots, h_{31})$ en little-endian. Le clampage Ed25519 est l'application :

$$\text{clamp} : \{0,1\}^{256} \to \mathbb{Z}_\ell$$

définie par les opérations suivantes sur les octets :

$$\begin{aligned} h_0 &\leftarrow h_0 \wedge \texttt{0xF8} \quad \text{(bits 0-2 à 0)}, \\ h_{31} &\leftarrow (h_{31} \wedge \texttt{0x7F}) \vee \texttt{0x40} \quad \text{(bit 255 à 0, bit 254 à 1)}, \end{aligned}$$

puis on pose

$$a = \text{val}_{\text{le}}(h_0, h_1, \ldots, h_{31}) \bmod \ell.$$

Définition 4.19 (Génération de clés Ed25519)**.** Soit $K_{\text{Ed}} \in \{0,1\}^{256}$. On calcule :

$$H = \text{SHA-512}(K_{\text{Ed}}) = H_1 \parallel H_2 \quad \text{avec } H_1, H_2 \in \{0,1\}^{256}$$

La clé privée est :

$$a = \text{clamp}(H_1) \in \mathbb{Z}_\ell$$

La clé publique est :

$$A = aB \in E(\mathbb{F}_q)$$

4.5 Algorithme de signature Ed25519

Définition 4.20 (Signature Ed25519)**.** L'algorithme de signature $\text{Sign} : \mathbb{Z}_\ell \times \{0,1\}^* \to \{0,1\}^{512}$ est défini comme suit :

Pour $(a, M) \in \mathbb{Z}_\ell \times \{0,1\}^*$:

1. Calculer $H = \text{SHA-512}(K_{\text{Ed}}) = H_1 \parallel H_2$
2. Définir $r = \text{val}_{\text{le}}(\text{SHA-512}(H_2 \parallel M)) \bmod \ell$
3. Calculer $R = rB \in E(\mathbb{F}_q)$
4. Calculer $k = \text{val}_{\text{le}}(\text{SHA-512}(\text{enc}(R) \parallel \text{enc}(A) \parallel M)) \bmod \ell$
5. Calculer $s = (r + k \cdot a) \bmod \ell$
6. La signature est $\sigma = \text{enc}(R) \parallel \text{bytes}_{\text{le}}(s)$

4.6 Algorithme de vérification Ed25519

Définition 4.21 (Vérification Ed25519)**.** L'algorithme de vérification $\text{Verify} : E(\mathbb{F}_q) \times \{0,1\}^* \times \{0,1\}^{512} \to \{\text{OK}, \perp\}$ est défini comme suit :

Pour $(A, M, \sigma) \in E(\mathbb{F}_q) \times \{0,1\}^* \times \{0,1\}^{512}$:

1. Parser $\sigma = R_{enc} \parallel s_{enc}$ avec $R_{enc}, s_{enc} \in \{0,1\}^{256}$
2. Décoder $R = \text{dec}(R_{enc}) \in E(\mathbb{F}_q)$
3. Décoder $s = \text{val}_{\text{le}}(s_{enc}) \in \mathbb{Z}_\ell$
4. Vérifier que A et R sont des points valides sur $E(\mathbb{F}_q)$
5. Vérifier que $s \in \{0, \ldots, \ell - 1\}$
6. Calculer $k = \text{val}_{\text{le}}(\text{SHA-512}(R_{enc} \parallel \text{enc}(A) \parallel M)) \bmod \ell$

7. Vérifier l'équation :
$$sB = R + kA$$
Si toutes les vérifications réussissent, retourner OK, sinon $\perp$.

Théorème 4.22 (Correction de la vérification Ed25519)**.** Pour toute paire (a, A) générée valide et tout message M,

$$\text{Verify}(A, M, \text{Sign}(a, M)) = \text{OK}$$

Démonstration. Soit $\sigma = (R, s)$ une signature valide. Alors :

$$\begin{aligned} sB &= (r + k \cdot a)B \\ &= rB + k \cdot (aB) \\ &= R + kA \end{aligned}$$

L'équation de vérification est donc satisfaite. □

Remarque 4.23 (Signature contextuelle dans le schéma k-sur-n)**.** Dans notre implémentation, on signe non pas M directement mais le message contextualisé :

$$M' = \texttt{"kofn-ed25519-v1"} \,\|\, \text{SHA256}(A) \,\|\, M$$

où A est la clé publique Ed25519. Cela empêche la réutilisation des signatures hors contexte et garantit la liaison avec la clé maîtresse du schéma.

Remarque 4.24 (Sécurité des signatures)**.** La sécurité d'Ed25519 repose sur la difficulté du problème du logarithme discret dans le groupe $\langle B \rangle$ et sur les propriétés de résistance aux collisions de SHA-512. Le clampage empêche les attaques par canaux auxiliaires et garantit que le scalaire est dans le sous-groupe principal.

5 RSA, MGF1 et OAEP

5.1 Clés RSA

Proposition 5.1 (Valeur de $\varphi(n)$ pour $n = pq$)**.** Si p et q sont deux nombres premiers distincts et $n = pq$, alors :
$$\varphi(n) = (p - 1)(q - 1)$$

Démonstration. Parmi les $n = pq$ entiers de 1 à n, les entiers qui ne sont pas premiers avec n sont :

- Les multiples de p : $p, 2p, 3p, \ldots, qp$ il y en a q
- Les multiples de q : $q, 2q, 3q, \ldots, pq$ il y en a p

L'entier pq a été compté deux fois. Par le principe d'inclusion-exclusion :

$$\varphi(n) = n - q - p + 1 = pq - p - q + 1 = (p - 1)(q - 1)$$

□

Définition 5.2 (Clés RSA)**.** On choisit deux nombres premiers p, q de taille comparable (typiquement $p, q \approx 2^{2048}$ pour un module de 4096 bits) et on pose $n = pq$.

On calcule $\varphi(n) = (p-1)(q-1)$.

On choisit un exposant public e tel que $\gcd(e, \varphi(n)) = 1$ (typiquement $e = 65537$). L'exposant privé d est l'inverse de e modulo $\varphi(n)$:

$$d \equiv e^{-1} \pmod{\varphi(n)}$$

La clé publique est (n, e) et la clé privée est (n, d).

Théorème 5.3 (Correction du chiffrement RSA)**.** Pour tout message $M \in \{0, \dots, n-1\}$ et toute clé RSA valide (n, e, d), on a :

$$(M^e)^d \equiv M \pmod{n}$$

Démonstration. Puisque $ed \equiv 1 \pmod{\varphi(n)}$, il existe k tel que $ed = 1 + k(p-1)(q-1)$.

Modulo p :

— Si $p \mid M$: alors $M \equiv 0 \pmod{p}$, donc $M^{ed} \equiv 0 \equiv M \pmod{p}$

— Si $p \nmid M$: par le petit théorème de Fermat (théorème 1.10), $M^{p-1} \equiv 1 \pmod{p}$, donc

$$M^{ed} = M^{1+k(p-1)(q-1)} = M \cdot (M^{p-1})^{k(q-1)} \equiv M \cdot 1^{k(q-1)} = M \pmod{p}$$

Ainsi, dans tous les cas, $M^{ed} \equiv M \pmod{p}$, donc $p \mid (M^{ed} - M)$.

Modulo q :

— Si $q \mid M$: alors $M \equiv 0 \pmod{q}$, donc $M^{ed} \equiv 0 \equiv M \pmod{q}$

— Si $q \nmid M$: par le petit théorème de Fermat (théorème 1.10), $M^{q-1} \equiv 1 \pmod{q}$, donc

$$M^{ed} = M^{1+k(p-1)(q-1)} = M \cdot (M^{q-1})^{k(p-1)} \equiv M \cdot 1^{k(p-1)} = M \pmod{q}$$

Ainsi, dans tous les cas, $M^{ed} \equiv M \pmod{q}$, donc $q \mid (M^{ed} - M)$.

Soit $N = M^{ed} - M$. Nous avons montré que $p \mid N$ et $q \mid N$. Puisque p et q sont premiers distincts, ils sont premiers entre eux.

Comme $p \mid N$, on peut écrire $N = p \cdot K$ pour un certain entier K. Puisque $q \mid N = p \cdot K$ et que $\gcd(p, q) = 1$, par le théorème de Gauss (théorème 1.11), $q \mid K$. Donc $K = q \cdot L$ pour un certain entier L, et ainsi :

$$N = p \cdot K = p \cdot q \cdot L = n \cdot L$$

Ce qui montre que $n \mid N$, c'est-à-dire :

$$M^{ed} \equiv M \pmod{n}$$

□

Remarque 5.4 (Risque lorsque $\gcd(M, n) \neq 1$)**.** Si $\gcd(M, n) \neq 1$, alors M est divisible par p ou par q, ce qui permettrait à un attaquant de factoriser n en calculant $\gcd(M, n)$.

Bien que théoriquement possible, cette attaque est pratiquement irréalisable :

— Pour un module RSA de 4096 bits, la probabilité qu'un message aléatoire M (de 4096 bits après encodage OAEP) ait un facteur commun avec n est d'environ 2^{-2048}

— Cette probabilité est bien inférieure à l'inverse du nombre estimé de particules dans l'univers observable ($\approx 10^{80} \approx 2^{266}$)
— Même en chiffrant un milliard de messages par seconde pendant l'âge de l'univers ($\approx 4 \times 10^{17}$ secondes), le nombre total de messages serait d'environ $10^{26} \approx 2^{86}$, et l'espérance du nombre de messages « vulnérables » serait de $2^{86} \cdot 2^{-2048} = 2^{-1962}$, ce qui reste bien inférieur à 1
— Si un tel événement se produisait par miracle, le chiffrement et le déchiffrement fonctionneraient parfaitement normalement, et personne ne s'en rendrait compte sans calculer explicitement $\gcd(M, n)$

En pratique, cette attaque n'est donc pas une préoccupation réaliste pour la sécurité de RSA avec des paramètres standard.

Définition 5.5 (Paramètres concrets pour le schéma k-sur-n)**.** Dans notre implémentation :
— Module n : 4096 bits
— Exposant public $e = 65537$
— Exposant privé $d \equiv e^{-1} \pmod{\varphi(n)}$

Remarque 5.6 (Sécurité RSA)**.** La sécurité de RSA repose sur la difficulté de la factorisation du module n. Pour un module de 4096 bits, cela offre une sécurité suffisante selon les standards actuels.

5.2 MGF1

Définition 5.7 (Fonction de génération de masque MGF1)**.** Soit $h : \{0,1\}^* \to \{0,1\}^d$ une fonction de hachage. On définit

$$\mathrm{MGF1}_h : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^\ell$$

par :

$$\mathrm{MGF1}_h(Z, \ell) = T_1 \,\|\, T_2 \,\|\, \ldots \,\|\, T_{\lceil \ell/d \rceil} \quad \text{tronqué à } \ell \text{ bits}$$

où

$$T_i = h(Z \,\|\, \mathrm{INT}_4(i-1)) \quad \text{pour } i = 1, 2, \ldots, \lceil \ell/d \rceil$$

et $\mathrm{INT}_4(j)$ est l'encodage big-endian de j sur 4 octets.

Propriété 5.8 (Propriétés de MGF1)**.** MGF1 est déterministe et peut générer des masques de longueur arbitraire. Sa sécurité repose sur les propriétés de résistance aux collisions et de préimage de la fonction de hachage sous-jacente h.

Remarque 5.9 (Rôle crucial de MGF1 dans OAEP)**.** MGF1 joue un rôle essentiel dans OAEP en empêchant les attaques par blocs indépendants :
— **Transformation de taille** : Passer d'une entrée fixe à une sortie de longueur arbitraire
— **Distribution uniforme** : Garantir que le masque n'a pas de motifs détectables
— **Non-corrélation** : Assurer que chaque partie du masque est unique grâce au compteur

— **Entrelacement cryptographique** : Créer des dépendances non linéaires entre `maskedSeed` et `maskedDB` via la structure :

$$\begin{cases} \texttt{maskedDB} = \texttt{DB} \oplus \mathrm{MGF1}_h(\texttt{seed}) \\ \texttt{maskedSeed} = \texttt{seed} \oplus \mathrm{MGF1}_h(\texttt{maskedDB}) \end{cases}$$

— **Prévention d'attaques par blocs** : Toute modification d'un bit dans une partie affecte de manière imprévisible l'ensemble du message, rendant impossible les attaques ciblées sur des blocs individuels

— **Sécurité prouvée** : Permettre les preuves formelles de sécurité d'OAEP contre les attaques adaptatives (IND-CCA2)

Sans MGF1, un attaquant pourrait manipuler séparément les différentes parties du message encodé, réduisant la sécurité à celle de schémas de padding vulnérables comme PKCS#1 v1.5. MGF1 est donc la « colle cryptographique » qui assure l'indissociabilité des composants d'OAEP.

5.3 RSA-OAEP

Définition 5.10 (Paramètres RSA-OAEP)**.** Soient k la taille en octets du module n (pour n de 4096 bits, $k = 512$), et $hLen$ la taille de sortie de la fonction de hachage en octets (pour SHA-256, $hLen = 32$). On fixe :

$$k_0 = k - 2hLen - 2, \quad k_1 = hLen$$

où k_0 est la longueur maximale du message en octets et k_1 la longueur de l'aléa r.

Définition 5.11 (Encodage OAEP)**.** L'encodage OAEP pour un message $M \in \{0,1\}^{8k_0}$, un label $L \in \{0,1\}^*$ et un aléa $r \in \{0,1\}^{8k_1}$ est défini comme suit :

1. Calculer $lHash = h(L)$
2. Former la chaîne $PS = 0^{8(k-k_0-2hLen-2)}$ (padding de zéros)
3. Construire le bloc $DB = lHash \,\|\, PS \,\|\, 0x01 \,\|\, M$
4. Calculer $maskedDB = DB \oplus \mathrm{MGF1}_h(r, 8(k - hLen - 1))$
5. Calculer $maskedSeed = r \oplus \mathrm{MGF1}_h(maskedDB, 8hLen)$
6. Le message encodé est $EM = maskedSeed \,\|\, maskedDB \in \{0,1\}^{8k}$

Définition 5.12 (Chiffrement RSA-OAEP)**.** Soient (n, e) une clé publique RSA, k_0, k_1 les paramètres de taille définis ci-dessus. On définit :

$$\begin{aligned} \text{RSA-OAEP}_{(n,e)} &: \{0,1\}^{8k_0} \times \{0,1\}^{8k_1} \to \{0, \ldots, n-1\} \\ \text{RSA-OAEP}_{(n,e)}(M; r) &= \mathrm{val}_{\mathrm{be}}(EM)^e \bmod n \end{aligned}$$

où EM est le résultat de l'encodage OAEP de M avec l'aléa r.

Définition 5.13 (Déchiffrement RSA-OAEP)**.** Soient (n, d) une clé privée RSA. On définit :

$$\text{RSA-OAEP}^{-1}_{(n,d)} : \{0, \ldots, n-1\} \to \{0,1\}^{8k_0} \cup \{\perp\}$$

$$\text{RSA-OAEP}^{-1}_{(n,d)}(C) = \begin{cases} M & \text{si le décodage réussit} \\ \perp & \text{sinon} \end{cases}$$

où $C \in \{0, \ldots, n-1\}$ est le **texte chiffré** (ciphertext) obtenu par $\text{RSA-OAEP}_{(n,e)}(M; r)$.

Le déchiffrement procède comme suit :

1. Calculer $EM = C^d \bmod n$ et convertir en binaire sur $8k$ bits
2. Parser $EM = maskedSeed \,\|\, maskedDB$ avec $maskedSeed \in \{0,1\}^{8hLen}$
3. Calculer $seed = maskedSeed \oplus \mathrm{MGF1}_h(maskedDB, 8hLen)$
4. Calculer $DB = maskedDB \oplus \mathrm{MGF1}_h(seed, 8(k - hLen - 1))$
5. Parser $DB = lHash' \,\|\, PS \,\|\, 0x01 \,\|\, M$ où PS est une suite de zéros
6. Vérifier que $lHash' = h(L)$ et que PS contient bien que des zéros
7. Si toutes les vérifications passent, retourner M, sinon $\perp$

Remarque 5.14 (Statut de l'aléa r dans OAEP)**.** Contrairement aux sels publics utilisés dans HKDF ou PBKDF2, l'aléa r dans OAEP est **cryptographiquement protégé** :

— **Non public** : r n'est pas stocké en clair ni transmis publiquement

— **Masqué cryptographiquement** : Il est caché dans le chiffré via $\texttt{maskedSeed} = r \oplus \mathrm{MGF1}_h(\texttt{maskedDB})$

— **Récupération au déchiffrement** : Seul le possesseur de la clé privée peut retrouver r

— **Rôle éphémère** : r est utilisé une seule fois puis "jeté"

La sécurité d'OAEP repose sur le fait que r reste **imprévisible** pour un attaquant au moment du chiffrement.

Remarque 5.15 (Paramètres numériques pour RSA-4096 et SHA-256)**.** Pour un module RSA de 4096 bits ($k = 512$ octets) avec SHA-256 ($hLen = 32$ octets), on a :

$$k_0 = 512 - 2 \times 32 - 2 = 446 \text{ octets}$$

Cette capacité de 446 octets est amplement suffisante pour une clé AES-256 de 32 octets, avec une marge importante pour les données de padding.

Théorème 5.16 (Correction de RSA-OAEP)**.** *Pour toute clé (n, e, d) valide, pour tout message M de longueur $8k_0$ bits et pour tout aléa $r \in \{0,1\}^{8k_1}$,*

$$\text{RSA-OAEP}^{-1}_{(n,d)}\big(\text{RSA-OAEP}_{(n,e)}(M; r)\big) = M.$$

Démonstration. La correction découle de la structure réversible de l'encodage OAEP. Les opérations de masquage utilisant MGF sont réversibles car déterministes, et les vérifications assurent l'intégrité du message. □

Remarque 5.17 (Propriétés de sécurité)**.** RSA-OAEP offre une sécurité prouvée dans le modèle de l'oracle aléatoire contre les attaques adaptatives à chiffrés choisis (IND-CCA2). Le padding OAEP prévient les attaques par canal auxiliaire et garantit l'intégrité du message.

Définition 5.18 (Utilisation dans le schéma k-sur-n)**.** Dans notre contexte, RSA-OAEP est utilisé pour chiffrer des clés AES-256 de 32 octets. Pour un module RSA de 4096 bits ($k = 512$) avec SHA-256 ($hLen = 32$), on a :

$$k_0 = 512 - 2 \times 32 - 2 = 446 \text{ octets}$$

Ce qui est amplement suffisant pour une clé AES-256 de 32 octets, avec une marge importante de 414 octets pour le padding OAEP.

6 AES-GCM

6.1 AES comme permutation de bloc

Définition 6.1 (AES)**.** Pour chaque clé $K \in \{0,1\}^{128} \cup \{0,1\}^{192} \cup \{0,1\}^{256}$, on dispose d'une permutation de bloc

$$E_K : \{0,1\}^{128} \to \{0,1\}^{128},$$

bijective avec réciproque $D_K = E_K^{-1}$.

6.2 Corps $\mathbb{F}_{2^{128}}$ et GHASH

Définition 6.2 (Corps $\mathbb{F}_{2^{128}}$)**.** On fixe un polynôme irréductible $P(X) \in \mathbb{F}_2[X]$ de degré 128 (par exemple $X^{128} + X^7 + X^2 + X + 1$) et on définit

$$\mathbb{F}_{2^{128}} = \mathbb{F}_2[X]/(P(X)).$$

On fixe une bijection entre $\{0,1\}^{128}$ et $\mathbb{F}_{2^{128}}$ en identifiant le bloc $(b_{127}, \ldots, b_0)$ au polynôme $\sum_{i=0}^{127} b_i X^i \bmod P(X)$.

Définition 6.3 (Sous-clé de hachage H)**.** Pour une clé AES K, on définit

$$H = E_K(0^{128}) \in \{0,1\}^{128},$$

que l'on identifie à un élément de $\mathbb{F}_{2^{128}}$.

Définition 6.4 (GHASH)**.** Soit $H \in \mathbb{F}_{2^{128}}$. Pour une séquence $(Y_1, \ldots, Y_m)$ de blocs de $\{0,1\}^{128}$, que l'on identifie à $\mathbb{F}_{2^{128}}$, on définit

$$\mathrm{GHASH}_H(Y_1, \ldots, Y_m) = \sum_{j=1}^{m} H^{m-j+1} \cdot Y_j \in \mathbb{F}_{2^{128}}.$$

Cette expression correspond à l'itération $((Y_1 H + Y_2)H + \cdots + Y_m)H$.

6.3 Mode AES-GCM

Définition 6.5 (AES-GCM)**.** Pour une clé $K \in \{0,1\}^{256}$, on modélise AES-GCM comme un couple d'applications

$$\begin{aligned} \text{AES-GCM}_K &: \{0,1\}^{96} \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^{128} \\ \text{AES-GCM}_K^{-1} &: \{0,1\}^{96} \times \{0,1\}^* \times \{0,1\}^{128} \to \{0,1\}^* \cup \{\perp\} \end{aligned}$$

où le premier argument est un nonce $N \in \{0,1\}^{96}$, le deuxième un message M, la sortie un couple (C, T) avec C le texte chiffré et T le tag (128 bits), et où la fonction inverse produit soit le message M, soit le symbole d'erreur $\perp$ (hors de $\{0,1\}^*$) si la vérification du tag échoue.

Remarque 6.6 (Unicité des nonces)**.** Pour toute clé K, les nonces N doivent être uniques. Formellement, si $(N, M) \neq (N', M')$ alors les ensembles de textes chiffrés générés doivent être disjoints avec probabilité écrasante.

Théorème 6.7 (Correction d'AES-GCM)**.** Pour toute clé $K \in \{0,1\}^{256}$, tout nonce $N \in \{0,1\}^{96}$ et tout message $M \in \{0,1\}^*$,

$$\text{AES-GCM}_K^{-1}\big(N,\ \text{AES-GCM}_K(N, M)\big) = M.$$

7 Construction globale du schéma k-sur-n

7.1 Paramètres et constantes

Définition 7.1 (Paramètres globaux)**.** On fixe :
- un entier $n \geq 2$ (nombre de participants) ;
- un entier k avec $2 \leq k \leq n$ (seuil) ;
- le corps $\mathbb{F}_P$ avec $P = 2^{521} - 1$;
- la fonction de hachage h (par exemple SHA-256) ;
- les fonctions dérivées HMAC_h, HKDF_h, $\mathrm{PBKDF2}_h$;
- un schéma de signature Ed25519 sur $(E(\mathbb{F}_q), \langle B \rangle)$;
- un schéma RSA-OAEP pour des modules de taille 4096 bits ;
- AES-GCM avec blocs de 128 bits et clés de 256 bits.

Définition 7.2 (Constantes de contexte et sels)**.** On fixe les constantes suivantes :
- $\mathrm{info}_{\mathrm{Ed}} \in \{0,1\}^*$, chaîne ASCII (par exemple `"ed25519-master-key"`) ;
- $\mathrm{info}_{\mathrm{RSA}} \in \{0,1\}^*$, (par exemple `"rsa-wrap-key"`) ;
- un sel HKDF public pour Ed25519, $\mathrm{saltEd} \in \{0,1\}^*$, typiquement l'ASCII de `"ed25519-salt"` ;
- pour chaque cérémonie, un sel HKDF public pour RSA, $W \in \{0,1\}^{\ell_{\mathrm{wrap}}}$, tiré uniformément ;
- pour chaque participant i, un sel PBKDF2 individuel $S_i \in \{0,1\}^{\ell_S}$ (avec ℓ_S fixé, par exemple 128 bits), tiré uniformément ;
- un nombre d'itérations $c \in \mathbb{N}^*$ pour PBKDF2 ;
- pour RSA-OAEP, un label $L \in \{0,1\}^*$ (souvent vide).

7.2 Cérémonie initiale

Définition 7.3 (Encodage des éléments de $\mathbb{F}_P$)**.** Soit $\phi : \mathbb{F}_P \to \{0,1\}^{521}$ l'isomorphisme qui à $a \in \mathbb{F}_P$ associe sa représentation binaire big-endian sur $\lceil \log_2 P \rceil = 521$ bits.

Définition 7.4 (Étape de génération)**.** La cérémonie initiale réalise les étapes suivantes :
1) Tirer $S \in \{0,1\}^{256}$ uniformément et définir $s = \mathrm{val}_{\mathrm{be}}(S) \bmod P \in \mathbb{F}_P$.
2) Appliquer $\mathrm{Share}(s)$ pour obtenir les parts $(x_i, y_i)_{1 \leq i \leq n}$.
3) Dériver la clé
$$K_{\mathrm{Ed}} = \mathrm{HKDF}_h(S, \mathrm{saltEd}, \mathrm{info}_{\mathrm{Ed}}, 32) \in \{0,1\}^{256}.$$
4) Dériver à partir de K_{Ed} la paire de clés Ed25519 (a, A).
5) Générer une paire RSA (n, e, d) avec n de 4096 bits.
6) Tirer $W \in \{0,1\}^{\ell_{\mathrm{wrap}}}$ et dériver la clé
$$K_{\mathrm{RSA}} = \mathrm{HKDF}_h(S, W, \mathrm{info}_{\mathrm{RSA}}, L_{\mathrm{RSA}}) \in \{0,1\}^{8L_{\mathrm{RSA}}}$$
avec $L_{\mathrm{RSA}} = 32$ (clé AES-256 de 32 octets).
7) Encoder la clé privée RSA en un mot binaire M_{RSA} (format PEM ou DER), puis calculer
$$(C_{\mathrm{RSA}}, T_{\mathrm{RSA}}) = \text{AES-GCM}_{K_{\mathrm{RSA}}}(N_{\mathrm{RSA}}, M_{\mathrm{RSA}})$$
pour un nonce $N_{\mathrm{RSA}} \in \{0,1\}^{96}$.
8) Stocker $(W, N_{\mathrm{RSA}}, C_{\mathrm{RSA}}, T_{\mathrm{RSA}})$ dans `rsa_wrapped.json`.

7.3 Protection des parts

Définition 7.5 (Sérialisation des parts)**.** Pour chaque part (x_i, y_i), on construit un dictionnaire JSON contenant les champs `x`, `y`, `k`, `n`, et `pub_hash`. On encode ce dictionnaire en UTF-8 pour obtenir le message à chiffrer $M_i \in \{0,1\}^*$.

Définition 7.6 (Dérivation des clés individuelles)**.** Pour chaque participant i, on fixe un mot de passe $P_i \in \{0,1\}^*$ et on définit

$$K_i = \mathrm{PBKDF2}_h(P_i, S_i, c, L_i) \in \{0,1\}^{8L_i},$$

où L_i est choisi en fonction de la clé AES à dériver (par exemple $L_i = 32$ pour une clé AES-256).

Définition 7.7 (Chiffrement des parts)**.** Pour une part sérialisée M_i, on calcule

$$(C_i, T_i) = \text{AES-GCM}_{K_i}(N_i, M_i)$$

avec un nonce $N_i \in \{0,1\}^{96}$. Le tuple (S_i, N_i, C_i, T_i) est stocké dans la partie personnelle du coffre du participant i.

7.4 Chiffrement hybride des fichiers

Définition 7.8 (Chiffrement hybride d'un fichier)**.** Soit un fichier $F \in \{0,1\}^*$. On procède comme suit :

1) Tirer une clé de session $K_{\text{AES}} \in \{0,1\}^{256}$.
2) Tirer un nonce $N \in \{0,1\}^{96}$.
3) Calculer $(C, T) = \text{AES-GCM}_{K_{\text{AES}}}(N, F)$.
4) Calculer $E_K = \text{RSA-OAEP}_{(n,e)}(K_{\text{AES}}; r)$ pour un aléa $r \in \{0,1\}^{k_1}$.

Le quadruplet (E_K, N, C, T) est le chiffrement hybride de F.

8 Correspondance avec les scripts Tails v1

Cette section établit la correspondance entre les objets formalisés ci-dessus et les éléments concrets (scripts, fichiers) de la procédure Tails v1.

8.1 Secret maître et partage Shamir

— Le secret $S \in \{0,1\}^{256}$ est généré par `os.urandom(32)` dans le script `ceremony_generate.py` (variable `S_int`).

— La conversion $s = \mathrm{val}_{\mathrm{be}}(S) \bmod P \in \mathbb{F}_P$ est implicite dans l'implémentation Python.

— Les partages (x_i, y_i) sont obtenus par évaluation d'un polynôme $f \in \mathcal{P}_{<k}$ via la fonction `shamir_split`.

— La reconstruction utilise l'interpolation de Lagrange dans `shamir_reconstruct` avec calcul explicite des coefficients λ_j.

8.2 Clés dérivées, sels HKDF et Ed25519

— La clé $K_{\mathrm{Ed}} = \mathrm{HKDF}_h(S, \mathrm{saltEd}, \mathrm{info}_{\mathrm{Ed}}, 32)$ est instanciée via :

```
HKDF(SHA256, 32, salt=b"ed25519-salt",
     info=b"ed25519-master-key")
```

— De même, $K_{\mathrm{RSA}} = \mathrm{HKDF}_h(S, W, \mathrm{info}_{\mathrm{RSA}}, 32)$ utilise `info=b"rsa-wrap-key"` avec un sel aléatoire W stocké dans `rsa_wrapped.json`.

— La paire (a, A) est construite via `Ed25519PrivateKey.from_private_bytes(ed_seed)`, où `ed_seed` est K_{Ed}. L'API effectue le clampage et calcule $A = aB$.

8.3 PBKDF2, sels S_i et chiffrement des parts

— Pour chaque participant i, le script tire un sel S_i (16 octets) et dérive $K_i = \mathrm{PBKDF2}_h(P_i, S_i, c, 32)$ avec $c = 501000$ itérations.

— Les parts sont sérialisées en JSON avec les champs `x`, `y`, `k`, `n`, et `pub_hash`, puis chiffrées par AES-GCM.

— L'enveloppe chiffrée inclut les métadonnées nécessaires pour la vérification lors de la reconstruction.

8.4 RSA, OAEP et chiffrement hybride

— La paire RSA (n, e, d) est générée avec $n = 4096$ bits et $e = 65537$.

— La clé privée RSA est sérialisée en PEM puis chiffrée sous AES-GCM avec K_{RSA}, produisant `rsa_wrapped.json`.

— Le script `rsa_hybrid_encrypt.py` implémente le chiffrement hybride :

  — Génération de $K_{\mathrm{AES}} \in \{0, 1\}^{256}$

  — Chiffrement de F par $\mathrm{AES\text{-}GCM}_{K_{\mathrm{AES}}}(N, F)$

  — Chiffrement de K_{AES} par $\mathrm{RSA\text{-}OAEP}_{(n,e)}$

— Le déchiffrement dans `kofn_rsa_decrypt.py` inverse ces étapes après reconstruction de S et déverrouillage de la clé RSA.

8.5 Signature Ed25519 avec contexte

— Les signatures utilisent un contexte déterministe :

$$\mathrm{contexte} = \texttt{"kofn-ed25519-v1"} \parallel \mathrm{SHA256}(A) \parallel M$$

pour éviter la réutilisation hors contexte.

— La vérification dans `verify_ed25519.py` utilise le même contexte, assurant l'authenticité même avec la même clé publique.

8.6 Gestion sécurisée de la mémoire

— Les secrets éphémères (S, seeds) sont stockés dans des `bytearray` pour permettre l'effacement explicite via `secure_wipe`.

— Cette mesure atténue les risques d'exposition en RAM après usage.

8.7 Résumé sur les sels publics

- Sel HKDF Ed25519 saltEd :
  - valeur : chaîne ASCII constante (par exemple `"ed25519-salt"`) ;
  - génération : aucun tirage, valeur codée en dur dans le script ;
  - stockage : implicite dans le code, public et identique pour toutes les cérémonies.
- Sel HKDF RSA (wrap salt) W :
  - valeur : bitstring aléatoire `wrap_salt` généré par `os.urandom(WRAP_SALT_SIZE)` ;
  - génération : une fois par cérémonie de génération de la clé RSA ;
  - stockage : champ `"salt"` dans `rsa_wrapped.json`, encodé en Base64, public.
- Sels PBKDF2 S_i :
  - valeur : bitstrings aléatoires individuels, un par participant, générés par `os.urandom(SALT_SIZ`
  - génération : lors de la création ou de la mise à jour de la protection de la part du participant i ;
  - stockage : dans la partie personnelle du coffre du participant, aux côtés de (N_i, C_i, T_i), public mais lié au participant.

9 Résumé des propriétés de sécurité

Définition 9.1 (Modèle de sécurité)**.** On considère un adversaire $\mathcal{A}$ ayant accès aux oracles :

- $\mathcal{O}_{\text{Share}}$: renvoie jusqu'à $t < k$ parts
- $\mathcal{O}_{\text{Sign}}$: signe des messages avec la clé Ed25519
- $\mathcal{O}_{\text{Decrypt}}$: déchiffre des textes chiffrés

Théorème 9.2 (Confidentialité conditionnelle)**.** Soit $\mathcal{A}$ un adversaire polynomial n'ayant accès qu'à $t < k$ parts. Alors pour toute fonction f calculable en temps polynomial :

$$|\Pr[\mathcal{A}(f(S)) = 1] - \Pr[\mathcal{A}(f(U)) = 1]| \leq \epsilon(\lambda)$$

où U est uniforme dans $\{0,1\}^{256}$ et ϵ est négligeable dans le paramètre de sécurité λ.

Théorème 9.3 (Intégrité des signatures et des chiffrés)**.** Sous les hypothèses que HKDF, PBKDF2, AES-GCM et RSA-OAEP sont cryptographiquement sûrs, il est computationnellement difficile pour un adversaire polynomial de forger une signature Ed25519 valide ou de modifier un texte chiffré AES-GCM sans être détecté.

Remarque 9.4. Une preuve complète de sécurité nécessiterait un modèle d'adversaire précis (oracles, ressources de calcul) et l'utilisation de techniques de réduction dans des modèles comme l'oracle aléatoire. On ne la développe pas ici.