PROJ104 - Threshold re-encryption

Gabrielle Lalou, Alexis Mellier, Clémence Audibert

2024

Merci à Matthieu Rambaud pour son aide à la réalisation de ce projet.

Table des matières

Introduction

La sécurité des communications et le stockage sécurisé des données sont des défis majeurs dans le domaine de la cryptographie. Les algorithmes basés sur le problème de Learning With Errors (LWE) et le partage de secrets de Shamir ont émergé comme des solutions robustes pour assurer la confidentialité et l'intégrité des informations. Dans ce document, nous explorons une approche qui combine ces deux techniques cryptographiques afin de réaliser un système de ré-encryption basé sur des seuils.

Notre principal objectif est de répondre au problème suivant : « Une société doit sécuriser le mot de passe de son coffre-fort. Elle pourrait utiliser un moyen standard, mais que faire si la détentrice ou le détenteur de la clé n'est pas disponible ou meurt ? Que se passe-t-il si la clé est compromise par un pirate malveillant ou si le détenteur de la clé décide de trahir la société, et utilise son pouvoir sur le coffre à son avantage ? » Pour garantir la sécurisation de ce mot de passe, n collègues [1] de la société ont accès à ce qu'on appelle un "share", une partie du mot de passe. Il faut t + 1 shares pour reconstruire ce mot de passe, de telle sorte qu'un nombre de shares inférieur à t ne permet pas d'avoir accès à celui-ci.

Dans les deux premières sections, nous donnons le socle mathématique nécessaire à l'implémentation. On pourra y trouver les preuves de t-privacy, ou encore la formule de reconstruction à partir de shares.

Dans les deux suivantes, on étudie les cas 0 et 1-privacy pour l'implémentation, en passant par les définitions des fonctions Encryption, Decryption, et de la forme affine correspondant au système de machines.

1. A partir de là, nous dirons n "machines" plutôt que "collègues" par soucis de simplicité.

1 Objectif 1 : t-privacy

Dans cette section, nous nous appuyons sur l'algorithme randomisé de Shamir [2].
Notre objectif ici est de répondre au problème de privacy énoncé dans l'introduction, pour t un entier de $[\![1, n]\!]$ à définir. Pour ce faire, nous avons besoin d'un constructeur pour ces shares. Ici, nous montrerons que si ce constructeur est choisi de manière uniforme, alors ces shares le seront aussi.
Soit q un nombre premier.
Fixons $s \in \mathbb{F}_q$ notre secret, n le nombre de shares qu'on veut transmettre et $t \leq n$ un entier. Fixons également $\{i_1, ..., i_t\} \subset [\![1, n]\!]$. On veut montrer que, si f varie uniformément dans $\{P \in \mathbb{F}_{q,t}[X] | P(0) = 0\}$ (qu'on assimilera à $\mathbb{F}_q^t$), alors $(g(i_1), ..., g(i_t))$ (où $g := f + s$) varie uniformément dans $\mathbb{F}_q^t$.

Lemme : Soit E un espace vectoriel de dimension t et h une application affine surjective de E.

Soit X suivant la loi uniforme dans E.

Alors $h(X)$ suit aussi la loi uniforme.

Démonstration. Fixons $A \in \mathcal{M}_t(\mathbb{F}_q)$ et $w \in E$ tels que

$$\forall u \in \mathbb{F}_q^t, h(u) = Au + w.$$

h est surjective donc A est inversible. Soit $\mathbb{P}$ la loi de probabilité dont E est muni et $v \in E$. On a

$$\mathbb{P}(h(X) = v) = \mathbb{P}(AX = v - w) = \mathbb{P}(X = A^{-1}(v - w)).$$

X suivant la loi de probabilité uniforme, $h(X)$ la suit donc aussi.

□

Posons

$$V = \begin{pmatrix} i_1 & ... & i_1^t \\ . & & . \\ . & & . \\ i_t & ... & i_t^t \end{pmatrix}$$

et $h : \begin{array}{ccc} \mathbb{F}_q^t & \to & \mathbb{F}_q^t \\ (a_1, ..., a_t) & \mapsto & V \begin{pmatrix} a_1 \\ \vdots \\ a_t \end{pmatrix} + \begin{pmatrix} s \\ \vdots \\ s \end{pmatrix} \end{array}$

Théorème : Soit G suivant la loi uniforme dans $\mathbb{F}_q^t$.
Alors $h(G)$ suit cette même loi uniforme.

Démonstration. h est affine. Montrons que V est inversible, on en déduira que h est surjective, ce qui suffira à conclure d'après le lemme.
Par l'absurde, fixons $(\lambda_1, ..., \lambda_t) \neq (0, ..., 0)$ tel que

$$\sum_{k=1}^{t} \lambda_k \begin{pmatrix} i_k \\ \vdots \\ i_k^t \end{pmatrix} = 0.$$

Alors

$$\sum_{k=1}^{t} \lambda_k i_k \begin{pmatrix} 1 \\ i_k \\ \vdots \\ i_k^{t-1} \end{pmatrix} = 0,$$

or $i_k \neq 0$ (car $i_k \in [\![1, n]\!]$).
Absurde car

$$\begin{pmatrix} 1 & ... & 1 \\ i_1 & ... & i_t \\ . & & . \\ . & & . \\ i_1^{t-1} & ... & i_t^{t-1} \end{pmatrix}$$

est inversible en tant que matrice de Vandermonde (avec les i_k deux à deux distincts). Donc V est bien inversible, ce qui conclut. □

Conclusion : Dans notre code `secret_sharing.rs`, la génération du polynôme par loi uniforme implique donc des shares uniformes : n'importent quels t shares parmi les n envoyés varient uniformément. Il y a donc une t-privacy.

2 Objectif 2 : (t+1)-reconstruction

2.1 (t+1)-reconstruction par une forme linéaire

Fixons $s \in \mathbb{F}_q$ [2] notre secret, n le nombre de shares que l'on veut transmettre et $t \leq n$ un entier. Fixons également $\{i_1, ..., i_{t+1}\} \subset [\![1, n]\!]$ représentant les indices de t+1 shares de s.

Soit $\begin{array}{ccccc} g & : & \mathbb{F}_q & \to & \mathbb{F}_q \\ & & x & \mapsto & g(x) \end{array}$ une fonction de $\mathbb{F}_{q,t}[X]$ tel que $g(0) = s$.

Pour plus de lisibilité, on peut poser $g(m) = s_m$ pour tout $m \in [\![0, n]\!]$.
Considérant le vecteur v $= (s_{i_1}, ..., s_{i_{t+1}})$ qui contient t+1 shares de s, nous voulons déduire une forme linéaire qui prend en entrée le vecteur v et qui output s. Étant donné que nous avons t+1 shares de s, nous pouvons utiliser l'interpolation de Lagrange pour trouver s.

2. En réalité $s \in \mathbb{F}_q^d$, et on réalise ces étapes d fois sur chacun des coefficients de s.

Soit Q l'application suivante :

$$\begin{array}{rccc} Q & : & \mathbb{F}_q^{t+1} & \to & \mathbb{F}_q \\ & & (y_1, ..., y_{t+1}) & \mapsto & \sum_{k=1}^{t+1} y_k \left(\prod_{l=1, l\neq k}^{t+1} \frac{-i_l}{i_k - i_l} \right) \end{array}$$

Propriété : Q est linéaire.
De plus, on rappelle qu'on a posé
- $s \in \mathbb{F}_q$
- $g \in \mathbb{F}_{q,t}[X]$ tel que $g(0) = s$
- $\forall m \in [\![0, n]\!], s_m = g(m)$ et $v = (s_{i_1}, ..., s_{i_{t+1}})$.

On obtient alors : $Q(v) = s$

Démonstration. On retrouve bien s via l'interpolation de Lagrange de g.
Montrons que Q est une forme linéaire :
Soient $x = (x_1, ..., x_{t+1})$ et $y = (y_1, ..., y_{t+1})$ deux vecteurs de $\mathbb{F}_q^{t+1}$, et α une constante de $\mathbb{F}_q$. Ainsi,
$Q(x + \alpha y)$
$= \sum_{k=1}^{t+1} (x_k + \alpha y_k) \left(\prod_{l=1, l\neq k}^{t+1} \frac{-i_l}{i_k - i_l} \right)$
$= \sum_{i=1}^{t+1} x_i \left(\prod_{l=1, l\neq k}^{t+1} \frac{-i_l}{i_k - i_l} \right) + \alpha \sum_{i=1}^{t+1} y_i \left(\prod_{l=1, l\neq k}^{t+1} \frac{-i_l}{i_k - i_l} \right)$
$= Q(x) + \alpha Q(y).$ □

Conclusion : Ainsi l'algorithme `secret_sharing.rs` peut bien reconstituer le polynôme de départ à partir d'(au moins) $t + 1$ shares.

2.2 Simulabilité des (t+1)-reconstruction shares honnêtes

Dans cette section, considérons que les t premières machines soient "corrompues". Ainsi, on aurait accès à leurs shares $\{s_{i_1}, ..., s_{i_t}\}$ et à leur état initial $\{i_1, ..., i_t\}$. Ce que l'on veut montrer est le fait qu'à partir de ces données et à l'aide d'un simulateur, on ne peut rien apprendre de plus que l'état initial des n-t machines "honnêtes" restantes.
Formellement :
Soit $\{i_1, ..., i_t\} \subset [\![1, n]\!]$ un sous-ensemble d'indices.
Soit w $= \{s, s_{i_1}, ..., s_{i_t}\}$ un ensemble contenant s et n'importe quel t-share de s.
On fixe m tel que s_m représente le m-ième share où $m \in [\![1, n]\!]$.

Soit L_m l'application suivante :

$$\begin{array}{rccc} L_m & : & \mathbb{F}_q^{t+1} & \to & \mathbb{F}_q \\ & & (y_0, ..., y_t) & \mapsto & \sum_{k=0}^{t} y_k \left(\prod_{l=0, l\neq k}^{t} \frac{m - i_l}{i_k - i_l} \right) \end{array} \quad \text{où } i_0 = 0.$$

Propriété : L_m est linéaire.
De plus, on rappelle qu'on a posé
- $g \in \mathbb{F}_{q,t}[X]$ et $\forall m \in [\![0,n]\!], s_m := g(m)$ (avec $s_0 = s$)
- $w = \{s, s_{i_1}, ..., s_{i_t}\}$.

On obtient alors $L_m(w) = s_m$ pour tout $m \in [\![1,n]\!]$.

Démonstration. Immédiat par le théorème d'interpolation de Lagrange sur un polynôme de degré t. Et la preuve de la linéarité se fait comme à la propriété précédente. □

3 Objectif 3 : n=1 machine

Pour rappeler le contexte, Alice souhaite envoyer un message à Bob. Pour ce faire, Alice envoie le ciphertext c dans un système qui réalisera l'application EoD, pour enfin envoyer le nouveau ciphertext c' chiffré sous p_B à Bob. Ainsi, ce que le système reçoit en entrée est c, la clé secrète s ; ce système génère de plus pendant la réencryption sous p_B un aléa (e1',e2', r'), qui correspond à du bruit.
C'est la raison pour laquelle, pour simplifier l'implémentation du code correspondant à l'application EoD, nous souhaitons qu'elle soit affine en $(s, e1', e2', r')$. Ainsi, nous voulons que EoD soit de la forme $EoD(X) = MX + N$ où $X = (s^\top, e1', e2', r')^\top$.

Soient m un message binaire dans {0,1}, q un entier positif tel que $\Delta = q/2$ si q est pair ($\Delta = \lfloor q/2 \rfloor$ sinon), et $d \in \mathbb{N}^*$ un entier non nul fixé.
Alice souhaite envoyer ce message à Bob.

Remarque : Dans toute la suite, on assimilera $\mathbb{Z}/q\mathbb{Z}$ à $[-q/2, q/2)$. Ainsi, $x \bmod(q)$ sera l'unique $x' \in [-q/2, q/2)$ tel que $x' = x \bmod(q)$.

3.1 Génération des paires de clés

Soit la fonction Key-generation suivante, inspirée de l'article [1] :
Du coté d'Alice, on génère aléatoirement
- $A \in \mathcal{M}_d(\mathbb{Z}/q\mathbb{Z})$ matrice uniforme
- E [3], $s \in \mathcal{M}_{1,d}(\mathbb{Z}/q\mathbb{Z})$ à partir de distributions gaussiennes.

On pose ensuite $B = sA + E \in \mathcal{M}_{1,d}(\mathbb{Z}/q\mathbb{Z})$ et $p = (A, B)$.
De même pour Bob, on génère aléatoirement
- $A' \in \mathcal{M}_d(\mathbb{Z}/q\mathbb{Z})$ matrice uniforme
- $E', s_B \in \mathcal{M}_{1,d}(\mathbb{Z}/q\mathbb{Z})$ à partir de distributions gaussiennes.

On pose ensuite $B' = s_B A' + E' \in \mathcal{M}_{1,d}(\mathbb{Z}/q\mathbb{Z})$ et $p_B = (A', B')$.

3. $E \sim \mathcal{N}(0, \overline{\Psi}_\alpha^2)$, où $\overline{\Psi}_\alpha$ correspond au paramètre de Regev dans l'article [1]

Remarque : Tout ceci est implémenté dans le code `gaussian.rs`. Certaines variables sont toutefois générées à partir de d'autres distributions pour implémenter la Key-generation d'autres articles.

3.2 Fonction Encryption

Pour qu'Alice envoie m, elle l'encode en Δm puis chiffre Δm avec sa clé publique p. Ainsi, pour la fonction *Encryption* citée ci-dessus, et implémentée dans notre code `lwe_functions.rs`, nous posons E, l'application correspondant à l'action d'Alice après l'encodage :

$$Encryption(p, m) = E(p, \Delta m)$$

Soit :

$$E(p, \Delta m) = c = (c1, c2) = (Ar + e_1, Br + e_2 + \Delta m)$$

La première étape de cet objectif est faite : Alice applique Encryption à son message m à l'aide de sa clé publique p.

Remarque : La fonction *Encryption* génère $e_1 \in \mathcal{M}_{d,1}(\mathbb{Z}/q\mathbb{Z})$, $e_2 \in \mathbb{Z}/q\mathbb{Z}$, et $r \in \mathcal{M}_{d,1}(\mathbb{Z}/q\mathbb{Z})$.

3.3 Fonction D

A cette étape, le cipher-text est envoyé dans un système qui va le déchiffrer à l'aide de la clé privée d'Alice s, et qui va le chiffrer à nouveau par la clé publique de Bob p_B.

3.3.1 Expression de D

Déchiffrement : On appelle D la méthode qui nous sert à déchiffrer c.
$D(s, c) = c2 - sc1$
= Br+e2+Δm-s.(Ar+e1)
= (sA+E)r+e2+Δm-s(Ar+e1)
= E.r+e2+Δm-se1
= Δm+Er+e2-se1
= Δm+e où on a posé e=Er+e2-se1.

On pose alors $résultat := \Delta m + e$.

La deuxième étape de cet objectif est faite : Le système applique D au cypher-text envoyé par Alice à l'aide de sa clé privée.

3.3.2 Majoration de l'erreur

Avoir un bruit e plus petit que $q/4$ en valeur absolue est ici nécessaire, car :

$$|e| \leq \frac{q}{4} \Leftrightarrow \frac{|e|}{\Delta} \leq \frac{1}{2} \Leftrightarrow \lceil \frac{|e|}{\Delta} \rfloor = 0$$

Montrons que c'est le cas avec une probabilité extrêmement proche de 1, grâce à la façon dont les paramètres ont été générés. On prendra dans la suite $q = 2^8$, qui vérifie bien $q^{1/4} << q$, et avec quoi on a bien $\mathbb{P}[|e| \leq q/4] \approx 1$. Ce sera *a fortiori* vrai pour des valeurs supérieures de q (davantage utilisées dans l'implémentation).

On a $e = Er + e_2 - se_1$, où :
- e_1, e_2, $r \sim \mathcal{N}(0, 4)$
- $E \sim \mathcal{N}(0, \sqrt{q})$
- $s \sim \mathcal{U}(\{-1, 0, 1\})$

Toutes ces variables étant indépendantes, on obtient $Er \sim \mathcal{N}(0, \frac{4\sqrt{q}}{4+\sqrt{q}})$. Majorons $\mathbb{P}[|e| \geq q/4]$.

1. Supposons $s = 0$: Alors $e \sim \mathcal{N}(0, \Sigma_1^2)$ où $\Sigma_1 := 4[1+\frac{4\sqrt{q}}{4+\sqrt{q}}]$. Donc, comme $\mathbb{P}[|e| \geq q/4] = 2\mathbb{P}[e \geq q/4]$ (loi gaussienne centrée en 0),

$$\mathbb{P}[|e| \geq q/4] = 2\int_{q/4}^{+\infty} \frac{1}{\Sigma_1\sqrt{2\pi}} \exp(-\frac{t^2}{2\Sigma_1^2})dt$$

   Après simulation, on obtient une valeur numérique de $erfc(\frac{16\sqrt{10}}{3})$, de l'ordre de 10^{-127} !

2. Supposons $s = 1$: Alors $e \sim \mathcal{N}(0, \Sigma_2^2)$ où $\Sigma_2 := 4[2 + \frac{4\sqrt{q}}{4+\sqrt{q}}]$. Donc

$$\mathbb{P}[|e| \geq q/4] = 2\int_{q/4}^{+\infty} \frac{1}{\Sigma_2\sqrt{2\pi}} \exp(-\frac{t^2}{2\Sigma_2^2})dt$$

   Après simulation, on obtient une valeur numérique de $erfc(16\sqrt{\frac{5}{7}})$, de l'ordre de 10^{-82}.

3. Supposons $s = -1$: De la même manière que dans le point précédent, on obtient $\mathbb{P}[|e| \geq q/4]$ de l'ordre de 10^{-82}.

3.4 Fonction E : chiffrement du résultat précédent

$$E(p_{kB}, résultat) = c' = (c1', c2') = (A'r' + e1', B'r' + e2' + résultat)$$
$$= (A'r' + e1', B'r' + e2' + \Delta m + Er + e2 - se1)$$
$$= (A'r' + e1', B'r' + \Delta m + e')$$

en posant $e' := e'_2 + Er + e_2 - se_1$.

Remarques :

— Ici encore, les bruits e'_1 et e' sont très inférieurs à $q/2$, donc

$$E(p_{kB}, résultat) \approx (A'r', B'r' + \Delta m)$$

— r' est en fait un input de E : les participants l'ont pré-généré entre eux sous forme de secret partagé.

La troisième étape de cet objectif est faite : Le système applique E au résultat précédent à l'aide de la clé publique de Bob.

3.5 Fonction Decryption

Pour cette dernière étape, c'est au tour de Bob de déchiffrer c' à l'aide de sa clef s_B. *Decryption*(c', s_B) donne $D(c', s_B)$ arrondi au multiple de q/2 le plus proche.

Soit d'abord :
D(c', s_B)
= c2'-s_Bc1'
= B'r'+e2'+Δm+(Er+e2-se1)-s_B(A'r'+e1')
= (s_BA'+(E'r'+e'2+Δm+Er+e2-se1)-s_B(A'r'+e1')
= E'r'+e2'+Δm+(Er+e2-se1)-s_Be1'
= Δm + (E'r'+e2'+Er+e2-se1-s_Be1')
= $\Delta m + \tilde{e}$ où $\tilde{e} = E'r' + e2' + Er + e2 - se1 - s_B e1'$

Enfin, l'arrondi au multiple de q/2 le plus proche donne Δm car tout ce qui est entre les dernières parenthèses est strictement inférieur à q/2.
Effectivement, on a

$$m = \underset{k \in \mathbb{Z}}{\operatorname{argmin}} \, |(\Delta m + \tilde{e})\Delta^{-1} - k|$$

Ce que l'on peut réecrire comme $\lceil (\Delta m + \tilde{e} \bmod(\mathrm{q}))\Delta^{-1} \rfloor$, et c'est comme cela que l'on va l'implémenter.
La quatrième et dernière étape de cet objectif est faite : Bob applique Decryption au nouveau cypher-text à l'aide de sa clé secrète.
Ainsi, m est trouvé.

Tout cela est implémenté dans `lwe_functions.rs`, à la suite d'*Encrypt*.

Remarque :

$$m \in \{0, 1\} \Rightarrow \Delta m \in \{0, \frac{q}{2}\} \Rightarrow \Delta m + \tilde{e} \in \{\tilde{e}, \frac{q}{2} + \tilde{e}\}$$

On pose z le choix de Δm selon la valeur de $\tilde{e}$ tel que :

$$z = \begin{cases} 0 \text{ si } \Delta m + \tilde{e} \in]-\infty, \frac{q}{4}] \\ \frac{q}{2} \text{ si } \Delta m + \tilde{e} \in]\frac{q}{4}, +\infty[\end{cases}$$

On a vu qu'il fallait une erreur inférieure en valeur absolue à $q/4$. Montrons en quoi c'est concrètement nécessaire.
Supposons $|\tilde{e}| > q/4$.

- Si $\tilde{e} > q/4$: Si par exemple $m = 0$, alors $\Delta m + \tilde{e} \in]\frac{q}{4}, +\infty[$, donc $z = q/2$ donc on ne récupère pas la bonne valeur de Δm.
- Si $\tilde{e} < -q/4$: Si par exemple $m = 1$, alors $\Delta m + \tilde{e} \in]-\infty, q/4]$, donc $z = 0$ donc on ne récupère pas la bonne valeur de Δm.

De fait, le $\frac{q}{4}$ se réfère à la moitié de l'intervalle $[0, \frac{q}{2}]$, et c'est comme cela que l'on a défini ces régions de décisions. Ainsi, avoir une erreur inférieure en valeur absolue à $q/4$ garantit le bon décodage du message.

Précision : Pour plus de sécurité, nous faisons donc toujours en sorte que $|\tilde{e}| < \frac{q}{4}$ dans l'implémentation puisqu'il y a une probabilité non nulle que ce ne le soit pas. C'est la raison pour laquelle des "min" apparaissent dans le code.

3.6 Bilan

On a finalement calculé $Decryption(EoD(Encryption(m)))$: $Encryption$ chiffrait m sous la clé publique d'Alice, EoD déchiffrait le résultat puis le réenchiffrait sous la clé publique de Bob, et $Decryption$ déchiffrait le message sous la clé privée de Bob. Celui-ci peut alors découvrir ce qu'Alice lui avait envoyé via le système.

3.7 Forme affine EoD

Détaillons l'expression mathématique de EoD, implémentée dans `EoD_sys_initial.rs`. On a :

$$E(\Delta m, e'_1, e'_2, r') = (A'r' + e'_1, B'r' + e'_2 + \Delta m)$$

Or $\Delta m \simeq c_2 - sc_1$, d'où :

$$E(\Delta m, e'_1, e'_2, r') = \begin{pmatrix} 0 & 1 & 0 & A' \\ -c_1^\top & 0 & 1 & B' \end{pmatrix} \begin{pmatrix} s^\top \\ e'_1 \\ e'_2 \\ r' \end{pmatrix} + \begin{pmatrix} 0 \\ c_2 \end{pmatrix}$$

Ainsi, pour $D(s) = c_2 - sc_1 \simeq \Delta m$, on a :

$$E(D(s), e'_1, e'_2, r') = \begin{pmatrix} 0 & 1 & 0 & A' \\ -c_1^\top & 0 & 1 & B' \end{pmatrix} \begin{pmatrix} s^\top \\ e'_1 \\ e'_2 \\ r' \end{pmatrix} + \begin{pmatrix} 0 \\ c_2 \end{pmatrix}$$

On notera dans la suite $EoD(X)$ cette expression, où :

$X = \begin{pmatrix} s^\top \\ e'_1 \\ e'_2 \\ r' \end{pmatrix}$.

En posant $M = \begin{pmatrix} 0 & 1 & 0 & A' \\ -c_1^\top & 0 & 1 & B' \end{pmatrix}$, et $N = \begin{pmatrix} 0 \\ c_2 \end{pmatrix}$, on obtient

$$EoD(X) = MX + N.$$

Remarque : Conscients qu'écrire $EoD(X)$ constitue un abus de notation ici, nous utiliserons quand même cette expression, qui a le mérite de bien décrire ce que font concrètement les n machines.

4 Complexification du problème

Pour garantir de la sécurité de cet envoi, Alice ne va plus envoyer {c,s} à un seul système, mais elle va générer n shares de la clé secrète s puis les envoyer à n machines différentes, ce qui réalisera l'évaluation de EoD à n reprises en parallèle.
On utilise ici de l'algorithme de Secret Sharing de Shamir [2], contenant un algorithme de sharing, et un algorithme de reconstruction (cf. section 2.1).

On appelle le i-ème share de s : s_i avec $i \in [\![1, n]\!]$.
Vu que l'aléa r' est propre à la machine utilisée, on l'appelle r'_i pour la i-ème machine. Cependant, les aléas (e'_1, e'_2) sont les mêmes pour chacune des machines, on les génère avant d'appliquer EoD.
Donc la i-ème machine réalisera l'opération $EoD(s_i, e'_1, e'_2, r'_i)$ [4], ce qu'on appellera l'application L_i. L_i est affine en (s_i, e'_1, e'_2, r'_i), de la même manière que ci-dessus, dans le cas $n = 1$. On nomme son résultat c'_i.

Après que chaque machine i a effectué son opération EoD, les outputs c'_i sont recombinés via le théorème de Lagrange (cf. section 2.1). Ils parviennent à Bob sous la forme d'un ciphertext c' chiffré sous sa clé p_B.

4.1 n=2 machines

Précision : d représente la dimension ici.

4. Dans les algorithmes qui suivent, p_B et c sont pris en entrée.

Algorithm 1 Algorithme de secret sharing pour 2 machines

Require: la clé secrète d'Alice s, le cypher-text c, la clé publique de Bob p_B, et des aléas

Ensure: le message d'Alice m

1. $Coefs$ et $Shares$ sont deux tableaux vides
2. **for** s_i in s **do**
3. On génère un polynôme $P_i(X) = s_i + a_i X$, ce qui retourne $coef_i = [s_i, a_i]$.
4. On génère les shares sur P_i, ce qui retourne $shares_i = [s_{i,1}, s_{i,2}]$ $= [P_i(1), P_i(2)]$
5. On stocke $coef_i$ et $shares_i$ dans les tableaux $Coefs$ et $Shares$
6. **end for**
7. On obtient $Shares = [[s_{1,1}, s_{1,2}], ..., [s_{d,1}, s_{d,2}]]$.
8. $[c_{1,1}, c_{1,2}] \leftarrow EoD_1 :=$ EoD appliqué à $[s_{1,1}, ..., s_{d,1}]$, c, et des aléas
9. $[c_{2,1}, c_{2,2}] \leftarrow EoD_2 :=$ EoD appliqué à $[s_{1,2}, ..., s_{d,2}]$, c, et des aléas
10. **for** j in $[\![1, d]\!]$ **do**
11. On applique l'algorithme de reconstruction (cf. section 2.1) sur les shares $[c_{1,1_j}, c_{2,1_j}]$, ce qui donne $c1_j$, que l'on stocke dans un tableau c_1
12. **end for**
13. On applique l'algorithme de reconstruction sur les shares $[c_{1,2}, c_{2,2}]$, ce qui donne $c2$
14. On applique Decryption sur (c_1, c_2), ce qui nous donne le message initial m d'Alice.

Remarque 1 : $c_{1,2}$ et $c_{2,2}$ sont de dimension 1.

Remarque 2 : Si on fait un rapprochement avec la section 3, nous pouvons nous demander que vaut t+1 ? Effectivement, si t+1 était différent de n, cela impliquerait $0 < t+1 < n$, d'où $t = 0$. Mais cette égalité impliquerait que le share de s est en fait s, ce qui est absurde. Dans le cas $n = 2$, on a alors $t+1 = n$ et c'est le seul cas pour lequel on aura cette égalité.

4.2 n=3 machines

Nous souhaitons implémenter le cas $n = 3$, avec $q = 2^e$ (avec e suffisament grand) et $t = 1$. Pour ce faire [5], nous allons plonger nos vecteurs depuis $(\mathbb{Z}/2^e\mathbb{Z})^d$ dans $GR(2, e)$, anneau de Galois de degré 2 sur $q = 2^e$. Cet anneau vérifie :

$$GR(2, e) = \mathbb{Z}/2^e\mathbb{Z}[Y]/(Y^2 + Y + 1).$$

Comme tout élément de $GR(2, e)$ s'écrit sous la forme $b_0 + b_1 Y$, cet anneau est en bijection avec $(\mathbb{Z}/2^e\mathbb{Z})^2$. Pour y plonger un élément $s \in (\mathbb{Z}/2^e\mathbb{Z})^d$, il suffit donc d'intercaler d zéros entre les coordonnées de s (on obtient donc $s^{GR} = (s_1, 0, s_2, 0, ..., s_d, 0)$). Puis on génère aléatoirement $a_1 \in GR(2, e)$ pour obtenir le polynôme $P = s^{GR} + a_1 Y$. $GR(2, e)$ contient notamment les éléments $0, 1, Y, Y + 1$ dont les différences sont inversibles ; ceci va nous permettre d'utiliser le théorème de Lagrange avec des coefficients dans $GR(2, e)$.
On partage ainsi les shares $s_1 = P(1), s_2 = P(Y), s_3 = P(Y + 1)$ de la clé secrète aux n machines. A la fin, comme en section 4.5, le but pour Bob est de reconstituer le polynôme de départ à partir de ses shares pour obtenir $P(0)$, soit le secret (les n machines ont en effet appliqué EoD entre temps, donc la donnée s^{GR} a été modifiée en $(E(\Delta m + e))^{GR}$). Le théorème de Lagrange qu'il utilise a cette fois ses coefficients dans $GR(2, e)$ et une règle de multiplication modulo $Y^2 + Y + 1$.

Règle d'addition dans $GR(2, e)$:

Soient $b_0, b_1, b'_0, b'_1 \in \mathbb{Z}/2^e\mathbb{Z}$.
On a $b_0 + Yb_1$, $b'_0 + Yb'_1 \in GR(2, e)$ tel que :

$$b_0 + Yb_1 + b'_0 + Yb'_1 = (b_0 + b'_0) + Y(b_1 + b'_1).$$

Ainsi, on utilisera la règle d'addition classique dans $(\mathbb{Z}/2^e\mathbb{Z})^2$:

$$\begin{pmatrix} b_0 \\ b_1 \end{pmatrix} + \begin{pmatrix} b'_0 \\ b'_1 \end{pmatrix} = \begin{pmatrix} b_0 + b'_0 \\ b_1 + b'_1 \end{pmatrix}$$

5. Nous nous inspirons ici de la section C.4 de l'article de Matthieu Rambaud [3].

Règle de réécriture pour l'implémentation :

$\begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$ peut se réécrire sous la forme $b_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + b_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.
Ainsi, seule la règle de multiplication suivante nous est nécessaire pour l'implémentation.

Règle de multiplication dans $GR(2, e)$:

On note $\begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$ pour un élément $b_0 + b_1 Y$.

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Valeur de t :

Comme pour le cas n=2, nous pouvons nous demander à quoi est égal t dans ce contexte. Avec un raisonnement similaire à celui du cas précédent, t ne peut pas être nul mais peut valoir 1 ou 2. Or t=2 impliquerait que l'on ait une 3-reconstruction, ce qui n'est pas pertinent vu que l'on a trois machines à disposition, mais ce cas marche. Ainsi, t=1 et on peut donc adapter l'algorithme 1 à n'importe quelle paire de shares. On a choisi les shares 1 et 2 dans l'algorithme suivant par souci de simplicité, mais on aurait pu prendre n'importe quelle autre paire.

Algorithm 2 Algorithme de secret sharing pour 3 machines

Require: $s = [s_1, ..., s_d]$, $c = [(c1_1, ..., c1_d), c2]$, p_B, et des aléas
Ensure: m

1. $Coefs$ et $Shares$ sont deux tableaux vides
2. $s^* \leftarrow [s_1, 0, s_2, 0, ..., s_d, 0]$
3. $c_1^* \leftarrow [c1_1, 0, c1_2, 0, ..., c1_d, 0]$
4. $c_2^* \leftarrow [c2, 0]$
5. $c^* \leftarrow [c_1^*, c_2^*]$
6. On plonge aussi les aléas dans $(\mathbb{Z}/2^e\mathbb{Z})^{2d}$
7. **for** $[s_i, 0]$ in s^* **do**
8. On génère un polynôme $P_i(X) = \begin{pmatrix} s_i \\ 0 \end{pmatrix} + \begin{pmatrix} a_{1_i} \\ a_{2_i} \end{pmatrix} X$, ce qui retourne $coef_i = [s_i, 0, a_{1_i}, a_{2_i}]$
9. $shares_i \leftarrow [s_{i,1}, s_{i,2}, s_{i,3},] := = [P_i(\begin{pmatrix} 1 \\ 0 \end{pmatrix}), P_i(\begin{pmatrix} 0 \\ 1 \end{pmatrix}), P_i(\begin{pmatrix} 1 \\ 1 \end{pmatrix})]$
10. On stocke $coef_i$ et $shares_i$ dans les tableaux $Coefs$ et $Shares$
11. **end for**
12. On obtient $Shares = [shares_1, ..., shares_d]$
13. $[c_{1,1}, c_{1,2}] \leftarrow EoD_1 :=$ EoD appliqué à $[s_{1,1}, ..., s_{d,1}]$, c^*, et des aléas
14. $[c_{2,1}, c_{2,2}] \leftarrow EoD_2 :=$ EoD appliqué à $[s_{1,2}, ..., s_{d,2}]$, c^*, et des aléas
15. **for** j in $[\![1, d]\!]$ **do**
16. On applique l'algorithme de reconstruction sur les shares $[c_{1,1_j}, c_{2,1_j}]$, ce qui donne $c1_j$, que l'on stocke dans un tableau c_1
17. **end for**
18. On applique l'algorithme de reconstruction sur les shares $[c_{1,2}, c_{2,2}]$, ce qui donne $c2$
19. On applique Decryption sur (c_1, c_2), ce qui nous donne le message initial m d'Alice.

Remarque : $s_{i,1}$, $s_{i,2}$, et $s_{i,3}$ sont de dimension 2, avec la deuxième coordonnée non connue à l'avance car l'évalutaion du polynome P_i implique d'utiliser la règle de multiplication énoncée plus haut.

De la même manière, $c_{1,2_j}$, $c_{1,2_j}$, $c_{1,2}$ et $c_{2,2}$ sont de dimension 2, et leur deuxième coordonnée n'est pas connue d'avance car la fonction EoD utilise la règle de multiplication.

Conclusion

Le secret-sharing a ainsi permis de déléguer les calculs au système des n machines, en lesquelles on ne fait pas totalement confiance. La confidentialité était assurée par la t-privacy : nous avons en effet montré dans l'objectif 1 que, le polynôme étant choisi par loi uniforme, t shares variaient aussi uniformément. A la sortie des n machines, comme nous l'avons prouvé en objectif 2, Bob peut bien reconstituer le message chiffré en utilisant le théorème de Lagrange. On n'apprend toutefois rien de plus avec t machines corrompues qu'on ne pourrait déduire de l'état initial des $n - t$ machines honnêtes restantes.

Ces calculs réalisés par le système ont permis un ré-enchiffrement du message, nécessaire à sa compréhension par Bob. Ce ré-enchiffrement a pris la forme d'une fonction EoD dont l'affinité facilitait l'implémentation. Il n'était possible que si l'erreur ne dépassait pas le seuil $q/4$; mais cette condition était *de facto* vérifiée avec une probabilité très proche de 1. Les tests effectués suite à notre implémentation ont ainsi toujours fonctionné quand on avait $|e| \leq q/4$.

Lorsque le nombre de machines augmente, par exemple pour 3 machines, un plongement des variables dans un anneau de Galois est utile pour garantir la confidentialité. L'étude théorique achevée, il reste à implémenter ce cas $n = 3$, et à se pencher sur les cas $n \geq 4$.

Références

[1] Rikke Bendlin et al. *Semi-Homomorphic Encryption and Multiparty Computation*. Cryptology ePrint Archive, Paper 2010/514. 2010. URL : https://eprint.iacr.org/2010/514.

[2] Adi Shamir. « How to share a secret ». In : *Communications of the ACM* 22.11 (1979), p. 612-613.

[3] Antoine Urban et Matthieu Rambaud. « Share\& Shrink:(In-) Feasibility of MPC from one Broadcast-then-Asynchrony, and Improved Complexity ». In : *Cryptology ePrint Archive* (2022).