Migwel

Zeste de savoir

Partager une clé secrète entre plusieurs personnes : la méthode de Shamir

23 février 2022

Table des matières

Introduction

Dans ce tutoriel, nous allons nous intéresser au partage de clé secrète et plus particulièrement, à l'algorithme de Shamir [1]footnote:1. Ce que l'on veut dire par partage de clé secrète est que nous voulons chiffrer un message m et distribuer des parties à n personnes. Nous voulons ensuite que si k de ces n personnes (avec k < n) combinent leur partie, elles puissent récupérer le message m.

Afin d'aider à la compréhension, nous pouvons imaginer que nous sommes le patron d'une banque et nous voulons protéger le code (utilisons 5887 pour l'exemple) de notre coffre-fort. Nous avons 5 personnes de confiance à qui nous voulons distribuer une partie et si 3 d'entre d'elles combinent leur partie, elles devraient être en mesure de récupérer le code du coffre. Dans cet exemple que nous allons utiliser tout au long de ce tutoriel, m est le code du coffre (5887), n est le nombre de personnes de confiance (5) et k est le nombre de personnes suffisantes pour récupérer le code (3).

Dans le cadre de ce partage de clé, il est important que tout groupe de personnes composé de moins de k personnes ne puisse gagner aucune information sur notre code. Soit nous avons un nombre de clés suffisant et nous pouvons récupérer le code, soit ce n'est pas le cas et nous n'avons aucune information sur le code.

1. [2]footnote:1 https://dl.acm.org/doi/10.1145/359168.359176

1. L'approche naïve et ses défaults

Avant même de commencer, regardons d'abord à quoi ressemblerait une approche simple et naïve à ce problème et pourquoi nous avons besoin de quelque chose de plus puissant. Après tout, si une approche simple est suffisante, à quoi bon se casser la tête à élaborer un algorithme complexe pour arriver au même résultat?

L'approche la plus simple est de diviser notre clé en autant de personnes que nécessaire. Par exemple, si nous utilisons notre code 5887, on pourrait le diviser en quatre (5, 8, 8 et 7) et donner chaque chiffre à une personne différente. Ensuite, il suffira à ces personnes à se réunir et elle pourront récupérer le code initial. Cette approche a cependant plusieurs défauts :

— Si l'une des quatre personnes perd son fragment de code ou si elle décède, elle est perdue à jamais et nous ne pourrons jamais reconstruire le code initial.
— Recevoir un fragment nous donne une information importante sur le code initial. Aussi, combiner deux ou trois de ces fragments nous donne de plus en plus de connaissances quant à ce code. Avec trois fragments, nous pouvons même deviner le code initial en ayant une chance sur 10 d'avoir raison. Or, nous voulons vraiment éviter ce scénario! Nous ne voulons donner des informations sur le code initial que si nous avons tous les fragments nécessaires (ici, quatre fragments).

Bien sûr, le premier défaut peut être plus ou moins amélioré en améliorant un peu notre stratégie mais le deuxième est beaucoup plus difficile à régler. C'est principalement pour cette raison que nous allons devoir faire appel à des algorithmes plus avancés.

2. Les fonctions et les notations

2.1. Fonctions

Avant de se lancer dans le vif du sujet, il nous faut faire un rapide rappel de mathématiques, en commençant par les fonctions.

Wikipedia propose la définition suivante :

> En mathématiques, une fonction permet de définir un résultat (le plus souvent numérique) pour chaque valeur d'un ensemble appelé domaine.

Ici, nous allons nous intéresser au domaine des nombres réels $\mathbb{R}$. Ainsi, par exemple, ceci est une fonction :

$$f(x) = x^2$$

Pour chaque valeur x de notre domaine $\mathbb{R}$, cette fonction définit un résultat $f(x) = x^2$. Par exemple, pour la valeur $x = 2$, le résultat de notre fonction est 4. Et ceci peut être calculé pour tout $x \in \mathbb{R}$.

x	f(x)

0	0
1	1
2	4
3	9
3,5	12.15
4	16
10	100
...	...

Nous pouvons aussi représenter cette fonction sur un graphe à deux dimensions où l'axe horizontal représentera les valeurs que prend x tandis que l'axe vertical représentera les valeurs que prend $f(x)$. Pour noter les points sur un tel graphe, nous utilisons la notation (x, y) où x est la valeur de l'abscisse (la distance horizontale du point à l'origine) et y est la valeur de l'ordonnée (la distance verticale du point à l'origine).

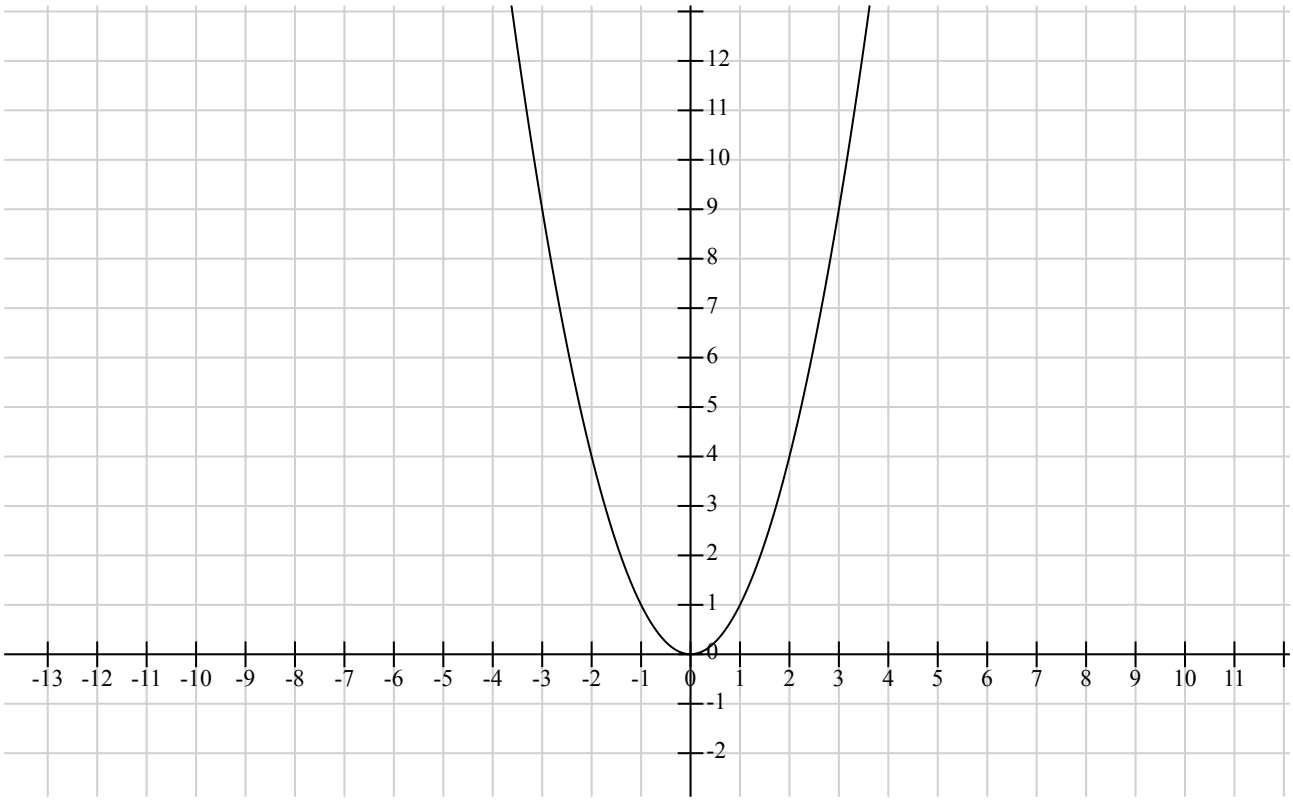

FIGURE 2.1. – Graphe x^2

Et nous pouvons bien voir que la courbe tracée passe par les points calculés dans le tableau défini plus haut.

2.2. Notations

Avant de continuer, il nous faut aussi rapidement introduire deux notations qui seront utilisées au long de ce tutoriel: $\sum$ et $\prod$. Il s'agit respectivement des signes de somme et de produit. Par exemple, plutôt que d'écrire $f(X) = X + X^2 + X^3 ... + X^{100}$ nous allons plutôt écrire

$$f(X) = \sum_{i=1}^{100} X^i$$

Et de même pour le produit où nous allons préférer

$$\prod_{i=1}^{100} X^i$$

à $X \times X^2 \times X^3 \times ... \times X^{100}$.

Pour les programmeurs d'entre vous, vous pouvez voir ces opérations comme des boucles effectuant des additions ou des multiplications.

Un autre symbole que nous utiliserons dans ce tutoriel est $\forall$ qui signifie "pour tout". Ainsi, si nous écrivons $f(x) = 0, \forall x \in \mathbb{R}$, il faut le lire comme "la fonction $f(x)$ vaut 0 pour tout x dans $\mathbb{R}$".

3. Fonction polynomiale

Une famille particulière de fonctions que nous allons utiliser dans ce tutoriel est celle des fonctions polynomiales. Voyons ce que Wikipedia peut nous dire à leur propos[3] footnote:1 :

> En mathématiques, une fonction polynomiale (parfois appelée fonction polynôme) est une fonction obtenue en évaluant un polynôme.

Je vous l'accorde, ça ne nous avance pas beaucoup… Pour être plus concret, une fonction polynomiale est une fonction qui peut s'écrire sous la forme

$$f(X) = a_n X^n + a_{n-1} X^{n-1} + ... + a_1 X + a_0 = \sum_{i=0}^{n} a_i X^i$$

où les valeurs $a_0, a_1, ..., a_{n-1}, a_n$ sont appeléss *coefficients* et sont des valeurs réelles ou complexes.

Pour vous aider à comprendre ce qu'est une fonction polynomiale, voici quelques exemples :

$$f(X) = X^2$$

Et nous pouvons vérifier qu'il s'agit bien d'une fonction polynomiale car $X^2 = a_2 X^2 + a_1 X + a_0$ où $a_2 = 1, a_1 = 0, a_0 = 0$.

$$f(X) = X^5 + 10X^4 - 2X^3 + 58$$

Et nous pouvons vérifier qu'il s'agit bien d'une fonction polynomiale car $X^5+10X^4-2X^3+58 = a_5X^5 + a_4X^4 + ... + a_1X + a_0$ où $a_5 = 1, a_4 = 10, a_3 = -2, a_2 = 0, a_1 = 0, a_0 = 58$.

Par contre, le fonction suivante **n'est pas polynomiale**.

$$f(X) = e^X + X^2$$

En effet, e^X ne peut pas s'écrire sous la forme a_nX^n.

4. Interpolation de Lagrange

Maintenant que nous savons ce qu'est une fonction et que nous savons trouver tous les points par lesquels elle passe, nous allons nous intéresser à l'opération inverse: comment trouver une fonction polynomiale passant par certains nombres distincts donnés?

Pour trouver ce polynôme, nous allons appliquer une méthode appelée **l'interpolation de Lagrange**. Celle-ci se base sur le fait que si nous partons de $n+1$ points $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, il est toujours possible de trouver une et une seule fonction polynomiale de degré maximal n qui passe par tous ces points.

Par exemple, si nous partons des points $(0, 5), (-1, 3), (-2, 3)$ et appliquons l'interpolation de Lagrange, nous serons en mesure de retrouver le polynôme de degré 2 passant par tous ces points (en l'occurrence, $f(X) = X^2 + 3X + 5$).

Nous allons d'abord donner l'approche générique en termes mathématiques puis l'illustrer avec un exemple. Si vous n'arrivez pas à suivre la partie stricte, jetez un œil à l'exemple, il devrait vous aider à comprendre comment cette méthode fonctionne.

Pour trouver ce polynôme, nous allons d'abord calculer les polynômes de Lagrange qui s'écrivent de la façon suivante :

$$l_i(X) = \frac{X - x_0}{x_i - x_0} \times ... \times \frac{X - x_{i-1}}{x_i - x_{i+1}} \times \frac{X - x_{i+1}}{x_i - x_{i+1}} \times ... \times \frac{X - x_n}{x_i - x_n} = \prod_{j=0, j\neq i}^{n} \frac{X - x_j}{x_i - x_j}$$

Pour chacun de ces polynômes, il faut noter les choses suivantes :

1. Pour chaque x_j où $j \neq i$, $l_i(x_j) = 0$. En effet, pour ces valeurs, le numérateur s'annulera pour l'une des fractions présentes dans le produit.
2. Pour x_i, $l_i(x_i) = 1$. Ceci vient du fait que dans ce cas, le numérateur et le dénominateur sont égaux.

1. [4]footnote:1 https://fr.wikipedia.org/wiki/Fonction_polynomiale

Maintenant, nous pouvons utiliser ces polynômes de Lagrange et les combiner afin de trouver le polynôme de degré maximal n passant par tous les *n+1* points :

$$L(X) = \sum_{i=0}^{n} y_i l_i(X)$$

En utilisant les propriétés des polynômes de Lagrange présentées ci-dessus, nous pouvons aisément conclure que $L(x_i) = y_i, \forall i \in 0, ..., n$.

Prenons un exemple pour illustrer cette méthode. Nous allons utiliser les 3 points suivants : $x_0 = (1, 2)$, $x_1 = (5, 10)$ et $x_2 = (4, 6)$. En utilisant ces points, nous pouvons calculer les polynômes de Lagrange suivants :

$$\begin{cases} l_0(X) = \dfrac{X-5}{1-5} \times \dfrac{X-4}{1-4} = \dfrac{1}{12}(X^2 - 9X + 20) \\ l_1(X) = \dfrac{X-1}{5-1} \times \dfrac{X-4}{5-4} = \dfrac{1}{4}(X^2 - 5X + 4) \\ l_2(X) = \dfrac{X-1}{4-1} \times \dfrac{X-5}{4-5} = \dfrac{-1}{3}(X^2 - 6X + 5) \end{cases}$$

Et nous pouvons donc trouver le polynôme passant par les points donnés :

$$\begin{aligned} L(X) &= y_0 l_0(X) + y_1 l_1(X) + y_2 l_2(X) \\ &= \frac{2}{12}(X^2 - 9X + 20) + \frac{10}{4}(X^2 - 5X + 4) - \frac{6}{3}(X^2 - 6X + 5) \\ &= (\frac{2}{12} + \frac{10}{4} - \frac{6}{3})X^2 - (\frac{18}{12} + \frac{50}{4} - \frac{36}{3})X + (\frac{40}{12} + \frac{40}{4} - \frac{30}{3}) \\ &= \frac{2}{3}X^2 - 2X + \frac{10}{3} \end{aligned}$$

Nous pouvons nous convaincre que ce résultat est correct en vérifiant que $L_i(x_i) = y_i$:

$$\begin{cases} L(x_0) = L(1) = \dfrac{2}{3} - 2 + \dfrac{10}{3} = 2 \\ L(x_1) = L(5) = \dfrac{50}{3} - 10 + \dfrac{10}{3} = 10 \\ L(x_2) = L(4) = \dfrac{32}{3} - 8 + \dfrac{10}{3} = 6 \end{cases}$$

La dernière chose qu'il nous reste à prouver est que le polynôme trouvé par cette méthode est bien unique.

Pour ce faire, nous allons imaginer l'existence d'un autre polynôme $G(X)$ passant également par les points $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$. Pour chacun de ces points, nous avons donc $L(x_i) = y_i$ et $G(x_1) = y_i$ (où $i \in 0, 1, ..., n-1, n$). Nous pouvons aussi définir une fonction $D(X) = L(X) - G(X)$ et réaliser que

$$\begin{gathered} D(x_i) = L(x_i) - G(x_i) = y_i - y_i = 0 \\ \forall i \in \{0, 1, ..., n-1, n\} \end{gathered}$$

Cela signifie que $D(X)$ est un polynôme de degré n valant 0 pour $n+1$ valeurs de X[5]. Or il n'existe qu'un seul polynôme pour lequel ceci peut être vrai : le polynôme $D(X) = 0$. Or, si $D(X) = 0$, cela signifie que $L(X) = G(X)$. Ce qui prouve que le polynôme trouvé via la méthode d'interpolation lagrangienne est le seul polynôme de degré maximal n passant par les $n+1$ points donnés.

5. Génération des fragments

Tout d'abord, nous allons nous attarder à la génération des fragments. Pour rappel, les fragments sont les informations qui seront données aux personnes de confiance et qui pourront être combinées pour récupérer le code initial.

Pour ce faire, nous allons construire une fonction polynomiale de degré $k-1$. Pour rappel, k est le nombre de clés suffisant à fournir lors de l'étape de reconstruction du code.

Cette étape est relativement simple, notre fonction étant polynomiale, elle s'écrit sous la forme

$$P(X) = \sum_{i=0}^{k-1} a_i X^i$$

Nous pouvons choisir les termes $a_{k-1}, a_{k-2}, ..., a_1$ de façon aléatoire dans $\mathbb{R}$[7] et nous utilisons le code m pour a_0. Ainsi, cela nous donne $P(X) = m + \sum_{i=1}^{k-1} a_i X^i$. Avec cette fonction, nous pouvons générer un nombre n de fragments au format $(x_i, y_j) = (x_j, P(x_j))$ où l'on choisira x_j incrémentalement en partant de 1.

Tout ceci peut paraitre très abstrait, utilisons donc notre exemple de base où m = 5887, k = 3 et n = 5. Nous devons donc générer un polynôme de degré 2 sous la forme $P(X) = a_2X^2 + a_1X + 5887$. Nous pouvons choisir a_1 et a_2 aléatoirement. Utilisons $a_1 = 1689$ et $a_2 = 250$. Cela nous donne

$$P(X) = 250X^2 + 1689X + 5887$$

Maintenant que nous avons notre fonction polynomiale, il ne nous reste plus qu'à générer nos 5 fragments :

$$\begin{aligned}
P(1) = 7826 &\Rightarrow key_1 = (1, 7826) \\
P(2) = 10265 &\Rightarrow key_2 = (2, 10265) \\
P(3) = 13204 &\Rightarrow key_3 = (3, 13204) \\
P(4) = 16643 &\Rightarrow key_4 = (4, 16643) \\
P(5) = 20582 &\Rightarrow key_5 = (5, 20582)
\end{aligned}$$

Ces fragments peuvent maintenant être distribués aux 5 personnes de confiance sans qu'aucun de ces fragments individuellement ne dévoile quoi que ce soit quant à la valeur du code secret.

1. [6]footnote:1 On dit que $D(X)$ a $n+1$ racines
1. [8]footnote:1 Si vous vous demandez comment choisir un nombre aléatoirement dans $\mathbb{R}$ et si c'est même possible, bravo à vous, vous avez détecté un souci dans cette méthode. Nous y reviendrons plus loin dans ce tutoriel.

6. Récupération du code à partir des fragments

Très bien, nous avons 5 fragments, comment pouvons-nous maintenant récupérer le code secret en utilisant 3 de ces fragments? C'est ici que l'interpolation lagrangienne va rentrer en jeu.

Notre code de déchiffrement va recevoir un nombre k de fragments $(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$. À partir de ces fragments, il est possible de retrouver l'unique fonction polynomiale de degré maximal $k-1$ passant par ces k points comme présenté plus tôt dans ce tutoriel. Nous pouvons calculer les polynômes de Lagrange $l_i(X)$ puis les utiliser pour trouver la fonction P(X) passant par ces points. Cette fonction sera de la forme

$$P(X) = a_{k-1}X^{k-1} + a_{k-2}X^{k-2} + ... + a_1X + a_0$$

Or, lors de l'étape de génération des fragments, on a décidé que notre code secret m était utilisé pour la valeur a_0. TADA! Notre code est récupéré et on peut le retourner à nos 3 personnes de confiance.

Une fois encore, réutilisons notre exemple pour mieux visualiser ceci. Nous allons utiliser les fragments $(1, 7826), (2, 10265), (4, 16643)$ (n'importe quelle autre combinaison de 3 fragments fonctionnerait aussi). Calculons maintenant nos polynômes de Lagrange :

$$\begin{cases} l_0(X) = \dfrac{X-2}{1-2}\dfrac{X-4}{1-4} = \dfrac{1}{3}(X^2 - 6X + 8) \\ l_1(X) = \dfrac{X-1}{2-1}\dfrac{X-4}{2-4} = \dfrac{-1}{2}(X^2 - 5X + 4) \\ l_2(X) = \dfrac{X-1}{4-1}\dfrac{X-2}{4-2} = \dfrac{1}{6}(X^2 - 3X + 2) \end{cases}$$

Et avec ces polynômes, on peut calculer notre fonction P :

$$\begin{aligned} P(X) &= y_0l_0(X) + y_1l_1(X) + y_2l_2(X) \\ &= \frac{7826}{3}(X^2 - 6X + 8) - \frac{10265}{2}(X^2 - 5X + 4) + \frac{16643}{6}(X^2 - 3X + 2) \\ &= (\frac{7826}{3} - \frac{10265}{2} + \frac{16643}{6})X^2 - (15652 - \frac{51325}{2} + \frac{16643}{2})X + (\frac{62608}{3} - 20530 + \frac{16643}{3}) \\ &= 250X^2 + 1689X + 5887 \end{aligned}$$

Et notre code secret est bien **5887**!

7. Place à l'arithmétique modulaire

Cette partie n'est pas nécessaire pour comprendre le cœur de la méthode de Shamir mais elle est indispensable pour que cette méthode soit strictement correcte et que sa sécurité soit garantie. Elle implique cela dit des calculs un peu plus compliqués que ce que nous avons vu jusqu'ici. Si vous n'arrivez pas à suivre cette section, ne vous en faites pas, elle ne bouleverse pas l'idée de base que nous avons utilisée plus haut.

Il est maintenant temps pour nous de revenir sur une note mentionnée plus tôt dans ce tutoriel. Nous avons énoncé que les coefficients $a_1, a_2, ..., a_{n-1}, a_n$ étaient choisis aléatoirement dans $\mathbb{R}$. Comment faire cela en pratique? La réponse est très simple: c'est impossible. $\mathbb{R}$ est un ensemble infini, il nous est donc impossible de choisir des élements de façon uniformément aléatoire. Pour y remédier, nous allons passer à une autre forme d'arithmétique: l'arithmétique modulaire.

Nous allons donc plus choisir nos coefficients dans $\mathbb{R}$ mais plutôt dans $\mathbb{Z}_p$, c'est à dire l'ensemble des nombres entiers **modulo p**.

C'est quoi un modulo?

Modulo, c'est l'opération qui nous donne le reste d'une division. Par exemple, $17 \mod 5 = 2$ car $17 = 3 * 5 + 2$ (2 est le reste de la division de 17 par 5). Faire de l'arithmétique dans $\mathbb{Z}_p$ plutôt que dans $\mathbb{Z}$ ou $\mathbb{R}$ signifie qu'on se limite aux nombres entre 0 et p-1.

Pour continuer à utiliser notre exemple, regardons quelques opérations dans $\mathbb{Z}_5$:

$$(17 + 11) \mod 5 = 28 \mod 5 = 3$$

$$(3 \times 7) \mod 5 = 21 \mod 5 = 1$$

Notre ensemble est donc maintenant fini et il est possible de choisir des nombres aléatoirement et de façon uniforme dans cette ensemble.

Il ne nous reste plus qu'à préciser un détail: pour que notre interpolation lagrangienne fonctionne correctement en arithmétique modulaire, il nous faudra choisir un nombre p **premier**. Nous avons déjà abordé pas mal de concepts mathématiques et prouver cette propriété serait assez compliqué. Donc pour ne pas trop dévier du sujet initial, cette affirmation ne sera pas démontrée ici mais vous pouvez essayer le démontrer vous-mêmes.

Qu'est-ce que ça change pour notre algorithme?

Pour voir comment ceci affecte notre algorithme, prenons un nombre premier $p > m$, par exemple 6301. Nous pouvons donc recalculer nos clés en utilisant la formule suivante :

$$P(X) = 250X^2 + 1689X + 5887 \mod 6301$$

Maintenant que nous avons notre fonction polynomiale, il ne nous reste plus qu'à générer nos 5 clés :

$$\begin{aligned}
P(1) = 7826 \mod 6301 = 1525 &\Rightarrow key_1 = (1, 1525) \\
P(2) = 10265 \mod 6301 = 3964 &\Rightarrow key_2 = (2, 3964) \\
P(3) = 13204 \mod 6301 = 602 &\Rightarrow key_3 = (3, 602) \\
P(4) = 16643 \mod 6301 = 4041 &\Rightarrow key_4 = (4, 4041) \\
P(5) = 20582 \mod 6301 = 1679 &\Rightarrow key_5 = (5, 1679)
\end{aligned}$$

Lorsque nous fournirons les clés key_1, key_2, key_4, l'algorithme de récupération générera les mêmes polynômes de Lagrange mais le calcul du polynôme P changera comme suit :

$$\begin{aligned}
P(X) &= y_0 l_0(X) + y_1 l_1(X) + y_2 l_2(X) \mod 6301 \\
&= 1525 \cdot 3^{-1}(X^2 - 6X + 8) - 3964 \cdot 2^{-1}(X^2 - 5X + 4) + 4041 \cdot 6^{-1}(X^2 - 3X + 2) \mod 6301 \\
&= (1525 \cdot 3^{-1} - 3964 \cdot 2^{-1} + 4041 \cdot 6^{-1} \mod 6301)X^2 + (-3050 + 19820 \cdot 2^{-1} - 4041 \cdot 2^{-1} \mod 63 \\
&= 250X^2 + 1689X + 5887
\end{aligned}$$

Et nous voyons que nous sommes bien parvenus à retrouver le bon polynôme en utilisant l'arithmétique modulaire.

8. Comparaison avec l'approche naïve

Revenons à ce que nous avons mentionné au début de ce tutoriel quant aux défauts d'une approche naïve. Avec l'algorithme de Shamir, nous pouvons générer autant de fragments que l'on souhaite et nous pouvons définir un seuil de fragments nécessaires à la récupération du code. Donc nous ne craignons plus qu'un fragments soit perdu par mégarde: n'importe quelle combinaison de fragments, tant qu'il y en assez, nous permettra de récupérer le code initial.

Regardons maintenant le second et principal défaut de l'approche naïve: cette approche nous donnait de plus en plus d'information au fur et à mesure que l'on combinait les fragments ensemble. Or ce que nous cherchons est une approche de tout ou rien: **soit nous n'avons pas assez de fragments et je ne peux rien dire quant au code initial, soit j'en ai assez et je peux récupérer ce code**.

Ainsi, pour pouvoir affirmer que notre algorithme est sûr, nous devons montrer que si quelqu'un possède au plus $k-1$ clés, il n'est pas plus proche de récupérer le message secret qu'une personne n'en ayant aucune[9]footnote:1.

Heureusement, pour notre algorithme, c'est bel et bien le cas! En effet, avec $k-1$ clés, le hacker connaitrait $k-1$ points par lesquels notre polynôme passe mais nous savons que notre polynôme est de degré $k-1$. Or, dans $\mathbb{Z}_p$, il existe p polynômes passant par ces $k-1$ points, correspondant à autant de a_0 distincts (c'est-à-dire à des codes secrets distincts). Ainsi, même avec $k-1$ points,

on en est au même point qu'un hacker sans le moindre fragment : chacun des codes dans $\mathbb{Z}_p$ est équiprobable. Le hacker n'a ainsi gagné aucune information sur le polynôme utilisé pour créer les clés et notre algorithme est bel et bien correct et sûr.

La sécurité de l'algorithme repose également sur le fait que lorsque nous choisissons les paramètres du polynôme lors de la génération des clés, nous choisissons ceux-ci de façon aléatoires dans $\mathbb{Z}_p$. En pratique, cette contrainte n'est pas aussi simple à réaliser qu'il n'y parait mais l'étude de la génération de nombres aléatoires est trop vaste et complexe pour être abordée dans le cadre de ce tutoriel.

Conclusion

Voilà, grâce à la méthode de Shamir, nous avons maintenant une méthode nous permettant de partager une clé secrète entre plusieurs personnes, sans compromettre la sécurité de notre système. Donc si vous avez un coffre-fort ou une bombe nucléaire en votre possession, vous êtes maintenant en mesure de partager leur code à plusieurs proches de confiance. En espérant qu'ils en fassent bonne utilisation.

Si vous voulez approfondir le sujet, je vous invite à visiter la page Wikipedia . Malheureusement, la page francophone est assez difficile à suivre donc je vous invite plutôt à vous diriger vers la page anglophone qui est plutôt bien expliquée et qui contient même un exemple d'implémentation en Python.

Pour finir, un grand merci à @**Lucas-84**, @Gabbro et @melepe pour leurs retours durant la beta et la validation de ce tutoriel.

1. [10]footnote:1 Si vous voulez une explication plus en profondeur sur cette définition de la sécurité d'un algorithme, vous pouvez lire cet article de Bermudes.