Devoir 2 : Code de Reed-Solomon (*structures algébriques*)

Informations pratiques

Ce devoir est individuel. Il vous est demandé de résoudre ce problème individuellement. Vous êtes autorisé · e, et même encouragé · e, à échanger des idées avec d'autres étudiant · e · s sur la façon d'aborder ce devoir. En revanche, vous devez rédiger votre solution individuellement : ne partagez pas votre production. Nous serons intransigeant · e · s si nous observons des similitudes dans votre code, lequel passera dans les logiciels anti-plagiat de Gradescope. Enfin, utiliser un assistant virtuel à la rédaction comme ChatGPT est autorisé, mais il est impératif de centionner très clairement en commentaires dans votre code à quels endroits cela a été le cas, en précisant les requêtes utilisées.

Echéance : mercredi 8 novembre, 18h00.

Soumission. Vous remettrez votre production via les étapes suivantes :

1. Sur `https://www.gradescope.com/`, se connecter via le bouton "Log In" en haut à droite.
2. Cliquer sur "School Credentials" en bas à gauche.
3. Cliquer sur "UCLouvain Username" dans le liste alphabétique des universités.
4. Vous serez redirigé · e vers la page d'identification UCLouvain, sur laquelle vous vous connectez comme d'habitude.
5. Si vous êtes inscrit · e sur la page Moodle du cours, vous devriez voir apparaître LEPL1108 dans la liste de vos cours. Si ce n'est pas le cas, vous devriez pouvoir rejoindre le cours en vous servant du code `6GGV63`.
6. Complétez le fichier `reed_solomon.py` et soumettez-le rempli sur Gradescope.

Consignes supplémentaires.

- Un fichier "squelette" `reed_solomon.py` à compléter est fourni. Vous devez uniquement soumettre le fichier `reed_solomon.py`. Respectez le modèle fourni, vous risquez sinon d'avoir des problèmes de codage ou des problèmes de lecture du code par Gradescope.
- Le seul package Python autorisé est : `numpy`. Pour information, Gradescope tourne sur la version 3.10.6 de Python.
- La note finale sera évaluée sur base de tests secrets. Gradescope vous indiquera uniquement si votre code réussit les tests publics ; le reste étant caché. Les tests publics sont représentatifs des tests privés.

- Un fichier `run_pub_tests.py` vous est également fourni. Celui-ci est à placer dans le même dossier que les fichiers `reed_solomon.py` sur votre ordinateur afin de tester votre code. Exécuter ce fichier vous affichera les notes des tests publics. Pour rappel, il ne faut pas le soumettre sur Gradescope.
- Toutes les questions concernant le devoir doivent être posées sur le Forum dédié au Devoir 2 sur la page Moodle du cours.

1 Objectifs et contexte

Dans ce deuxième devoir, nous vous demandons d'implémenter une partie de la méthode de codage de Reed-Solomon. L'algorithme de Reed-Solomon est un code correcteur d'erreurs basé sur les corps finis. Il peut être appliqué dans de nombreux domaines tels que la lecture de CD-ROMS/DVD, la transmission de données sur des réseaux sans fils ou l'ADSL.

L'objectif du code de Reed-Solomon est de coder l'information à transmettre sur le canal de communication (en transformant et en augmentant légèrement le message original) de manière telle que, même si un certain nombre de bits sont corrompus par le canal de communication, il reste possible de décoder, c'est-à-dire de parfaitement reconstruire le message initial.

La figure 1 montre la structure globale de la transmission d'information que l'on souhaite faire dans ce devoir.

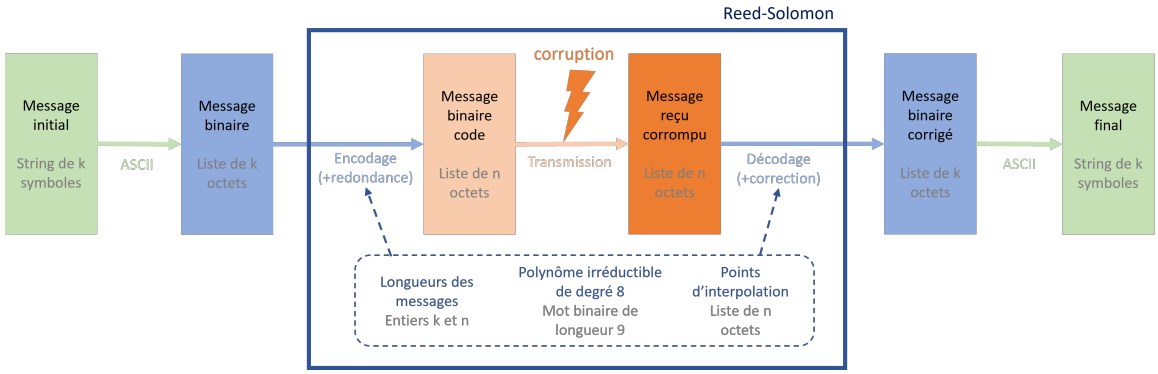

FIGURE 1 – Structure de transmission de message grâce à Reed-Solomon

Dans ce devoir, nous souhaitons envoyer une phrase écrite en français. Cette phrase est décomposée en une liste de symboles (lettres, chiffres, espaces,...), qui sont transformés en bits via le code ASCII. Pour information, le code ASCII (American Standard Code for Information Interchange) est un standard pour associer des symboles avec des octets, voir `https://www.ascii-code.com/`. Chaque symbole est alors représenté par un "mot" de 8 bits (= 1 octet). Chaque bit pouvant prendre deux valeurs, il y a donc $2^8 = 256$ mots possibles. Par la suite, le message est encodé grâce à l'encodage de Reed-Solomon, puis transmis sur un canal soumis à des interférences qui vont modifier le message. Ensuite, le message reçu est décodé et corrigé grâce au décodage de Reed-Solomon. Enfin, la suite d'octets est re-traduite en une phrase en français.

Durant la transmission, des bits peuvent être corrompus. Dans ce devoir, nous nous cantonnerons à des corruption de type "effacement", où un bit dont on connait l'emplacement est illisible et

remplacé par un “x”. On peut imaginer qu'il a été effacé durant la transmission, que la transmission a simplement été coupée à ce moment précis, ou encore qu'il y a une poussière à un endroit précis d'un DVD. Voici quelques exemples où l'on peut voir que certains octets contiennent des bits effacés alors que d'autres sont intacts après transmission :

Octet original	10110001	10111110	11011111	01001011	01011100	01000000
Octet reçu	10110x01	10111110	11011111	010xxx11	01011100	xxxxxxxx
Corrompu	oui	non	non	oui	non	oui

2 Définition des opérations arithmétiques sur le corps

2.1 Définition du corps fini $\mathbb{F}_{2^8}$

Le codage de Reed-Solomon se fait sur un corps fini que nous allons caractériser dans la première partie de ce devoir. Durant le TP 4, vous avez construit un corps à $2^2 = 4$ éléments (exercice 4.6). Ici, pour obtenir un code plus puissant, nous allons construire un code à $2^8 = 256$ éléments. Pour cela, procédons étape par étape.

D'abord, rappelons que l'anneau $\mathbb{Z}_p = \{0, ..., p-1\}$ muni de l'addition $+$ et de la multiplication $\times$ est un corps (fini) si et seulement si p est premier. On notera ce corps $\mathbb{F}_p$, où p est un nombre premier. Dans ce devoir, nous nous attardons sur $p = 2$ qui est bien premier et donc sur :

$$\mathbb{F}_2 = \{0, 1\}. \tag{1}$$

Ce corps contient seulement deux éléments : 0 et 1, qui peuvent s'interpréter comme les booléens False et True, comme les chiffres binaires ou bien encore comme les valeurs possibles d'un bit.

A partir de $\mathbb{F}_2$, nous pouvons construire $\mathbb{F}_2[X]$ l'anneau des polynômes en X de degré quelconque à coefficients dans $\mathbb{F}_2$ (c'est-à-dire à coefficients binaires), muni de l'addition $+$ et de la multiplication $\times$:

$$\mathbb{F}_2[X] = \left\{ a(X) = \sum_{i=0}^{d-1} a_i X^i \;\middle|\; a_i \in \mathbb{F}_2, d \in \mathbb{N} \right\}. \tag{2}$$

Alors que $\mathbb{F}_2[X]$ possède un nombre infini d'éléments, on peut se restreindre à un nombre fini d'éléments en limitant le degré du polynôme, ce qui va s'avérer très bénéfique en pratique. En particulier, choisir un degré maximum strictement inférieur à $L \in \mathbb{N} \setminus \{0\}$ limite le nombre d'éléments à 2^L :

$$\mathbb{F}_{2^L} = \left\{ a(X) \in \mathbb{F}_2[X] \middle| \deg(a(X)) < L \right\} = \left\{ \sum_{i=0}^{L-1} a_i X^i \;\middle|\; a_i \in \mathbb{F}_2, \forall i \right\} \subset \mathbb{F}_2[X]. \tag{3}$$

Des détails complémentaires sur la définition de $\mathbb{F}_{2^L}$ sont disponibles en Section 5.1. Dans notre situation, nous nous intéressons au corps $\mathbb{F}_{2^8}$ ($L = 8$), où chaque élément peut s'exprimer comme un polynôme de degré au plus 7 dont les coefficients sont dans $\mathbb{F}_2$. Nous avons donc un polynôme de

la forme $a_7X^7+a_6X^6+...+a_1X+a_0$, où $a_i \in \mathbb{F}_2, \forall i \in \{0,...,7\}$. Chaque coefficient correspond donc à un bit. Le polynôme peut donc être encodé par une suite de 8 bits correspondant donc à un octet. Par exemple, le mot/l'octet 10101100 est représenté par le polynôme $a(X) = X^7 + X^5 + X^3 + X^2$.

Jusqu'ici, X (appelé l'indéterminée) ne représente rien de concret. On peut même complètement l'oublier en utilisant la forme octet. Le polynôme en X est parfois appelé "polynôme formel" pour souligner qu'il encode une suite de coefficients, plutôt qu'une fonction de $\mathbb{F}_2$ dans $\mathbb{F}_2$, où X serait considéré comme une variable à valeurs dans $\mathbb{F}_2$. Par exemple, X^2 et X^3 sont considérées comme des polynômes (formels) différents, alors qu'en tant que fonctions de $\mathbb{F}_2$ dans $\mathbb{F}_2$, que l'on nommerait $f(X)$ et $g(X)$, ils sont indistinguables. En effet, en tous les points de leur domaine $\{0,1\}$, ces fonctions prennent la même valeur : $f(0) = g(0) = 0$ et $f(1) = g(1) = 1$. On pourrait faire le parallèle avec les fonctions génératrices en probabilités, qui seront vues plus loin dans ce cours, où l'indéterminée est également "formelle", sans valeur concrète.

On souhaite à présent munir $\mathbb{F}_{2^8}$ d'une addition et d'une multiplication, qui feront de $\mathbb{F}_{2^8}$, non seulement un anneau, mais également un corps !

2.2 Addition

Pour rappel, la table de Cayley pour l'addition dans $\mathbb{F}_2$ est :

+	0	1
0	0	1
1	1	0

Dans $\mathbb{F}_{2^8}$, on applique l'addition entre deux polynômes en additionnant les coefficients des termes de même puissance en suivant les règles de l'addition dans $\mathbb{F}_2$, ce qui revient à réaliser un opération XOR. Par exemple, considérons les polynômes $a, b \in \mathbb{F}_{2^8}$ avec $a(X) = X^2 + X$ et $b(X) = X^3 + X$. On trouve leur somme en appliquant une fonction XOR :

+	X^7	X^6	X^5	X^4	X^3	X^2	X	1
$a(X)$	0	0	0	0	0	1	1	0
$b(X)$	0	0	0	0	1	0	1	0
$a(X) + b(X)$	0	0	0	0	1	1	0	0

ce qui nous donne $a(X) + b(X) = X^3 + X^2$.

L'élément neutre (ou l'identité) pour l'addition est évidemment le polynôme nul : 0 (c'est-à-dire $0X^7 + 0X^6 + 0X^5 + 0X^4 + 0X^3 + 0X^2 + 0X + 0$, ou encore 00000000). On remarque aussi que tout élément du corps est son propre inverse additif : $a = -a$. En effet, additionner un polynôme à lui-même donne le polynôme nul. La soustraction et l'addition (et l'opération XOR) sont donc identiques dans $\mathbb{F}_{2^8}$.

2.3 Multiplication

La multiplication dans $\mathbb{F}_{2^8}$ se base évidemment sur la multiplication dans $\mathbb{F}_2$. On rappelle donc la table de Cayley pour la multiplication dans $\mathbb{F}_2$, qui correspond à une opération AND :

$\times$	0	1
0	0	0
1	0	1

La multiplication s'étend très naturellement à $\mathbb{F}_2[X]$, en suivant les règles familières de multiplication de polynômes classiques. Comme pour les polynômes dans $\mathbb{R}$, les degrés s'additionnent, et les coefficients se multiplient en suivant la multiplication dans $\mathbb{F}_2$. Par exemple, si $a(X) = X^7 + X^6 + X^4 + X^2 + X$ et $b(X) = X^5 + X^3 + X$, alors $a(X) \times b(X) = X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^4 + X^3 + X^2$. On peut représenter cette multiplication dans $\mathbb{F}_2[X]$ sous forme de tableau, dans lequel on décompose la multiplication de $a(X)$ par chacun des termes de $b(X)$, puis on fait la somme des produits (à la manière d'un calcul écrit) :

$\times$	X^{14}	X^{13}	X^{12}	X^{11}	X^{10}	X^9	X^8	X^7	X^6	X^5	X^4	X^3	X^2	X	1
$a(X)$	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0
$b(X)$	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
$a(X) \times X$	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0
$a(X) \times X^3$	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0
$a(X) \times X^5$	0	0	1	1	0	1	0	1	1	0	0	0	0	0	0
$a(X) \times b(X)$	0	0	1	1	1	0	1	1	1	0	1	1	1	0	0

En général, soient $a(X) = \sum_{i=0}^{\deg(a)} a_i X^i \in \mathbb{F}_2[X]$ et $b(X) = \sum_{i=0}^{\deg(b)} b_i X^i \in \mathbb{F}_2[X]$. Le produit appartient alors bien à $\mathbb{F}_2[X]$:

$$a(X) \times b(X) = \left(\sum_{i=0}^{\deg(a)} a_i X^i \right) \times \left(\sum_{i=0}^{\deg(b)} b_i X^i \right) = \sum_{i=0}^{\deg(a)+\deg(b)} \left(\sum_{j=0}^{i} a_{i-j} \times b_j \right) X^i \qquad (4)$$

Cependant, cette multiplication dans $\mathbb{F}_2[X]$ ne peut pas s'étendre à la multiplication dans $\mathbb{F}_{2^8}$. En effet, le produit $(a \times b)(X)$ sortirait de $\mathbb{F}_{2^8}$ si $a(X), b(X) \in \mathbb{F}_{2^8}$ car $a(X) \times b(X)$ pourrait être de degré plus grand ou égal à $L = 8$, jusqu'à $2(L-1) = 14$ (selon les valeurs des coefficients a_i et b_i). Afin de ramener tout élément de $\mathbb{F}_2[X]$ dans $\mathbb{F}_{2^8}$, la multiplication dans $\mathbb{F}_{2^8}$ requiert alors l'utilisation d'un polynôme $P(X)$ de degré exactement 8 à coefficients dans $\mathbb{F}_2$ pour effectuer une réduction modulaire. De plus, $P(X)$ doit nécessairement être irréductible. Pour plus de détails, voir Section 5.2.

De la même façon que dans $\mathbb{F}_2$ on réduit tout élément $n \in \mathbb{N}$ en ne gardant que le reste de sa division par $p = 2$ (c'est-à-dire que $n \mod 2 = r$ où $n = q.2 + r$ avec $r \in \mathbb{F}_2$), on réduira

maintenant tout polynôme de degré quelconque $n(X) \in \mathbb{F}_2[X]$ en ne gardant que le reste de la division euclidienne de ce polynôme par $P(X)$. Autrement dit, si $n(X) = q(X) \times P(X) + r(X)$ avec $r(X) \in \mathbb{F}_{2^8}$, alors $n(X) \mod P(X) = r(X)$, ou plus simplement dans $\mathbb{F}_{2^8}$: $n(X) = r(X)$.

Voici donc le même exemple de multiplication, cette fois-ci dans $\mathbb{F}_{2^8}$. On choisit pour l'exemple le polynôme irréductible $P(X) = X^8 + X^6 + X^3 + X^2 + 1$, c'est-à-dire 101001101. Le produit vaut $a(X) \times b(X) = r(X) = X^7 + X^5 + X^4 + X + 1$. En effet, dans $\mathbb{F}_2[X]$, on a bien $n(X) = X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^4 + X^3 + X^2 = (X^4 + X^3 + X + 1) \times (X^8 + X^6 + X^3 + X^2 + 1) + (X^7 + X^5 + X^4 + X + 1) = q(X) \times P(X) + r(X)$. Sous forme de tableau, cela donne :

$\times$	X^8	X^7	X^6	X^5	X^4	X^3	X^2	X	1
$a(X)$	0	1	1	0	1	0	1	1	0
$b(X)$	0	0	0	1	0	1	0	1	0
$P(X)$	1	0	1	0	0	1	1	0	1
$a(X) \times b(X)$	0	1	0	1	1	0	0	1	1

De manière générale, la multiplication dans $\mathbb{F}_{2^8}$ s'écrit donc :

$$a(X) \times b(X) = \left(\sum_{i=0}^{7} a_i X^i\right) \times \left(\sum_{i=0}^{7} b_i X^i\right) \mod P(X) = \sum_{i=0}^{14} \left(\sum_{j=0}^{i} a_{i-j} \times b_j\right) X^i \mod P(X) \quad (5)$$

La formule présentée en (5) possède un gros défaut : elle demande de sortir temporairement de $\mathbb{F}_{2^8}$ avant d'effectuer la division euclidienne par $P(X)$. Cependant, il est possible de construire un algorithme simple et rapide afin de faire une multiplication dans $\mathbb{F}_{2^8}$ sans jamais sortir de $\mathbb{F}_{2^8}$ durant les étapes intermédiaires. Cet algorithme est dérivé dans la Section 5.3 et est détaillé dans l'Algorithm 1.

Algorithm 1: Multiplication sur $\mathbb{F}_{2^8}$

```
1 def Multiply(a,b):
2     mult ← 0
3     while b ≠ 0 do
4         if coeff b_0 est égal à 1 then
5             mult ← mult + a
6             b ← b + 1
7         a ← (X × a)%P
8         b ← b/X
9     return mult
```

Dans $\mathbb{F}_{2^8}$, l'identité (ou l'élément neutre) pour la multiplication est évidemment le polynôme unitaire : 1 (c'est-à-dire $0X^7 + 0X^6 + 0X^5 + 0X^4 + 0X^3 + 0X^2 + 0X + 1$, ou encore 00000001).

2.4 Inversion

Dans $\mathbb{F}_{2^8}$, chaque élément non-nul $a \neq 00000000$ possède un inverse multiplicatif a^{-1}, c'est-à-dire dont le produit avec a vaut l'identité : $a \times a^{-1} = a^{-1} \times a = 1$. De plus, on peut facilement le calculer car il est égal à $a^{-1} = a^{254}$. En effet, les 255 éléments non-nuls du corps $\mathbb{F}_{2^8}$ forment un groupe pour la multiplication, puisqu'ils sont tous inversibles. Puisque c'est un groupe, alors le théorème d'Euler-Fermat vu au cours nous dit que $a^{255} = e = 1$. Donc $a^{254} \times a = a \times a^{254} = 1$, ce signifie bien que a^{254} est l'inverse de a : $a^{254} = a^{-1}$.

Pour calculer cet inverse, il y a un moyen plus astucieux que d'effectuer 253 multiplications. On peut calculer $a^2, a^4, ..., a^{128}$ par mises au carré successives ($a \times a, a^2 \times a^2, ..., a^{64} \times a^{64}$), ce qui demande 7 multiplications. Ensuite, selon l'observation que $254 = 128+64+32+16+8+4+2$, on multiplie ces résultats : $a^{254} = a^{128+64+32+16+8+4+2} = a^{128} \times a^{64} \times a^{32} \times a^{16} \times a^8 \times a^4 \times a^2$, ce qui ne requiert que 6 multiplications. Au total, nous avons besoin de seulement $7+6 = 13$ multiplications.

2.5 A faire

Compléter les fonctions `add`, `multiply` et `inverse` dans le fichier `reed_solomon.py` selon les consignes données dans le fichier. Ces 3 fonctions doivent être suffisamment rapides car elles seront réutilisées dans la seconde partie du devoir. Les détails quant à l'implémentation sont donnés dans le fichier `reed_solomon.py`.

3 Encodage et décodage

3.1 Encodage

Comme défini au cours, le code est une fonction

$$c : (\mathbb{F}_{2^8})^k \to (\mathbb{F}_{2^8})^n \,, \tag{6}$$

où k est la dimension du message à transmettre (le nombre d'octets à transmettre) et n est la taille de bloc que l'on veut transmettre (le nombre d'octets que l'on transmet après encodage), avec $n \geqslant k$. On transmettra donc un surplus d'information, qui sera utile pour récupérer le message initial malgré les effacements durant la transmission.

Dans ce devoir, on souhaite envoyer une phrase écrite en français de k symboles. D'abord, chaque symbole de la phrase est transformé en un octet $a_i \in \mathbb{F}_{2^8}$ via le code ASCII, $i \in \{0, ..., k-1\}$, celle-ci peut donc être représentée par une suite de k octets. Ainsi, le message initial est de la forme

$$\mathbf{a} = (a_0, a_1, ..., a_{k-1}) \in (\mathbb{F}_{2^8})^k \tag{7}$$

Précédemment, on avait définit $\mathbb{F}_2[X]$ comme l'ensemble des polynômes formels de degré quelconque en l'indéterminée X à coefficients dans $\mathbb{F}_2$. Dans la même manière, on définit désormais $\mathbb{F}_{2^8}[Y]$ comme l'ensemble des polynômes formels de degré quelconque en l'indéterminée Y à coefficients dans $\mathbb{F}_{2^8}$:

$$\mathbb{F}_{2^8}[Y] = \left\{ a(Y) = \sum_{i=0}^{d-1} a_i Y^i \;\middle|\; a_i \in \mathbb{F}_{2^8}, d \in \mathbb{N} \right\}. \tag{8}$$

On note que les coefficients a_i, qui sont des éléments de $\mathbb{F}_{2^8}$, sont eux-mêmes des polynômes en l'indéterminée X. Donc, chaque élément de $\mathbb{F}_{2^8}[Y]$ est un polynôme en Y dont les coefficients sont des polynômes en X : un polynôme de polynômes. Pour ne pas tout mélanger, on considère uniquement la forme octet des coefficients a_i, pour ne plus avoir à affaire avec l'indéterminée X. On peut donc considérer qu'on s'intéresse à des éléments de $\mathbb{F}_{2^8}[Y]$ qui sont des polynômes en Y avec des octets comme coefficients. Ainsi, le message initial $\mathbf{a}$ définit un unique polynôme formel $A(Y) = a_0 + a_1 Y + ... + a_{k-1} Y^{k-1} \in \mathbb{F}_{2^8}[Y]$, de degré $k-1$ en Y avec des coefficients $a_i \in \mathbb{F}_{2^8}$.

Alors que jusqu'ici, Y était simplement une indéterminée (il ne représentait pas une variable à valeur dans un ensemble particulier), on lui impose désormais un ensemble : Y sera évalué en des éléments de $\mathbb{F}_{2^8}$, c'est-à-dire en des octets. Pourquoi choisir de travailler avec Y dans $\mathbb{F}_{2^8}$? C'est bien simple : ce corps permet de représenter et de travailler avec des octets, omniprésents en informatique, de manière efficace.

On remarque que lorsqu'on évalue $A(Y)$ en un point $Y = y \in \mathbb{F}_{2^8}$, alors $A(y) \in \mathbb{F}_{2^8}$, puisqu'il se calcule par des multiplications et des additions dans $\mathbb{F}_{2^8}$. Pour encoder le message, on choisit $n \geqslant k$ éléments distincts $y_i \in \mathbb{F}_{2^8}, i \in \{0, ..., n-1\}$, que l'on nomme les points d'interpolation y_i. On notera $\mathbf{y} = (y_0, y_1, ..., y_{n-1}) \in (\mathbb{F}_{2^8})^n$. On publie $\mathbf{y}$ ainsi que le polynôme irréductible $P(Y)$ choisi avec la définition du code (qui comprend les nombres k et n). Le message envoyé est constitué de n éléments appelés "mots-codes" qui sont les évaluations du polynôme $A(Y)$ aux points d'interpolation $\mathbf{y}$, et que l'on nomme $\mathbf{I}$ (pour "Interpolation") :

$$\mathbf{I} = (I_0, I_1, ..., I_{n-1}) = (A(y_0), A(y_1), ..., A(y_{n-1})) \in (\mathbb{F}_{2^8})^n \tag{9}$$

Le message envoyé est donc une séquence de n mots-codes de 8 bits, qui proviennent tous du même polynôme $A(Y)$ de degré $k-1$ en Y. Pour être cohérent avec la notation de code définie en (6), on établit l'égalité $c(\mathbf{a}) = \mathbf{I}$.

3.2 Décodage

Durant la transmission du message, des octets vont être effacés. On peut facilement imaginer cela dans le cadre d'une communication Terre-satellite, ou d'une griffe sur un CD par exemple. Important : dans ce devoir, on suppose que tous les octets corrompus sont facilement localisables, car ils souffrent d'effacement, comme présenté dans la Section 1.

Nous entrons dans le vif du sujet : comment décoder ? C'est-à-dire, comment utiliser le surplus d'information envoyé afin de reconstituer le message initial malgré des octets effacés ?

Utilisons un exemple pour comprendre. Supposons que $k = 4$ et $n = 6$. Une fois qu'on dispose du message original composé de 4 octets a_0, a_1, a_2, a_3, on définit le polynôme formel $A(Y) := a_0 + a_1 Y + a_2 Y^2 + a_3 Y^3$. On se choisit ensuite 6 éléments (non-nuls) différents du corps, par exemple $y_0 = 00001000$, $y_1 = 00000001$, $y_2 = 00000010$, $y_3 = 00001100$, $y_4 = 01000000$, $y_5 = 10001001$. On calcule les six valeurs $(I_0, ..., I_5) = (A(y_0), ..., A(y_5))$ et on les transmet sur le canal. Si deux octets parviennent effacés, par exemple I_3 et I_4, qu'à cela ne tienne : $A(Y)$ est un polynôme de degré 3 en Y, donc il suffit de connaître sa valeur en 4 points distincts, c'est-à-dire 4 mots-codes qui n'ont pas été effacés durant la transmission (I_0, I_1, I_2 et I_5). On peut alors retrouver les coefficients du polynôme d'origine, donc le message transmis.

Afin de retrouver les coefficients a_i (et donc le message initial), on doit résoudre un système de k équations linéaires indépendantes, qui correspondent à k mots-codes qui n'ont pas été effacés.

On choisit donc k indices i_j parmi les n, avec $j \in \{0, ..., k-1\}$. Alors, l'équation correspondant au $j^{\text{ème}}$ mot-code choisi, autrement dit à y_{i_j}, est alors :

$$A(y_{i_j}) = a_0 + a_1 y_{i_j} + ... + a_{k-1} {y_{i_j}}^{k-1} \tag{10}$$

Le problème se réduit à trouver les k coefficients $\mathbf{a} = (a_0, ..., a_{k-1})$ qui permettent que les k égalités soit respectées. Autrement dit, il s'agit d'effectuer une interpolation polynomiale sur le corps fini, qui est définie de manière unique ! Pour plus de détails, voir Section 5.4. Le système linéaire s'écrit alors

$$V\mathbf{a} = \mathbf{I_{restr}} \tag{11}$$

où $\mathbf{I_{restr}}$ est la restriction du vecteur $\mathbf{I}$ aux k indices sélectionnés, et où V est la matrice de Vandermonde. L'utilisation de cette matrice dans le cadre de l'interpolation a déjà été vue dans le cours de Méthodes Numériques (LEPL1104), preuve que les cours se recoupent. La différence par rapport à LEPL1104, c'est que dans ce cas-ci, les entrées de V sont des éléments de $\mathbb{F}_{2^8}$:

$$V = \begin{pmatrix} 1 & y_{i_0} & \cdots & (y_{i_0})^{k-1} \\ 1 & y_{i_1} & \cdots & (y_{i_1})^{k-1} \\ \vdots & & & \vdots \\ 1 & y_{i_{k-1}} & \cdots & (y_{i_{k-1}})^{k-1} \end{pmatrix} \tag{12}$$

Notez bien que le nombre maximum d'octets effacés ne peut excéder $n-k$. S'il s'avérait que plus de $n-k$ octets étaient effacés, alors il n'est plus possible de retrouver exactement le message initial, puisque l'interpolation ne serait pas déterminée uniquement (et on ne pourrait pas savoir quelle solution est la bonne). Dans le devoir, vous devez aussi être capable de détecter cette situation.

Il est maintenant nécessaire de résoudre le système linéaire. Pour cela, nous utilisons une élimination gaussienne classique décrite dans l'Algorithme 2, en faisant appel aux opérations d'addition, de multiplication et d'inverse définies sur $\mathbb{F}_{2^8}$ que vous avez implémentées dans la première partie du devoir.

Algorithm 2: Elimination Gaussienne

```
  // On souhaite résoudre le système Va = I
  // P est la matrice augmentée: P = [V|I]
1 def gaussian_elimination(P):
2     for i in range(k) do
3         for j in range(k) do
4             if i ≠ j then
5                 r ← P[j,i]/P[i,i]
6                 for m in range(k+1) do
7                     P[j,m] ← P[j,m] − r × P[i,m]
8     for i in range(k) do
9         x[i] ← P[i,k]/P[i,i]
10    return x
```

Une fois le message décodé, on peut le retraduire en français grâce au code ASCII. Si tout s'est bien passé, on retrouve alors la phrase que l'on souhaitait envoyer initialement.

En réalité, ce devoir n'implémente pas l'entièreté du code de Reed-Solomon, mais seulement une partie. Celui-ci est encore plus puissant : il permet également de détecter et de corriger d'autres types de corruptions beaucoup plus subtiles comme des "bit flips", où des 1 deviennent des 0 et inversement. C'est quasiment magique, mais c'est également hors du contexte du cours (voir Section 5.7).

3.3 A faire

Compléter les fonctions `encoding`, `gaussian_elimination` et `decoding` décrites dans le fichier `reed_solomon.py`. Notez bien que vos fonctions `add`, `multiply` et `inverse` doivent être correctement codées et suffisament rapides pour que le code complet puisse fonctionner. Les détails quant à l'implémentation sont donnés dans le fichier `read_solomon.py`.

4 A retenir

On se rend compte qu'une fois que le corps et l'algèbre sur ce corps sont correctement définis (éléments, opérations, identités, inverses, ...), on peut appliquer les mêmes principes de calcul que les règles de l'algèbre classique, et que tout fonctionne correctement ! De plus, certains théorèmes très puissants (comme le théorème de l'interpolation polynomiale) sont également valides sur ce corps. En termes imprécis, des ensembles très différents à première vue fonctionnent en profondeur de la même façon si jamais ils ont la même structure algébrique. Cette capacité de généralisation des structures algébriques en font un outil mathématique très puissant.

5 Compléments d'informations

5.1 Définition de $\mathbb{F}_{2^L}$ (suite)

Dans le cours, il a été vu que $\mathbb{Z}_n$ (l'anneau des entiers modulo n) est un corps fini $\mathbb{F}_n$ si et seulement si n est premier ($n = p$). Lorsque $p = 2$, on obtient bien $\mathbb{F}_2$ dont les éléments sont $\{0, 1\}$. Cependant, $q = 2^L = p^L$ n'est pas premier dès que $L > 1$. La notation $\mathbb{F}_{2^L}$ n'est-elle alors pas abusive ? Non, seulement trompeuse. En effet, $\mathbb{F}_{2^L}$ est différent de $\mathbb{Z}_{2^L}$, puisque $\mathbb{F}_{2^L}$ est un corps fini de polynômes et alors que $\mathbb{Z}_{2^L}$ est un anneau d'entiers.

De plus, on peut prouver que $\mathbb{F}_{2^L}$ est bel et bien un corps fini, souvent appelé $\mathrm{GF}(2^L)$ en anglais (pour Galois field). Il est possible de construire ce corps comme un quotient qui fait intervenir $P(X)$, un polynôme irréductible de degré L à coefficients dans $\mathbb{F}_2$:

$$\mathbb{F}_{2^L} = \mathbb{F}_2[X]/P(X). \tag{13}$$

Trois théorèmes permettent d'utiliser $\mathbb{F}_{2^L}$ sans ambiguïté. D'abord, $\mathbb{F}_{2^L}$ est un corps fini si et seulement $P(X)$ est un polynôme irréductible de degré L, comme expliqué en partie en Section 5.2. Ensuite, il existe toujours au moins un polynôme irréductible de degré L. Enfin, tous les polynômes

irréductibles de $\mathbb{F}_2[X]$ de degré L produisent des corps identiques, à un isomorphisme près. Ce qui nous autorise à dire "le corps à 2^L éléments" (à un isomorphisme près). La nature est bien faite.

5.2 Polynôme $P(X)$ pour la multiplication dans $\mathbb{F}_{2^8}$

Le polynôme $P(X)$ doit suivre 3 critères. D'abord, de manière évidente, il est nécessaire que $P(X)$ ait des coefficients dans $\mathbb{F}_2$. Ensuite, il doit être de degré exactement 8 afin d'assurer que le reste de la réduction modulaire soit de degré au plus 7. Notons que si le degré du produit est de degré inférieur ou égal à 7, il n'est pas nécessaire de faire appel à $P(X)$. Le troisième critère est que le polynôme soit irréductible, c'est-à-dire non-factorisable avec des polynômes de plus petits degrés à coefficients dans $\mathbb{F}_2$. Cela permet que tout élément non-nul de $\mathbb{F}_{2^8}$ ait un inverse défini de manière unique.

Ce dernier point mérite plus d'explications. D'abord, il faut savoir que choisir n'importe quel polynôme de degré 8 pour la réduction modulaire permettrait de définir un anneau de polynômes de degré au plus 7 (qui serait l'ensemble quotient de $\mathbb{F}_2[X]$ par ce polynôme). Cependant, cet anneau n'aurait pas certaines propriétés désirables. En particulier, il pourrait contenir des diviseurs non-nuls de zéro, c'est-à-dire que le produit de deux éléments non-nuls donneraient un produit nul. Par exemple, si l'on choisit $P(X) = X^8 + X^6 + X^2 + 1 = (X^6 + 1)(X^2 + 1)$, alors $(X^6 + 1) \times (X^2 + 1) = 0$ dans l'anneau construit, bien que les deux facteurs soient non-nuls. Pour se rattacher à des choses connues, le même phénomène se produit dans $\mathbb{Z}_n$ si n n'est pas premier. Si $n = 6$, alors $3 \times 2 = 6 = 0$, donc 3 et 2 sont des diviseurs non-nuls de zéro. Pour un polynôme dans $\mathbb{F}_2[X]$, être irréductible est l'équivalent d'être premier pour un nombre dans $\mathbb{Z}^+$.

De plus, dans un anneau, un diviseur non-nul de zéro ne peut pas être inversible. En effet, un élément d'un anneau ne peut pas être à la fois inversible et un diviseur non-nul de zéro. Preuve par contradiction : supposons que $a \neq 0$ soit inversible (il existe $a^{-1} \neq 0$ tel que $a \times a^{-1} = a^{-1} \times a = 1$) et que a soit un diviseur non-nul de zéro (il existe $b \neq 0$ tel que $a \times b = 0$), alors $b = 1 \times b = a^{-1} \times a \times b = a^{-1} \times 0 = 0$, ce qui est une contradiction.

Donc, pour qu'il n'y ait pas de diviseur de zéro et que chaque élément non-nul ait un inverse, on demande que $P(X)$ soit irréductible. Ainsi, en choisissant un $P(X)$ irréductible, les éléments non-nuls de $\mathbb{F}_{2^8}$ forment ce qu'on appelle un groupe fini pour la multiplication. Par définition, cela implique que $\mathbb{F}_{2^8}$ est un corps fini.

Au fait, comment trouver un polynôme irréductible $P(X)$ pour la construction de $\mathbb{F}_{2^8}$? On peut en fait effectuer la construction avec n'importe quel polynôme qui respecte les trois conditions. Il a donc suffi de trouver un polynôme irréductible de degré 8 à coefficients dans $\mathbb{F}_2$ et le tour était joué. Ceci peut être fait en effectuant une recherche exhaustive, en utilisant des algorithmes tels que Berlekamp ou Cantor-Zassenhaus, ou encore en vérifiant des critères tels que celui d'Eisenstein. Le plus simple est encore d'aller piocher dans des tables existantes, par exemple sur `https://www.ece.unb.ca/tervo/ece4253/polyprime.shtml`.

5.3 Algorithme pour la multiplication dans $\mathbb{F}_{2^8}$

Une idée simple pour multiplier deux polynômes $a(X)$ et $b(X)$ dans $\mathbb{F}_{2^8}$ est de les multiplier dans $\mathbb{F}_2[X]$, puis d'effectuer une division euclidienne du produit par $P(X)$ et de ne garder que le reste, comme le propose l'équation (5). Cependant, on veut pouvoir effectuer cette multiplication

sans jamais sortir de $\mathbb{F}_{2^8}$, même dans les étapes intermédiaires. Pourquoi ?

Comme chaque élément de $\mathbb{F}_{2^8}$ est équivalent à un octet, il peut être représenté en mémoire par exemple par un *Integer 8* (qui existe dans un grand nombre de langages informatiques, notamment en Python). Or, sortir temporairement de $\mathbb{F}_{2^8}$ demanderait plus qu'un octet et nécessite donc de sortir temporairement de la représentation en *Integer 8*, ce qui est indésirable. D'une part, certaines machines pourraient ne fonctionner qu'avec des *Integer 8*. D'autre part, la quantité de stockage augmenterait. Notons bien que dans ce devoir, il n'est pas obligatoire d'utiliser des *Integer 8* de Python, mais pour d'autres applications industrielles, c'est une vraie préoccupation.

Heureusement, il est possible de définir un algorithme pour la multiplication qui ne sorte jamais de $\mathbb{F}_{2^8}$. Définissons $a(X) = \sum_{i=0}^{7} a_i X^i$ et $b(X) = \sum_{i=0}^{7} b_i X^i$. Leur produit dans $\mathbb{F}_{2^8}$ s'écrit alors :

$$a(X) \times b(X) \mod P(X) = a(X) \times \left(\sum_{i=0}^{7} b_i X^i \right) \mod P(X) \tag{14}$$

L'idée est de calculer le produit un terme de b à la fois, en commençant par b_0. On voit que $a(X)b_0$ est de degré 7 maximum, et donc que ce terme ne nécessite pas d'autres calculs. On met X en évidence dans le second terme, faisant apparaître le produit de $p(X) = a(X)X$ (de degré au plus 8) et $q(X) = \sum_{i=1}^{7} b_i X^{i-1}$ (de degré au plus 6).

$$a(X) \times b(X) \mod P(X) = a(X)b_0 + \left(a(X)X \times \left(\sum_{i=1}^{7} b_i X^{i-1} \right) \mod P(x) \right) \tag{15}$$

En utilisant la propriété que $p(X) \times q(X) \mod P(X) = (p(X) \mod P(X)) \times (q(X) \mod P(X)) \mod P(X)$ et le faible degré de $q(X)$, on peut réécrire le produit initial comme ceci :

$$a(X) \times b(X) \mod P(X) = a(X)b_0 + \left((a(X)X \mod P(X)) \times \left(\sum_{i=1}^{7} b_i X^{i-1} \right) \mod P(X) \right) \tag{16}$$

Il suffit alors de calculer $a(X)X$ modulo $P(X)$. Cette opération est réalisable sans trop de difficulté en réfléchissant en termes d'opérations sur les bits, et implémentée intelligemment, permet de ne pas sortir de $\mathbb{F}_{2^8}$. Si l'on nomme $a(X)\Big|_1 = a(X)X \mod P(X)$ (de degré au plus 7), on retrouve pour le second terme une forme semblable au produit initial, avec $a(X)\Big|_1$ qui remplace $a(X)$ et $b(X)$ qui a perdu un degré et son premier coefficient.

$$a(X) \times b(X) \mod P(X) = a(X)b_0 + \left(a(X)\Big|_1 \times \left(\sum_{i=1}^{7} b_i X^{i-1} \right) \mod P(X) \right) \tag{17}$$

L'opération peut être répétée itérativement avec les degrés suivants. Si l'on nomme $a(X)\Big|_i = a(X)\Big|_{i-1}.X \mod P(X)$ et $a(X)\Big|_0 = a(X)$, on obtient la formule suivante pour la multiplication dans $\mathbb{F}_{2^8}$ sous forme d'une somme d'éléments de $\mathbb{F}_{2^8}$:

$$a(X) \times b(X) \mod P(X) = \sum_{i=0}^{7} b_i a(X)\Big|_i \tag{18}$$

Cette formule se traduit très facilement en algorithme. De plus, en réfléchissant intelligemment aux spécificités des opérations dans $\mathbb{F}_2$ (en termes d'opérations sur les bits par exemple, telles que XOR, AND, décaler vers la gauche, décaler vers la droite), on peut simplifier et accélérer cet algorithme, et cela donne l'Algorithm 1 que l'on vous demande d'implémenter pour effectuer des multiplication dans $\mathbb{F}_{2^8}$. Notons que l'équation (18) et l'algorithme pourraient s'étendre facilement à $\mathbb{F}_{2^L}$, tant que l'on a un polynôme $P(X)$ irréductible de degré L.

5.4 Interpolation polynomiale sur corps fini

Il est bien connu qu'un polynôme de degré au plus $k-1$ à coefficients réels est déterminé de manière unique par k points d'interpolation (x_i, y_i) si les abscisses x_i sont distinctes, où $i \in \{0, ..., k-1\}$. Ce théorème, vu par exemple dans le cours de méthodes numériques LEPL1104, est appelé le *théorème d'interpolation polynomiale de Lagrange*. Ce théorème, vrai pour les polynômes de degré $k-1$ sur les réels, est également vrai pour les polynômes de degré $k-1$ dans $\mathbb{F}_{2^8}[Y]$, puisque $\mathbb{F}_{2^8}$ est un corps. Pour le démontrer, on utilise le fait que dans un corps, un polynôme de degré $k-1$ possède au plus k racines distinctes. Les détails sont fournis en partie dans les slides du cours. Cela signifie que l'interpolation polynomiale est unique sur $\mathbb{F}_{2^8}$.

5.5 Le code de Reed-Solomon en pratique

Le code de Reed-Solomon peut être défini sur n'importe quelle valeur de $L \geqslant 1$, en utilisant le corps fini $\mathbb{F}_{2^L}$ qui contient 2^L éléments. Dans la pratique, on utilise souvent $L = 8$ comme nous l'avons fait, car la majorité des systèmes informatiques fonctionnent avec des octets (ou des *Integer 8*). Par contre, on peut utiliser des valeurs de k et de n beaucoup plus grandes que ce qui est proposé dans les exemples de l'énoncé ou bien dans les tests, en restant évidemment limité par le nombre d'éléments du corps fini : 2^L. Par exemple, sur l'ADSL, on utilise typiquement des polynômes de degré 238 ($k = 239$) évalués en les $n = 255$ éléments non-nuls du corps, ce qui permet donc de récupérer 16 octets effacés. En général, on utilise des codes de Reed-Solomon avec $n = 2^L - 1$ où l'on n'utilise pas l'élément nul pour l'évaluation du polynôme car il n'a pas d'inverse.

A noter que la capacité du code de Reed-Solomon à récupérer des "trains" d'erreurs est fort commode pour des applications où les erreurs ne sont typiquement pas isolées. Sur un CD ou DVD par exemple, la moindre poussière ou griffe efface un grand nombre de bits consécutifs.

5.6 Pourquoi utiliser des corps finis ?

Pourquoi avons nous utilisé des polynômes sur un corps fini plutôt que sur les réels ou sur les entiers, par exemple ? Après tout, ce code ne fait que de l'interpolation polynomiale, qui fonctionne aussi sur les réels, et cela nous dispenserait de toute la théorie des corps finis. La raison est bien simple : évaluer un polynôme de haut degré à valeurs dans $\mathbb{R}$ ou même dans $\mathbb{Z}$ en beaucoup de points peut générer des nombres effroyablement grands, à plusieurs centaines voire milliers de chiffres binaires, là où un polynôme à valeurs dans $\mathbb{F}_{2^8}$ a toujours une valeur exprimée avec un seul octet. Cette expansion de la taille des nombres manipulés augmenterait considérablement la bande passante nécessaire pour communiquer, ou l'espace de stockage dont on a besoin, sans pour autant apporter de bénéfice en termes de capacité de correction d'erreurs : celle-ci dépend des degrés des polynômes utilisés, qui ne change pas.

Par exemple, l'ADSL envoie des messages de taille $k = 239$ avec une taille de bloc $n = 255 = 2^8 - 1$. Si on choisit de travailler sur $\mathbb{Z}$, on prend comme meilleur choix $y_i = i + 1$ (pour éviter d'utiliser 0 qui n'a pas d'inverse). On a donc $y_{n-1} = 255$, alors $A(y_{n-1})$ pourrait impliquer le calcul de ${y_{n-1}}^{k-1} = 255^{239}$ qui vaut à peu près 2^{1911}. Ainsi, au lieu de transmettre 1 octet en calculant $A(y_{n-1})$ dans $\mathbb{F}_{2^8}$, on devrait en transmettre $\lceil \frac{1911}{8} \rceil = 239$ si on calcule dans $\mathbb{Z}$, ce qui est à la fois très coûteux en ressources de calcul et en bande passante pour la communication ou le stockage. Et ce sans amélioration de notre capacité à corriger des erreurs...

5.7 Le code de Reed-Solomon dans toute sa puissance

Dans ce devoir, les localisations des octets corrompus sont connues car ceux-ci souffrent d'effacements de bits. Dans ce cas, le code de Reed-Solomon fonctionne malgré au plus $n - k$ effacements d'octets. Cependant, en réalité, des corruptions différentes peuvent survenir, comme des bit flips : un 1 devient un 0, ou un 0 devient un 1. Tous les octets du message transmis seront lisibles et il n'y a *a priori* pas d'information sur la localisation des "erreurs". Retrouver le message est-il alors impossible ? Non, peu importe ! Le code (complet) de Reed-Solomon est également capable de détecter ce genre d'erreurs. Le nombre d'erreurs détectables est cependant deux fois plus petit que le nombre d'effacements détectables : il peut gérer au plus $\lfloor \frac{n-k}{2} \rfloor$ erreurs. Par exemple, l'ADSL peut gérer $n - k = 16$ effacements, ou alors $\lfloor \frac{n-k}{2} \rfloor = 8$ erreurs. Evidemment, un mix d'effacements et d'erreurs peut également être corrigé, par exemple 10 effacements et 3 erreurs.

Notez bien que la partie de l'algorithme de Reed-Solomon qui permet cette prouesse est très complexe : elle se compose de plusieurs sous-algorithmes imbriqués et est basée sur une bonne dose de mathématiques avancées. Dans notre grande bonté, nous vous épargnons de la comprendre et de la coder. Si vous souhaitez tout de même en savoir plus sur la puissance du code de Reed-Solomon et sur la théorie des codes, nous vous conseillons de vous orienter vers les cours suivants : LINGI2348 (Information theory and coding) et LMAT2460 (Mathématiques discrètes - Structures combinatoires).

Finalement, pourrait-on faire mieux que Reed-Solomon ? En un sens, non. En effet, le code de Reed-Solomon est "optimal" dans le sens où il atteint la borne de Singleton : parmi tous les codes, ils corrigent le max d'erreurs possible étant donné n et k