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

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 6 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 `Z3BEBY`.
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.

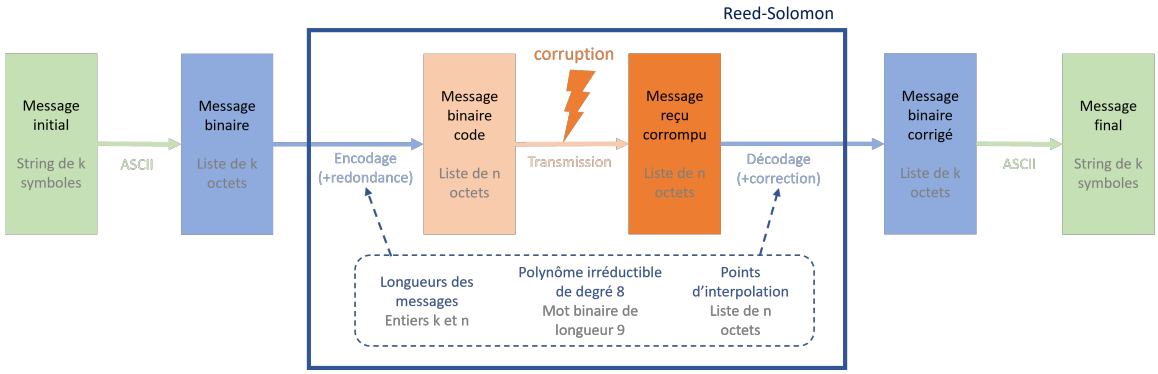

FIGURE 1 – Structure de transmission de message grâce à Reed-Solomon

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 Encodage et décodage

2.1 Encodage

Comme défini au cours, le code est une fonction

$$c : (\mathbb{F}_{2^8})^k \to (\mathbb{F}_{2^8})^n , \tag{1}$$

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

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

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

$$\mathbb{F}_{2^8}[Y] = \left\{ a(Y) = \sum_{i=0}^{d-1} a_i Y^i \;\middle|\; a_i \in \mathbb{F}_{2^8}, d \in \mathbb{N} \right\}. \tag{3}$$

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") :

$$\mathbf{I} = (I_0, I_1, ..., I_{n-1}) = \big(A(y_0), A(y_1), ..., A(y_{n-1})\big) \in (\mathbb{F}_{2^8})^n \tag{4}$$

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 (1), on établit l'égalité $c(\mathbf{a}) = \mathbf{I}$.

2.2 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 :

$$A(y_{i_j}) = a_0 + a_1 y_{i_j} + ... + a_{k-1} {y_{i_j}}^{k-1} \tag{5}$$

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 **??**. Le système linéaire s'écrit alors

$$V\mathbf{a} = \mathbf{I_{restr}} \tag{6}$$

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 $\mathbb{F}_{2^8}$:

$$V = \begin{pmatrix} 1 & y_{i_0} & \cdots & (y_{i_0})^{k-1} \\ 1 & y_{i_1} & \cdots & (y_{i_1})^{k-1} \\ \vdots & & & \vdots \\ 1 & y_{i_{k-1}} & \cdots & (y_{i_{k-1}})^{k-1} \end{pmatrix} \tag{7}$$

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 1, 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 1: Elimination Gaussienne

```
// On souhaite résoudre le système Va = I
// P est la matrice augmentée: P = [V|I]
def gaussian_elimination(P):
    for i in range(k) do
        for j in range(k) do
            if i ≠ j then
                r ← P[j,i]/P[i,i]
                for m in range(k+1) do
                    P[j,m] ← P[j,m] − r × P[i,m]
    for i in range(k) do
        x[i] ← P[i,k]/P[i,i]
    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 3.2).

2.3 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`.

3 Compléments d'informations

3.1 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.

3.2 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.