Cybersecurity MASTER

UNIVERSITÉ Grenoble Alpes

Grenoble INP ensimag

Mater Cybersecurity
Master of Science in Informatics at Grenoble
Master Informatique / Master Mathématiques & Applications

Randomisation des nombres dans l'exponentiation modulaire

Amiot Simon

18 août 2017

Research project performed at *Imaths*

Under the supervision of:
Nicolas Méloni, *Imaths*

Defended before a jury composed of:
[Prof/Dr/Mrs/Mr] <first-name last-name>
[Prof/Dr/Mrs/Mr] <first-name last-name>
[Prof/Dr/Mrs/Mr] <first-name last-name>
[Prof/Dr/Mrs/Mr] <first-name last-name>

[September] 2017

Résumé

Le premier objectif de ce stage est de proposer une implémentation logicielle d'un algorithme décrit dans un article pensé et rédigé par Nicolas Méloni, mon superviseur pour ce stage. L'algorithme a pour but de recoder l'exposant dans l'exponentiation modulaire afin de rendre sa représentation aléatoire et efficace.

Le deuxième objectif de ce stage est de généraliser le concept et l'algorithme de l'article à la double exponentiation.

Résumé

Dans une première partie, une brève introduction explicite l'exponentiation modulaire et son utilisation dans plusieurs protocoles essentiels en cryptographie.

La deuxième partie explique l'état de l'art du recodage.

La troisième partie introduit la double exponentiation ainsi que les différentes méthodes de recodage existantes.

La quatrième partie est constituée de l'analyse du problème et de l'approche de la solution.

La cinquième partie concerne l'implémentation de la solution trouvée dans la partie précédente.

La sixième partie traite de l'étude statistique de l'algorithme et de son efficacité d'un point de vue sécurité et d'un point de vue performance.

Table des matières

— 1 —
Introduction

L'exponentiation modulaire est un algorithme au coeur d'un grand nombre de protocoles en cryptographie, notamment pour le RSA ou ECC.

L'exponentiation modulaire et le RSA

Dans le protocole RSA, l'exponentiation modulaire joue un rôle central. Voici les étapes du chiffrement et du déchiffrement *RSA* :

1) Choisir p et q, deux nombres premiers distincts.

2) Calculer leur produit $n = pq$, appelé *module de chiffrement*.

3) Calculer $\phi(n) = (p-1)(q-1)$.

4) Choisir un entier naturel e premier avec $\phi(n)$ et strictement inférieur à $\phi(n)$, appelé *exposant de chiffrement*.

5) Calculer l'entier naturel d, inverse de e modulo $\phi(n)$, et strictement inférieur à $\phi(n)$, appelé *exposant de déchiffrement*.

Pour toutes ces étapes, l'exponentiation modulaire n'intervient pas.

Cependant pour chiffrer un message M, représenté par un entier naturel strictement inférieur à n, il faut calculer $C \equiv M^e \mod n$.

Le déchiffrement de C nécessite également une exponentiation modulaire, pour retrouver le message clair, il faut calculer $M \equiv C^d \mod n$.

Dans le protocole *RSA*, l'exponentiation modulaire a une place importante (à la fois dans le chiffrement et dans le déchiffrement) ; cependant il y a d'autres protocoles pour lesquels l'exponentiation est requise, comme par exemple le protocole basé sur les courbes elliptiques appelé *ECC*.

Il est toutefois important de signaler que les opérations sont différentes dans le groupe des courbes elliptiques : les multiplications du *RSA* deviennent des additions pour *ECC* et l'exponentiation modulaire correspond à la multiplication scalaire pour le protocole *ECC*.

La différence principale entre le protocole *RSA* et le protocole *ECC* est la suivante :

- L'opposé d'un point P, noté $-P$, est facile à calculer pour *ECC* (négligeable comparé à l'addition de points),

- L'inverse d'un entier naturel m modulo n, noté m^{-1}, est coûteux en temps de calcul pour *RSA* (plus coûteux que la multiplication modulaire).

Les premiers algorithmes d'exponentiation

Pour implémenter l'exponentiation, il existe deux méthodes principales : les algorithmes *Left-To-Right* et *Right-To-Left*. Dans ce stage, nous avons choisi de traiter l'algorithme *Left-To-Right*.

Algorithm 1 Algorithme Left-To-Right

Require: x, N et k avec $k = [k_0, k_1, \ldots, k_n]$ with $k_i \in \{0,1\}$
Ensure: $x^k \mod N$
  $S \leftarrow 1$
  for i from n to 0 **do**
    $S \leftarrow S \times S \mod N$
    if $k_i = 1$ **then**
      $S \leftarrow S \times x \mod N$
    end if
  end for

Cet algorithme est un algorithme de base permettant de comprendre le processus d'exponentiation mais aussi de voir que l'exponentiation dépend essentiellement de la représentation binaire du secret k et en particulier du poids de Hamming (nombre de 1). Ici, la représentation choisie est la décomposition canonique en base 2 de k avec coefficients valant soit 0, soit 1. Cependant, nous allons voir par la suite qu'il existe d'autres représentations ayant des avantages intéressants.

— 2 —

Représentation aléatoire de l'exposant

2.1 State of the art...

Afin d'améliorer l'efficacité de l'exponentiation, il est possible de précalculer certaines valeurs et de changer la représentation de l'exposant en fonction de ces nombres. La première partie explicite l'algorithme d'exponention basé sur le précalcul, puis les parties suivantes donnent 3 méthodes différentes de représentation.

2.1.1 Précalcul et Exponentiation

Soit $\mathscr{G}$ un groupe et $\mathscr{D}$ un ensemble d'entiers positifs impairs. Soit $k = (k_{l-1} \dots k_0)_2$ un entier, où les k_i sont des éléments de $\mathscr{D}$, et soit $g \in \mathscr{G}$. L'algorithme est en deux étapes :

1) Calculer g^d, où d sont les éléments de $\mathscr{D}$

2) Si $k_i \neq 0$, multiplier par l'élément précalculé g^{k_i}

Algorithm 2 Algorithme d'exponentiation avec précalcul

Require: un élément g, un entier N positif, un entier k avec $k = (k_{l-1}, k_1, \dots, k_n)$ with $k_i \in \mathscr{D}$
Ensure: $g^k \mod N$

```
S ← 1
for i from n to 0 do
    S ← S × S mod N
    if k_i ≠ 0 then
        S ← S × g^{k_i} mod N
    end if
end for
```

A la différence de l'algorithme classique, on multiplie par une valeur g^{k_i} qui dépend de la valeur du bit numéro i lorsque celui est non nul (dans l'algorithme classique on a $\mathscr{D} = \{1\}$, et donc le seul élément précalculé est $g^1 = g$, c'est donc l'élément que l'on multiplie à chaque bit non nul).

2.1.2 Représentation *NAF* (Non Adjacent Form)

La représentation *NAF* est l'unique représentation d'un nombre en entiers relatifs pour laquelle le poids de Hamming est minimal.

Exemple 1. *Le nombre* 7 *s'écrit comme :*

$$\begin{array}{llllll} (0 & 1 & 1 & 1)_2 & = & 4+2+1 & =7 \\ (1 & 0 & -1 & 1)_2 & = & 8-2+1 & =7 \\ (1 & -1 & 1 & 1)_2 & = & 8-4+2+1 & =7 \\ (1 & 0 & 0 & -1)_2 & = & 8-1 & =7 \end{array}$$

La représentation NAF de 7 *est la dernière :*

$$(1 \quad 0 \quad 0 \quad -1)_2$$

2.1.3 Représentation *wNAF* (window Non Adjacent Form)

L'idée est de recoder l'exposant avec d'autres coefficients que -1, 0 et 1 en ajoutant tous les nombres impairs inférieurs à 2^w avec w appelée la taille de la fenêtre.

Soit $\mathscr{B} = \{1,3,\ldots,2^w-1\}$, la représentation *wNAF* de k est l'unique représentation de k telle que :

$$k = \quad (b_n \quad \ldots \quad b_1 \quad b_0)_2$$

où $b_i \in \mathscr{B} \cup (-\mathscr{B}) \cup \{0\}$, c'est-à-dire b_i peut être égal à 0, un élément de $\mathscr{B}$ (1, 3 , ...) ou à l'opposé d'un élément de $\mathscr{B}$ (-1, -3, ...).
Et ainsi k vérifie :

$$k = \sum_{i=0}^{n} b_i 2^i$$

L'utilité de cette représentation est de réduire le poids de Hamming de la représentation de k, et de ce fait de réduire le temps de calcul de $x^k \mod N$.

Exemple 2. *Le nombre* 88 *s'écrit comme :*

$$(1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0)_2 \quad = \quad 64+16+8 \quad =88$$

Pour calculer la représentation 3*NAF de* 88*, il faut définir* $\mathscr{B} = \{1,3,5,7\}$*. En regardant les* 3 *premiers bits de poids fort de* 88*, on peut remplacer* $(101)_2$ *par* 5*, puis en regardant les* 3 *suivants , puis à nouveau les suivants, on remarque que l'on ne peut pas remplacer les coefficients, on obtient donc :*

$$(5 \quad 1 \quad 0 \quad 0 \quad 0)_2 \quad = \quad 5\times 16+8 \quad =88$$

2.1.4 Fractional Windows

Cette méthode consiste à adapter la taille de w au fil de l'algorithme. La taille de la fenêtre (3 dans l'exemple précédent) n'est plus fixée, elle est adaptée au fil de l'algorithme afin d'optimiser la vitesse et les calculs de l'algorithme d'exponentiation.

2.2 Randomisation et Optimisation

La partie précédente a présenté les bases et prérequis pour la compréhension de l'article de Nicolas Méloni. La partie suivante s'intéresse à la méthode d'exponentiation modulaire expliquée dans l'article, et à un moyen d'implémenter cette méthode le plus efficacement possible. L'idée de la méthode est de recoder de manière aléatoire l'exposant (secret).

La première section explique la méthode de randomisation telle que détaillée dans l'article. La deuxième présente l'implémentation de celle-ci ainsi que l'optimisation de l'algorithme trouvé, Enfin, la troisième correspond à la démonstration de la validité de cet algorithme.

2.2.1 Randomisation dans l'algorithme de Fractional Windows

L'idée de l'algorithme de recodage de l'exposant k est de changer à chaque éxecution de système de codage, et plus particulièrement de changer l'ensemble des coefficients $\mathscr{D}$.

Soit k l'exposant, N le modulo et x le message. On pose $\mathscr{D} = \{d_1, \ldots, d_l\}$ où d_l sont des entiers positifs impairs.

Le premier constat révélé dans l'article est que pour pouvoir recoder la totalité des exposants, il faut nécessairement que 1 soit dans $\mathscr{D}$.

La fonction *digit*

Soit $w > 0$ un entier. Pour tout entier x on définit $p_w(x) = x \mod 2^w$. Puis on pose $\mathscr{D}_w = p_w(\mathscr{D})$ et $\overline{\mathscr{D}_w} = \mathscr{D}_w \cup \{2^w - d : d \in \mathscr{D}_w\}$. Finalement on définit $W_n = \lfloor log_2(max_i(d_i)) \rfloor$.

Pour construire la fonction *digit*, on doit d'abord définir, pour tout k entiers k, $w_{max}(k)$ comme le plus grand nombre $w \leq W_n + 2$ tel qu'il existe un élément $d_i \in \mathscr{D}$ qui satisfait ces deux conditions :

1. $d_i < k$,
2. $p_w(k) \in \overline{\mathscr{D}_w}$

Finalement, la fonction $digit_{\mathscr{D}} : \mathbb{N} \to \overline{\mathscr{D}} \cup \{0\}$ est définie par :

- si k est pair : $digit(k) = 0$,
- si k est impair :
  - on pose $W_{max} = w_{max}(k)$
  - si $p_{W_{max}}(k) \in \mathscr{D}_{W_{max}}$, $digit_{\mathscr{D}}(k) = d$ avec $d < k$ tel que $p_w(k) = p_w(d)$
  - si $2^{W_{max}} - p_{W_{max}}(k) \in \mathscr{D}_{W_{max}}$, $digit_{\mathscr{D}}(k) = -d$ avec $d < k$ tel que $2^{W_{max}} - p_w(k) = p_w(d)$

La fonction *digit* est bien définie pour tout k dans $\mathbb{N}$. De plus, on peut remarquer que $1 \in \mathscr{D}$, ce qui implique que $1 \in \mathscr{D}_w$ pour tout w, et donc $W_{max} \geq 2$.

Voici un exemple simple pour bien comprendre la fonction *digit* :

Exemple 3. *On pose* $k = 101$ *et* $\mathscr{D} = \{1,3,9\}$*, on a donc* $W_n = 3$.
Pour calculer la fonction digit*, il faut d'abord déterminer* $w_{max}(k)$.
On pose $w = W_n + 2 = 5$*, on calcule* $p_w(k) = 101 \mod 2^5 = 5 \mod 32$.
Puis on calcule $\overline{\mathscr{D}_w} = \{1,3,9,32-1,32-3,32-9\} = \{1,3,9,31,29,23\}$.
On vérifie si $5 \in \overline{\mathscr{D}_w}$*, ce n'est pas le cas, donc on pose* $w = 4$ *et on recommence le processus.*
On a $p_w(k) = 5 \mod 16$.
Puis on calcule $\overline{\mathscr{D}_w} = \{1,3,9,15,13,7\}$.
On a $5 \notin \overline{\mathscr{D}_w}$*, donc on pose* $w = 3$.
On a $p_w(k) = 5 \mod 8$.
Puis on calcule $\overline{\mathscr{D}_w} = \{1,3,9 \mod 8, 8-1, 8-3, 8-9 \mod 8\} = \{1,3,5,7\}$.
On a $5 \in \overline{\mathscr{D}_w}$*, donc on a* $w_{max}(k) = 3$*. On pose* $W_{max} = w_{max}(k)$.
Ici, on est dans le cas où k est impair, et $2^{W_{max}} - p_{W_{max}} \in \mathscr{D}_{W_{max}}$*, donc on a* $digit_{\mathscr{D}}(k) = -3$ *car* $2^{W_{max}} - p_w(k) = 2^3 - 5 = 3$.

L'algorithme de représentation

Une fois la fonction *digit* définie, l'algorithme de la représentation est le suivant :

Algorithm 3 Représentation aléatoire de k

Require: Un entier k et un ensemble $\mathscr{D} = \{1, d_2, \ldots, d_n\}$
Ensure: $k = (k_t k_{t-1} \ldots k_1 k_0)_2$, $k_i \in \overline{\mathscr{D}} \cup \{0\}$
  $i \leftarrow 0$
  while $k! = 0$ **do**
    $k_i \leftarrow digit_{\mathscr{D}}(k)$
    $k \leftarrow \frac{k-k_i}{2}$
    $i \leftarrow i+1$
  end while
  return $(k_{i-1} \ldots k_0)$

Première implémentation

Le coeur de l'implémentation de la représentation réside dans celle de la fonction *digit*. Voici l'algorithme avec lequel j'ai implémenté la fonction *digit* au premier essai :
Il est composé de deux parties :

1) Un algorithme qui calcule i tel que $p_{W_{max}}(k) = p_{W_{max}}(d_i)$ où $d_i \in \overline{\mathscr{D}}$.

2) L'algorithme de la fonction :

1) L'idée est de partir de la fenêtre de longueur maximale $w = W_n + 2$ et de chercher dans $\overline{\mathscr{D}_w}$ si $p_w(k)$ est dedans, s'il n'y est pas on diminue w jusqu'à trouver $p_w(k)$.
On conviendra que, d'après l'algorithme, i est nécessairement compris entre 0 et $2\#D - 1$
2) A partir de cet algorithme, je peux désormais construire l'algorithme qui calcule la fonction *digit*.

Algorithm 4 Calculer i tel que $p_{W_{max}}(k) = p_{W_{max}}(d_i)$

Require: Un entier k, le nombre W_n et un ensemble $\mathscr{D} = \{1, d_2, \ldots, d_n\}$
Ensure: i tel que $p_{W_{max}}(k) = p_{W_{max}}(d_i)$ où $d_i \in \overline{\mathscr{D}}$

```
c = 0
w ← W_n + 2
while w > 1 et c ≠ 1 do
    D̄_w ← p_w(D̄)
    p_k = p_w(k)
    i = 0
    while c ≠ 1 et i < 2 × #D do
        if p_k = D̄_w[i] then
            c ← 1
        end if
        i ← i + 1
    end while
    w ← w − 1
end while
return i − 1
```

Algorithm 5 Calculer $digit_{\mathscr{D}}(k)$

Require: Un entier k, le nombre W_n et un ensemble $\mathscr{D} = \{1, d_2, \ldots, d_n\}$
Ensure: $digit_{\mathscr{D}}(k)$

```
digit ← 0
if k est impair then
    i = algorithm4(k, W_n, D)
    if i < #D then
        digit ← D[i]
    else
        digit ← −D[i − taille]
    end if
end if
return digit
```

2.2.2 Optimisation de l'algorithme

L'idée de l'optimisation est de réduire l'appel à la fonction de recherche d'élément dans un tableau dans *Algorithm 4*. Rechercher dans un tableau ou une table (ou un ensemble) est chronophage en programmation, ainsi pour réduire le temps et le coût d'éxecution de la représentation, nous avons pensé à un autre algorithme. Notre idée est basée sur les trois propositions suivantes :

- Construire un unique tableau d'éléments de $\mathscr{D}$.
- L'index du tableau correspond à l'antécédant et la case à l'image de l'index.
- Le remplissage du tableau nécessite uniquement des additions, des multiplications et des modulos.

Digit vue comme un tableau

Afin d'éviter l'appel à une recherche de table, nous avons abordé la fonction *digit* de manière différente, non pas comme une fonction mais comme la composition d'un modulo et d'une fonction définie par un vecteur. L'algorithme de la fonction *digit* suit ces étapes :

1) Construction d'un tableau D_{max} de taille 2^{W_n+2} où W_n est la taille du plus grand élément de $\mathscr{D}$
2) Remplissage de tableau D_{max} tel que pour $i \in \{0,\dots,2^{W_n+2}-1\}$, $D_{max}[i] = digit(i)$
3) Calcul de $k_{mod} = k \text{ Mod } 2^{W_n+2}$
4) Renvoi de $D_{max}[k_{mod}] = digit(k)$

Construction de D_{max}

Avant de calculer et de compléter le tableau D_{max}, il faut calculer W_n et $\overline{\mathscr{D}} = \mathscr{D} \cup (-\mathscr{D})$.

La construction du tableau D_{max} comporte 3 principales étapes :

- Initialiser toutes les cases à 0 et mettre les éléments de $\overline{D}$ à leur place ($D_{max}[1] = 1$, $D_{max}[23] = 23$).
- Pour i de $W_n + 1$ à 3, on ajoute $2^i \times (2l+1)$ à chaque élément $\overline{D}$ avec $0 \leq l < 2^{W_n+2-i-1}$ et on complète le tableau avec ces valeurs.
- On complète les cases impaires qui valent 0 avec 1 ou -1 suivant la case.

2.2.3 Preuve

Dans cette partie, nous allons montrer que l'algorithme construit à partir de D_{max} calcule la fonction *digit* :

Algorithm 6 Calculer $D_{max} = [1, d_i, \ldots, d_i'] \text{ avec } d_i, d_i' \in \overline{\mathscr{D}}$

Require: W_n et $\mathscr{D}$
Ensure: $D_{max} = [1, d_i, \ldots, d_i'] \text{ avec } d_i, d_i' \in \overline{\mathscr{D}}$

```
sizeD ← |D|
for k de 0 à sizeD − 1 do
    D̄[k] ← D[k]
    D̄[sizeD + k] ← 2^(W_n+2) − D[k]
end for
for k de 0 à 2^(W_n+2) − 1 do
    D_max[k] ← 0
end for
for k de 0 à sizeD − 1 do
    D_max[D̄[k]] ← D[k]
    D_max[D̄[taille + k]] ← −D[k]
end for
while i ≥ 3 do
    j ← 2 × sizeD
    borne ← 2^(W_n+2−i−1)
    while j ≥ 0 do
        for k de 0 à borne − 1 do
            d ← D̄[j] + 2^i × (2k + 1) Mod(2^(W_n+2))
            if D_max[d] = 0 then
                if j < sizeD then
                    D_max[d] ← D̄[j]
                else {j ≥ sizeD}
                    D_max[d] ← −D̄[j − taille]
                end if
            end if
        end for
        j ← j − 1
    end while
    i ← i − 1
end while
```

La fonction *digit* sous forme mathématique

On peut remarquer que calculer la fonction *digit* revient à trouver une fonction vérifiant cette propriété :

Propriété 1. *Soit* $\phi_{1_{\mathscr{D}}} : \mathbb{N} \to \mathbb{Z}/2^{W_n+2}\mathbb{Z}$*, la projection canonique restreinte à* $\mathbb{N}$*,*
et $\phi_{2_{\mathscr{D}}} : \mathbb{Z}/2^{W_n+2}\mathbb{Z} \to \overline{\mathscr{D}} \cup \{0\}$ *une fonction.*
On a $digit_{\mathscr{D}} = \phi_{2_{\mathscr{D}}} \circ \phi_{1_{\mathscr{D}}} \iff \forall k, \bar{k} = \phi_1(k) = k \mod 2^{W_n+2}$ *et :*

(i) Si k est pair $\phi_{2_{\mathscr{D}}}(\bar{k}) = 0$

(ii) Si k est impair $\phi_{2_{\mathscr{D}}}(\bar{k}) = d_i$*, avec* $d_i \in \overline{\mathscr{D}} \Rightarrow \exists j \in [\![2, W_n+2]\!]$*, tel que* $k = d_i \mod 2^j$ *et* $\forall m \in [\![j+1, W_n+2]\!], \forall d_i' \in \overline{\mathscr{D}}, k \neq d_i'$ *Mod* 2^m.

Remarque 1. *La condition (ii) explique que le j que l'on trouve doit être le plus grand possible.*

Preuve 1. *On prouve la double implication pour tout k dans* $\mathbb{N}$.
D'abord si k est pair, c'est évident car les deux fonctions valent 0 *et* $\phi_{1_{\mathscr{D}}}(k)$ *est "paire" pour k pair et ne peut valoir* 0 *si k est impair.*
Maintenant si k est impair. Commençons avec l'implication $\Leftarrow$ *:*

$\Leftarrow$ *Soit* $\phi_{2_{\mathscr{D}}}$ *verifiant les conditions (i) and (ii).*
Alors $\exists j \in [\![2, W_n+2]\!]$*, tel que* $k = d_i \mod 2^j$ *et* $\forall m \in [\![j+1, W_n+2]\!]$,
$\forall d_i' \in \overline{\mathscr{D}}, \bar{k} \neq d_i' \mod 2^m$.
D'où $\forall m > j$, $p_m(k) \notin \overline{\mathscr{D}_m}$.
Ce qui implique $w_{max} \leq j$.
De plus $\bar{k} = d_i \mod 2^j$ *avec* d_i *élément de* $\overline{\mathscr{D}}$*, d'où* $p_j(k) \in \overline{\mathscr{D}}_j$*. Finalement on a* $w_{max} = j$.

- *Si* $d_i \in \mathscr{D}$, $p_j(k) = p_j(d_i) \Rightarrow digit_{\mathscr{D}}(k) = \phi_{2_{\mathscr{D}}}(\bar{k})$.
- *Si* $-d_i \in \mathscr{D}$, $p_j(k) = p_j(d_i) \Rightarrow 2^j - p_j(k) = p_j(-d_i) \Rightarrow 2^j - p_j(k) \in \mathscr{D}$.
  Donc $digit_{\mathscr{D}}(k) = -(-d_i) = d_i = \phi_{2_{\mathscr{D}}}(\bar{k})$.

$\Rightarrow$ *On suppose que l'on a* $digit_{\mathscr{D}}(k) = \phi_{2_{\mathscr{D}}}(\bar{k}) = d_i$ *ou* $-d_i$ *avec* $d_i \in \mathscr{D}$.
On a $digit_{\mathscr{D}}(k) = d_i$ *avec* $d_i \in \overline{\mathscr{D}_m}$.
Soit $j = w_{max}(k) = W_max$.
Par definition de $w_{max}(k)$*, la condition* 2) *est fausse* $\forall m > j$,
donc $\nexists d_i' \in \mathscr{D}$ *tel que* $p_m(k) \in \overline{\mathscr{D}_m}$.
Ainsi $\forall m > j$, $\forall d_i' \in \overline{\mathscr{D}}$, $k \neq d_i' \mod 2^m$.

De plus $j = w_{max} \Rightarrow w_{max}$ *satisfait la condition* 2),
i.e. $p_j(k) \in \overline{\mathscr{D}_j} \Rightarrow \exists d_i \in \overline{\mathscr{D}}$ *tel que* $k = d_i \mod 2^j$.

Preuve de fonctionnement de D_{max}

Proposition 1. *Soit* $\phi_{1_{\mathscr{D}}} : \mathbb{N} \to \mathbb{Z}/2^{W_n+2}\mathbb{Z}$ *la projection canonique restreint à* $\mathbb{N}$ *et* $\phi_{2_{\mathscr{D}}} : \mathbb{Z}/2^{W_n+2}\mathbb{Z} \to \overline{\mathscr{D}} \cup \{0\}$ *tel que* $\phi_{2_{\mathscr{D}}}(\bar{k}) = D_{max}[\bar{k}]$*. Donc on a* $digit_{\mathscr{D}} = \phi_{2_{\mathscr{D}}} \circ \phi_{1_{\mathscr{D}}}$.

Preuve 2. *Par construction* $\phi_{2_{\mathscr{D}}} : \mathbb{Z}/2^{W_n+2}\mathbb{Z} \to \overline{\mathscr{D}} \cup \{0\}$ *donc* $\phi_{2_{\mathscr{D}}}$ *existe. Par la propriété* 1*, on doit prouver que* $\phi_{2_{\mathscr{D}}}$ *vérifie* $\forall k$ *:*

(i) Si k est pair $\phi_{2_{\mathscr{D}}}(\bar{k}) = 0$

(ii) Si k est impaire $\phi_{2_{\mathscr{D}}}(\bar{k}) = d_i$, *avec* $d_i \in \overline{\mathscr{D}} \Rightarrow \exists j \in [\![2, W_n+2]\!]$, *tel que* $k = d_i \mod 2^j$ *et* $\forall m \in [\![j+1, W_n+2]\!], \forall d_i' \in \overline{\mathscr{D}}, k \neq d_i' \mod 2^m$.

Maintenant regardons la preuve :

(i) La première condition est vérifiée parce que $D_{max}[l] = 0$ *pour* $0 \leq l < 2^{W_n+2}$ *(initialisation) et le tableau est complété en ajoutant des éléments impairs (les impairs de* $\mathscr{D}$ *et* $d+2p$, *d dans* $\mathscr{D}$*).*

(ii) La deuxième condition est plus difficile à prouver.

Par construction $\phi_{2_{\mathscr{D}}}(\bar{k}) = d_i$ *avec* $d_i \in \overline{\mathscr{D}}$ *signifie* $\exists j \in [\![2, W_n+2]\!]$ *s.t.* $\bar{k} = d_i + (2l+1)2^j \mod 2^{W_{max}}$ *avec* $0 \leq l < 2^{W_n+2-j-1}$, *(1)*
et $\forall m > j$, $\nexists d_i', l'$ *tel que* $\bar{k} = d_i' + (2l'+1)2^m \mod 2^{W_n+2}$. *(1bis)*

La relation (1) implique $\exists j$ *tel que* $k = \bar{k} = d_i \mod 2^j$.
Maintenant il faut montrer que $\forall m \in [\![j+1, W_n+2]\!], \forall d_i' \in \overline{\mathscr{D}}, k \neq d_i' \mod 2^m$.
Par l'absurde on suppose qu'il existe un tel $m > j$ *et* $d_i' \in \overline{\mathscr{D}}$ *tel que :*

$$k = d_i' \mod 2^m. \text{ (2)}$$

Avec (2), on a $k = d_i' + n \times 2^m \mod 2^{W_n+2}$ *avec* $0 \leq n < 2^{W_n+2-m}$. *(3)*
Par distinction de cas, on a :

- *Si* $n = 0$, *(3)* $\Rightarrow k = d_i' \mod 2^{W_n+2}$, *contradiction avec la construction de* D_{max} *(on complète d'abord le tableau avec les éléments de* $\overline{\mathscr{D}}$*).*

- *Si n est pair, i.e.* $n = 2n'$, *soit* $m' = m+1$ *et*
  (3) $\Rightarrow k = d_i' + n' \times 2^{m'} \mod 2^{W_n+2}$. *Si* n' *est encore pair, on réitère le procédé jusqu'à ce que* n' *soit pair. On réduit le problème au cas n impair.*

- *Si n est impair, i.e.* $n = 2l'+1$.
  Alors (3) $\Rightarrow k = d_i' + (2l'+1) \times 2^m \mod 2^{W_n+2}$ *avec* $0 \leq 2l'+1 < 2^{W_n+2-m}$.
  Et $0 \leq 2l'+1 < 2^{W_n+2-m} \Rightarrow 0 \leq l' < 2^{W_n+2-m-1}$.
  Finalement on a $k = d_i' + (2l'+1) \times 2^m \mod 2^{W_n+2}$ *avec* $0 \leq l' < 2^{W_n+2-m-1}$. *Il y a contradiction avec (1bis). Impossible.*

Donc $\phi_{2_{\mathscr{D}}}$ *vérifie la seconde condition and d'après la propriété, on a* $digit_{\mathscr{D}} = \phi_{2_{\mathscr{D}}} \circ \phi_{1_{\mathscr{D}}}$.

Corollaire 1. *On peut utiliser l'algorithme de construction de* D_{max} *pour calculer la fonction* digit.

2.2.4 Test et Performance

Nous avons effectué des tests de performance afin de déterminer l'influence de cette méthode de randomisation de la représentation sur la vitesse d'exponentiation.

TABLE 2.1 : Temps d'execution (10000 tests)

#D	|k|	*RDR*	*wNAF*	perte (en %)
4	256	373	244	54
4	4096	11950	10820	10
8	256	342	224	52
16	4096	12101	10585	14
32	256	526	214	146
64	256	1108	220	504
64	4096	12481	10538	18

Comparaison avec *wNAF*

On remarque que pour plus la taille de k augmente, plus la perte relative diminue. Cependant pour une taille plus petite de k (256 bits), la perte est significative ($> 50\%$), et plus particulièrement lorsqu'on augmente la taille de $\mathscr{D}$.

— 3 —
Etat de l'art de la double exponentiation

La partie précédente a permis de comprendre la *RDR* (Random Digit Representation) proposée par Nicolas Méloni. Cette partie s'intéresse à la généralisation de la *RDR* pour la multi-exponentiation et plus particulièrement pour la double exponentiation, c'est-à-dire au calcul de $g^a \times h^b$, où g et h sont des éléments de $\mathscr{G}$ (un groupe), et a et b des entiers naturels.

Le procédé de multi-exponentiation est très utilisé dans les protocoles de signatures, que ce soit à base de courbes elliptiques, ou dans d'autres cas comme dans la signature El-Gamal.

Pour se faire, il est important de rappeler l'état de l'art de la double exponentiation.
La première partie explique l'approche de Shamir dans la double exponentiation.
La seconde partie donne la méthode qui a servi de base pour construire la *DoubleRDR* (la *RDR* étendue à la double exponentiation).

3.1 L'approche de Shamir

Dans un premier temps, je donne la méthode naïve de la double exponentiation afin de constater comment Shamir a réussi à l'optimiser dans les parties suivantes.

3.1.1 La méthode naïve

La méthode la plus simple pour calculer $g^a \times h^b$ est de calculer g^a et h^b séparement, puis de les multiplier entre eux.

Appliquer ce processus revient à effectuer deux exponentiations et une multiplication, ce qui fait en moyenne $2l$ "square" et l multiplications, où l est la taille de a et de b (approximativement).

3.1.2 Le "square" en simultané

Le premier constat de Shamir fut qu'il était possible de faire un "square" pour les deux exposants à la fois.

Pour illustrer cette méthode, je prend l'exemple suivant :
Soit $a = 37$ et $b = 20$, ie on cherche à calculer $g^{37}h^{20}$.

$37 =$	$(1$	0	0	1	0	$1)$
$20 =$	$(0$	1	0	1	0	$0)$
$\square$	1	g^2	g^4h^2	g^8h^4	$g^{18}h^{10}$	$g^{36}h^{20}$
$\times g$	g			g^9h^4		$g^{37}h^{20}$
$\times h$		g^2h		g^9h^5		

Le symbole $\square$ correspond à l'opération "square" qu'on effectue à chaque bit.
Les autres symboles $\times g$ et $\times h$ correspondent tout simplement à la multiplication par g et h respectivement.

On s'aperçoit donc que chaque coefficient non nul dans la représentation de g et dans celle de h, donne lieu à une multiplication, cependant cette technique permet de diviser par 2 le nombre de "square" (seulement 5 au lieu de 10 dans notre cas).

Ainsi cette technique donne en moyenne l "square" et l multiplications et est appelée *Simple Shamir Method*.

3.1.3 Le précalcul

Le deuxième constat de Shamir concerna les mutliplications, et particulièrement la quatrième colonne du tableau précédent. Il s'aperçut qu'il était possible d'économiser une multiplication à chaque fois que le coefficient de chaque exposant était non nul simultanément (comme au bit 4 pour 37 et 20).

Le moyen simple d'éviter ces deux multiplications est de précalculer le produit $g \times h$, et ainsi de multiplier directement par $g \times h$ pour économiser une opération à chaque apparition de 1 sur la même colonne.

Avec le même exemple que dans la partie précédente, on a :

$37 =$	$(1$	0	0	1	0	$1)$
$20 =$	$(0$	1	0	1	0	$0)$
$\square$	1	g^2	g^4h^2	g^8h^4	$g^{18}h^{10}$	$g^{36}h^{20}$
$\times g$	g					$g^{37}h^{20}$
$\times h$		g^2h				
$\times gh$				g^9h^5		

Ainsi cette technique donne en moyenne l "square" et $\frac{3l}{4}$ multiplications et est appelée *Fast Shamir Method*.

3.2 L'approche de Solinas et la *JSF*

L'approche de Shamir a l'avantage de fonctionner pour tous les groupes, et en particulier dans le cas des courbes elliptiques, ainsi il est possible d'utiliser les astuces des parties précédentes dans ce cas.

C'est avec cette idée que Solinas a pensé une méthode pour recoder les deux exposants dans la double exponentiation afin d'en accélérer l'execution.

3.2.1 Basée sur un *NAF*

Dans le cas des courbes elliptiques, où le calcul de $-P$ est négligeable comparé à l'addition de points (analogie entre multiplication scalaire pour les courbes elliptiques et exponentiation modulaire), on peut utiliser une représentation signée ("signed binary expansion"), c'est-à-dire utiliser 0, 1 ET -1.

Cependant, étant donné qu'il existe une infinité de représentation signée, on utilise la représentation optimale (au vu du nombre de 0) de l'exposant, appelée aussi représentation *NAF* (vue dans la première partie) qui vérifie certaines propriétés.

Solinas est allé encore plus loin en constatant qu'il était encore plus optimal de choisir entre la représentation classique et la représentation *NAF* pour chacun des exposants afin que les coefficients non nuls de chaque recodage se situent sur les mêmes bits (à la même colonne dans la représentation).

Par exemple, avec les exposants 11 et 20, le calcul de $11P+20Q$ donne :

$11=$	$(1$	0	-1	0	$-1)$
$20=$	$(1$	0	1	0	$0)$
$\times 2$	$\mathscr{O}$	$2P+2Q$	$4P+4Q$	$6P+10Q$	$12P+20Q$
$\pm P$	P		$3P+4Q$		$11P+20Q$
$\pm Q$	$P+Q$		$3P+5Q$		

Pour le premier exposant, j'ai opté pour la représentation *NAF* alors que pour le deuxième exposant, j'ai choisi la représentation classique. Le résultat est que le nombre de zéros sur la même colonne permet d'économiser des multiplications.

Cette représentation est appelée "Joint Signed Binary Expansion", elle est adaptée à la "Simple Shamir Method" dans l'exemple. Il existe également une adaptation de ce raisonnement avec la "Fast Shamir Method" mais elle n'est pas triviale.

Quoiqu'il en soit, la "Joint Signed Binary Expansion" nécessite en moyenne l doublages (équivalent du "square") et $\frac{2l}{3}$ additions avec l'adaption simple, et l doublages et $\frac{5l}{9}$ additions avec l'adaption rapide de Shamir.

C'est avec cette approche que Solinas a trouvé une autre méthode de recodage.

3.2.2 La Joint Spar Form

Dans cette partie, j'expliquerai la méthode de Solinas pour recoder les deux exposants dans la double exponentiation.

Dans un premier temps, j'explicite la notion de poids "joint" de Hamming, puis dans un exemple je montre l'avantage de la *JSF* sur la représentation *NAF*. Ensuite je donne un exemple concret dont je me servirai dans les prochaines parties, puis finalement je donne les résultats qu'a obtenu Solinas avec cette méthode.

Joint Hamming Weight

Le Poids de Hamming Joint est le nombre de colonnes non nulles dans la représentation de deux nombres. Une colonne non nulle est un bit pour lequel l'un des deux (ou les deux !) exposant(s) a un coefficient non nul. Par exemple, dans le cas de la partie précédente, on a :

$$\begin{array}{lccccc} 11= & (1 & 0 & -1 & 0 & -1) \\ 20= & (1 & 0 & 1 & 0 & 0) \end{array}$$

Le poids de Hamming Joint de cette représentation est 3 car il y a 3 colonnes non nulles (et 2 colonnes $\binom{0}{0}$).

Si on applique la Fast Shamir Method à cette représentation, le nombre d'addition est exactement le poids de Hamming Joint de la représentation.

Comparaison avec *NAF*

Avant de comprendre la construction de la représentation *JSF*, je montre dans un exemple comparatif l'efficacité de cette dernière.

Alors que *NAF* a pour but de minimiser le poids de Hamming de chacun des exposants, la *JSF* a pour but de minimiser le Poids de Hamming Joint.

Dans l'exemple ci-dessous, on remarque qu'avec la représentation *NAF* :

$$\begin{array}{rl} 53 = & (0 \quad 1 \quad 0 \quad -1 \quad 0 \quad 1 \quad 0 \quad 1) \\ 102 = & (1 \quad 0 \quad -1 \quad 0 \quad 1 \quad 0 \quad -1 \quad 0) \end{array}$$

Le poids de Hamming Joint est 8 (il n'y a aucun $\binom{0}{0}$).

Les mêmes exposants avec la représentation *JSF* :

$$\begin{array}{rl} 53 = & (0 \quad 1 \quad 0 \quad 0 \quad -1 \quad 0 \quad -1 \quad -1) \\ 102 = & (1 \quad 0 \quad 0 \quad -1 \quad -1 \quad 0 \quad -1 \quad 0) \end{array}$$

Le poids de Hamming Joint est seulement 6 (il y a 2 $\binom{0}{0}$).

La *JSF* dans ce cas permet d'économiser deux additions, cependant il faut précalculer $P+Q$ et $P-Q$.

Un exemple simple

Voici un exemple simple de la Fast Shamir Method appliquée à la *JSF*, avec 53 et 102 comme exposants :

$53 =$	$(0$	1	0	0	-1	0	-1	$-1)$
$102 =$	$(1$	0	0	-1	-1	0	-1	$0)$
$\times 2$	$\mathcal{O}$	$2Q$	$2P+4Q$	$4P+8Q$	$8P+14Q$	$14P+26Q$	$28P+52Q$	$54P+102Q$
$\pm P$		$P+2Q$						$53P+102Q$
$\pm Q$	Q			$4P+7Q$				
$\pm(P+Q)$					$7P+13Q$		$27P+51Q$	
$\pm(P-Q)$								

Cette méthode nécessite en moyenne l doublages et $\frac{l}{2}$ additions.

— 4 —
Approche du problème

La partie précédente a rappelé l'état de l'art de la double exponentiation, cette partie s'intéresse à la problèmatique suivante : peut-on étendre la *RDR* de Nicolas Méloni à la double exponentiation ?

L'objectif principal de la *RDR* est la protection de l'algorithme d'exponentiation contre les attaques par canal auxiliaire. Ainsi, la *DoubleRDR* (*RDR* étendue à la double exponentiation) doit pouvoir utiliser des coefficients quelconques (impairs). Il faut également que les résultats ne dépendent pas des coefficients choisis sous peine de voir la méthode vulnérable à une attaque "side channel".

L'objectif secondaire de la *DoubleRDR* est de rendre l'exponentiation modulaire la plus efficace possible.

Dans une première section, on définit la *DoubleRDR* et les contraintes associées.

La deuxième section détaille la méthode générale de la *DoubleRDR*.

Puis la dernière section donne un exemple de *DoubleRDR* de deux exposants afin de mieux comprendre la méthode.

4.1 La représentation d'un couple d'exposants

Dans cette partie, on fixe un couple d'exposants $(k_1, k_2) = (53, 102)$ et un ensemble $\mathscr{D} = \{1, 3, 23, 27\}$, puis on détermine la représentation idéale, au sens des objectifs fixés, de ce couple.

Solinas a adapté la méthode *NAF* à la double exponentiation, ainsi il semble possible d'adapter la *RDR* à la double exponentiation, en utilisant le même procédé.

A l'aide d'un exemple simple, cette partie définit la *DoubleRDR* au sens des objectifs fixés et des avantages de cette représentation.

4.1.1 Une solution simple à l'objectif 1

Le premier objectif de la *DoubleRDR* est le recodage des deux exposants à l'aide de coefficients aléatoires (de cette façon, la *DoubleRDR* rend l'algorithme d'exponentiation résistant à une attaque "side-channel").

Une première approche triviale consiste à utiliser la *RDR* sur chacun des exposants, de cette manière la représentation obtenue utilise les coefficients de $\mathscr{D}$ et remplit l'objectif 1.

En suivant la *RDR*, on obtient :

$$\begin{array}{rcl} 53 & = & (1 \quad 0 \quad 0 \quad -3 \quad 0 \quad 0 \quad 0 \quad -27) \\ 102 & = & (0 \quad 0 \quad 3 \quad 0 \quad 0 \quad 0 \quad 3 \quad 0) \end{array}$$

Le poids de Hamming Joint de cette représentation est 5 car il y a trois $\binom{0}{0}$ sur huit bits.

Il s'agit de déterminer si cette représentation est la meilleure représentation possible au vu de l'objectif 2. Pour se faire, d'après l'approche de Shamir (Fast Shamir Method), la représentation la plus efficace de (k_1, k_2) est celle ayant un poids de Hamming Joint minimal, c'est-à-dire un nombre de $\binom{0}{0}$ maximal.

4.1.2 La *JSF*, solution à l'objectif 2

Afin de déterminer la représentation *DoubleRDR*, une seconde approche consiste à utiliser la méthode *JSF*, méthode la plus efficace pour l'objectif 2, puis à l'adapter avec des coefficients de $\mathscr{D}$.

Par exemple, la *JSF* de (k_1, k_2) est :

$$\begin{array}{rcl} 53 & = & (0 \quad 1 \quad 0 \quad 0 \quad -1 \quad 0 \quad -1 \quad -1) \\ 102 & = & (1 \quad 0 \quad 0 \quad -1 \quad -1 \quad 0 \quad -1 \quad 0) \end{array}$$

Le poids de Hamming Joint de la représentation est 6, cependant l'ensemble $\mathscr{D}$ est réduit à $\{1\}$. Il est évident que cette représentation présente des avantages en terme d'efficacité, et donc des avantages au vu de l'objectif 2, cependant elle ne remplit pas la condition de l'objectif 1 car elle n'utilise que les coefficients 0, 1 et -1.

Ce qui est important dans la méthode *JSF* est de faire correspondre les coefficients non nuls afin de créer des doubles zéros, comme on peut le voir dans l'exemple ci-dessous :

Représentation *NAF* de 53 et de 102 :

$$\begin{array}{rl} 53 = & (0 \quad 1 \quad 0 \quad -1 \quad 0 \quad 1 \quad 0 \quad 1) \\ 102 = & (1 \quad 0 \quad -1 \quad 0 \quad 1 \quad 0 \quad -1 \quad 0) \end{array}$$

Représentation *JSF* de 53 et de 102 :

$$\begin{array}{rl} 53 = & (0 \quad 1 \quad 0 \quad 0 \quad -1 \quad 0 \quad -1 \quad -1) \\ 102 = & (1 \quad 0 \quad 0 \quad -1 \quad -1 \quad 0 \quad -1 \quad 0) \end{array}$$

Les coefficients non nuls de la première représentation sont tous décalés (il n'y a aucun 0 sur la même colonne, et donc sur le même bit), alors que la deuxième représentation offre l'avantage d'avoir deux 0 sur le même bit (c'est-à-dire deux $\binom{0}{0}$) et donc un poids de Hamming de 6 au lieu de 8.

4.1.3 La représentation *DoubleRDR*

Voici deux autres représentations possible de $(53, 102)$ basées sur les deux premiers recodages *RDR* et *JSF* :

$$\begin{array}{rcl} 53 & = & (1 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad -23 \quad 3) \\ 102 & = & (1 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 3 \quad 0) \end{array}$$

Le poids de Hamming Joint de cette représentation est de 4.
Et :

$$\begin{array}{rcl} 53 & = & (0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad -1 \quad 23) \\ 102 & = & (0 \quad 0 \quad 3 \quad 0 \quad 0 \quad 0 \quad 3 \quad 0) \end{array}$$

Le poids de Hamming Joint de cette représentation est de 3.

Voici les premiers constats que l'on peut faire en observant ces résultats :

- La *RDR* de chacun des exposants k_1 et k_2 n'est pas la meilleure représentation de (k_1, k_2) au sens des objectifs donnés.
- Une représentation où les coefficients non nuls de chacun des exposants sont à la même place (1 et 3, puis -1 et 3 sur la même colonne) est plus efficace.

De ces constats, on peut définir la *DoubleRDR* selon les deux objectifs suivants :

Objectif 1 : On choisit d_0 et d'_0 les coefficients dans la *DoubleRDR* de k_1 et k_2 respectivement, tels que le prochain coefficient *non nul* de chaque exposant "coïncident", c'est-à-dire tels qu'il y ait le même nombre de 0 consécutifs après d_0 et d'_0 dans la représentation.

Objectif 2 : La représentation doit contenir un maximum de $\binom{0}{0}$ consécutifs.

Pour résumer, lorsqu'on choisit un coefficient pour k_1 et/ou pour k_2, il faut d'abord que les prochains coefficients *non nuls* de chaque exposant soient sur le même bit, puis si c'est possible qu'il y ait le maximum de 0 consécutifs dans la représentation.

Remarque 2. *Comme* $1 \in \mathscr{D}$*, si* k_1 *et* k_2 *impairs, il est toujours possible de choisir* d_0 *et* d'_0 *tels que* k_1 *et* k_2 *"coïncident".*

Par exemple, si $k_1 = 1 \mod 4$ *et* $k_2 = 3 \mod 4$*, alors il suffit de choisir* $d_0 = -1$ *et* $d'_0 = 1$*. Dans ce cas, on aura* $k_1 - d_0 = k_2 - d'_0 = 2 \mod 4$*, et donc* $\frac{k_1 - d_0}{2}$ *et* $\frac{k_2 - d'_0}{2}$ *tous deux impairs.*

4.2 La méthode *DoubleRDR*

Dans la partie précédente, la représentation *DoubleRDR* d'un couple d'exposant a été définie. Dans cette partie, une méthode pour calculer la *DoubleRDR* de k_1 et k_2 est détaillée.

4.2.1 Le début de la *DoubleRDR*

Avant de détailler le recodage de la *DoubleRDR*, il est nécessaire de rappeler certaines conditions :

- Pour recoder un exposant, il faut utiliser les propriétés arithmétiques du nombre, ou éventuellement sa représentation binaire (on ne peut pas utiliser la représentation *wNAF*, *RDR* ou encore *JSF*).

- Il faut fixer une taille de fenêtre maximale en fonction des éléments de $\mathscr{D}$. Etant donné que l'ensemble $\mathscr{D}$ est borné, le choix d'un coefficient dépend uniquement des premiers bits suivants (équivalent de $W_n + 2$ dans l'algorithme de *RDR*)

- Il est plus pratique et plus efficace de commencer la méthode de recodage par le bit de poids faible plutôt que par le bit de poids fort.

Pour la suite, les bits de poids faibles sont à gauche (et non à droite comme précédemment). Ainsi, le recodage s'effectue de la gauche (bit de poids faible) vers la droite (bit de poids fort).

L'exemple de 53 et 102 donne :

$$\begin{array}{rcl} 53 & = & (1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0) \\ 102 & = & (0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1) \end{array}$$

Il est nécessaire de fixer une taille de fenêtre en fonction des coefficients de $\mathscr{D}$, on calcule (comme dans la *RDR*) $W_n = \lfloor log_2(max_i(d_i)) \rfloor$, et on définit $W_{max} = W_n + 2$.

On peut définir également $\overline{\mathscr{D}} = \mathscr{D} \cup -\mathscr{D}$, qu'on utilise dans la *DoubleRDR*.

Ainsi pour $\mathscr{D} = \{1, 3, 23, 27\}$, on a $\overline{\mathscr{D}} = \{1, 3, 23, 27, -1, -3, -23, -27\}$, et $W_{max} = \lfloor log_2(27) \rfloor + 2 = 6$.

4.2.2 Le choix des coefficients

Cette partie explique comment choisir les coefficients de la *DoubleRDR*.

Bits de poids faibles

Tout d'abord, il faut déterminer le bit de poids faible $B_{f,1}$ et $B_{f,2}$ des exposants k_1 et k_2 respectivement.

Pour $k_1 = 53$ et $k_2 = 102$, on a $B_{f,1} = 0$ et $B_{f,2} = 1$.

Ainsi, les $B_{f,1}$ premiers coefficients de la *DoubleRDR* de k_1 sont nuls, de même pour k_2.

Dans notre exemple on a donc :

$$\begin{array}{rcl} 53 & = & (? \quad \ldots \quad \ldots) \\ 102 & = & (0 \quad ? \quad \ldots) \end{array}$$

Et pour le cas général, en supposant $B_{f,1} > B_{f,2}$, on a :

$$\begin{array}{rcl} k_1 & = & (\underbrace{0 \ldots 0}_{0 \ldots B_{f,2}-1} \quad 0 \quad \underbrace{0 \ldots 0}_{B_{f,1}-B_{f,2}-1} \quad ? \quad \ldots) \\ k_2 & = & (\underbrace{0 \ldots 0}_{0 \ldots B_{f,2}-1} \quad ? \quad \underbrace{0 \ldots 0}_{B_{f,1}-B_{f,2}-1} \quad \ldots \quad \ldots) \end{array}$$

Il faut ensuite diviser par $2^{B_{f,i}}$ l'exposant k_i (afin d'avancer dans la représentation).

Disjonction de cas

Ensuite, il faut distinguer deux cas principaux :

1$^{\text{er}}$ cas : Si $B_{f,1} = B_{f,2}$, ce cas est détaillé dans la prochaine partie.

2$^{\text{ème}}$ cas : Si $B_{f,1} \neq B_{f,2}$, alors on calcule $\omega = |B_{f,1} - B_{f,2}|$.

Dans cette partie, on explique le deuxième cas.

Prenons par exemple le cas où $B_{f,1} > B_{f,2}$ (dans le cas contraire, il suffit d'intervertir les rôles de k_1 et k_2), il s'agit de déterminer un coefficient d_0 pour k_2 tel que le prochain coefficient non nul de k_2 soit sur le bit $B_{f,1}$.

Ce qui donnera ceci dans la représentation :

$$\begin{array}{rcl} k_1 & = & (\underbrace{0\ldots0}_{0\ldots B_{f,2}-1} \quad 0 \quad \underbrace{0\ldots0}_{B_{f,1}-B_{f,2}-1} \quad ? \quad \ldots) \\ k_2 & = & (\underbrace{0\ldots0}_{0\ldots B_{f,2}-1} \quad d_0 \quad \underbrace{0\ldots0}_{B_{f,1}-B_{f,2}-1} \quad ? \quad \ldots) \end{array}$$

Afin de choisir le coefficient d_0 de k_2, il faut trouver $d_0 \in \overline{\mathscr{D}}$ tel que :

$$\begin{array}{c} 2^{\omega} \mid (k_2 - d_0) \mod (2^{W_{max}}) \\ 2^{\omega+1} \nmid (k_2 - d_0) \mod (2^{W_{max}}) \end{array}$$

Il peut arriver qu'aucun élément de $\overline{\mathscr{D}}$ vérifie ces conditions, dans ce cas on pose $\omega' = \omega - 1$, il faut trouver $d_0 \in \overline{\mathscr{D}}$ tel que :

$$\begin{array}{c} 2^{\omega'} \mid (k_2 - d_0) \mod (2^{W_{max}}) \\ 2^{\omega'+1} \nmid (k_2 - d_0) \mod (2^{W_{max}}) \end{array}$$

On répète le processus jusqu'à trouver une solution vérifiant les conditions.

Puis on calcule $\frac{k_2 - d_0}{2^{\omega}}$, afin d'avancer dans la représentation.

Exemple 4. *Pour $k_1 = 53$ et $k_2 = 102$, on a $0 = B_{f,1} < B_{f,2} = 1$, donc $\omega = 1 - 0 = 1$.*

Trouvons un coefficient $d_0 \in dbarre$ tel que :

$$\begin{array}{c} 2 \mid (53 - d_0) \mod 2^6 \\ 4 \nmid (53 - d_0) \mod 2^6. \end{array}$$

Les valeurs 3, 23, 27 et -1 sont solutions, on pose $d_0 = 3$ (choix arbitraire).

On obtient ainsi :

$$\begin{array}{rcl} 53 & = & (3 \quad ? \quad \ldots) \\ 102 & = & (0 \quad ? \quad \ldots) \end{array}$$

Puis on calcule $\frac{53-3}{2} = 25$.

Synchronisation des coefficients

Cette partie a pour but d'expliquer comment choisir les coefficients d_0 et d'_0 de k_1 et k_2 respectivement lorsqu'on est dans le premier cas (voir partie précédente).

Dans le cas où $B_{f,1} = B_{f,2}$, on choisit les coefficients d_0 et d'_0 de k_1 et k_2 simultanément.

Il faut choisir d_0 et d'_0 tels qu'il y ait un *même* nombre *maximum*, noté $\omega_{max} - 1$, de zéros consécutifs dans les représentations de k_1 et k_2, ce qui donnera ceci dans la représentation *DoubleRDR* :

$$\begin{array}{rcl} k_1 & = & (\underbrace{0\ldots0}_{0\ldots B_{f,1}-1} \quad d_0 \quad \underbrace{0\ldots0}_{\omega_{max}-1} \quad ? \quad \ldots) \\ k_2 & = & (\underbrace{0\ldots0}_{0\ldots B_{f,2}-1} \quad d'_0 \quad \underbrace{0\ldots0}_{\omega_{max}-1} \quad ? \quad \ldots) \end{array}$$

Afin de choisir d_0 et d'_0, il faut déterminer $\omega_{max} \leq W_{max}$ tel que :

$$\exists d_0 \in \overline{\mathscr{D}} \text{ tel que : } 2^{\omega_{max}} \mid (k_1 - d_0) \mod 2^{W_{max}} \text{ et } 2^{\omega_{max}+1} \nmid (k_1 - d_0) \mod 2^{W_{max}},$$
$$\exists d'_0 \in \overline{\mathscr{D}} \text{ tel que : } 2^{\omega_{max}} \mid (k_2 - d'_0) \mod 2^{W_{max}} \text{ et } 2^{\omega_{max}+1} \nmid (k_2 - d'_0) \mod 2^{W_{max}}.$$

On prend d_0 et d'_0 comme coefficients de la *DoubleRDR* de (k_1, k_2), puis on calcule $\frac{k_1 - d_0}{2^{\omega_{max}}}$ et $\frac{k_2 - d'_0}{2^{\omega_{max}}}$, afin d'avancer dans la représentation.

Exemple 5. *Pour $k_1 = 25$ et $k_2 = 51$ (la suite de l'exemple précédent), on a $B_{f,1} = B_{f,2} = 0$.*
Pour $\omega_{max} = 6$, il n'existe pas de coefficient d_0 vérifiant les conditions pour k_1.
De même pour $\omega_{max} = 5$.
Prenons $\omega_{max} = 4$, on pose $d_0 = -23$ et $d'_0 = 3$, et on a :

$$(k_1 - d_0) \mod 2^6 = 48 \text{ divisible par } 2^4 = 16, \text{ et non divisible par } 32,$$
$$(k_2 - d'_0) \mod 2^6 = 48 \text{ divisible par } 2^4 = 16, \text{ et non divisible par } 32.$$

Puis on calcule $\frac{48}{16} = 3$.
Ainsi, on choisit -23 et 3 comme coefficients de k_1 et k_2 respectivement, et on ajoute 3 zéros, ce qui donne dans la représentation :

$$\begin{array}{rcl} 53 & = & (3 \quad -23 \quad 0 \quad 0 \quad 0 \quad ? \quad \ldots) \\ 102 & = & (0 \quad 3 \quad 0 \quad 0 \quad 0 \quad ? \quad \ldots) \end{array}$$

Les derniers coefficients sont 3 pour k_1 et k_2 car $\frac{48}{16} = 3$. Ainsi la DoubleRDR de (k_1, k_2) est :

$$\begin{array}{rcl} 53 & = & (3 \quad -23 \quad 0 \quad 0 \quad 0 \quad 3) \\ 102 & = & (0 \quad 3 \quad 0 \quad 0 \quad 0 \quad 3) \end{array}$$

4.2.3 Fin de la *DoubleRDR*

Lorsqu'on doit recoder les derniers bits (de poids forts) des exposants, on applique la *RDR* à chacun des exposants jusqu'à la fin du recodage, en vérifiant la condition que les coefficients choisis sont inférieurs aux exposants.

Exemple 6. *Pour $k_1 = 7$ et $k_2 = 13$, on regarde chaque exposant indépendamment.*
La RDR de k_1 donne :

$$7 = (-1 \quad 0 \quad 0 \quad 1)$$

Celle de k_2 donne :

$$13 = (-3 \quad 0 \quad 0 \quad 0 \quad 1)$$

La DoubleRDR de (k_1, k_2) donne :

$$\begin{array}{rcl} 7 & = & (-1 \quad 0 \quad 0 \quad 1 \quad 0) \\ 13 & = & (-3 \quad 0 \quad 0 \quad 0 \quad 1) \end{array}$$

4.3 Exemple et Contraintes

4.3.1 Exemple

Prenons un exemple simple afin de comprendre les tenants et les aboutissants de cette méthode.

J'ai choisi $k_1 = 869027$ et $k_2 = 706440$, et $\mathscr{D} = \{1,3,23,27\}$.
On a donc $W_{max} = 6$ et $\overline{\mathscr{D}} = \{1,3,23,27,-1,-3,-23,-27\}$.

La représentation binaire du couple d'exposant (k_1,k_2) est :

$$\begin{array}{rcl} 869027 & = & (1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1) \\ 706440 & = & (0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1) \end{array}$$

Nombre de $\binom{0}{0}$: 6.

En appliquant la *RDR* sur chacun des nombres, on obtient :

$$\begin{array}{rcl} 869027 & = & (3\ \ 0\ \ 0\ \ 0\ \ 0\ \ -27\ \ 0\ \ 0\ \ 0\ \ 3\ \ 0\ \ 0\ \ 0\ \ 0\ \ -27\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1) \\ 706440 & = & (0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ -1\ \ 0\ \ 0\ \ 0\ \ -23\ \ 0\ \ 0\ \ 0\ \ 23\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0) \end{array}$$

Nombre de $\binom{0}{0}$: 11.

On applique la méthode *DoubleRDR* sur ces deux exposants, premièrement il s'agit d'observer les bits faibles de chacun des exposants.
On a donc $B_{f,1} = 0$ et $B_{f,2} = 3$, et donc $B_{f,1} < B_{f,2}$.
Ensuite, comme $\omega = B_{f,1} - B_{f,2} = 3$, il s'agit de trouver le coefficient d_0 de $\overline{\mathscr{D}}$ tel que :

$$\begin{array}{l} 2^3 \mid (k_1 - d_0) \mod 64 \\ 2^4 \nmid (k_1 - d_0) \mod 64 \end{array}$$

Il y a une méthode rapide et efficace pour déterminer rapidement le coefficient d_0, il suffit de calculer $k_1 - \overline{\mathscr{D}} = \{(k_1 - d) \mod 64, d \in \overline{\mathscr{D}}\}$, et de choisir un nombre divisible par 8 et non divisible par 16 dans cet ensemble.
Dans ce cas , on a $k_1 = 35 \mod 64$, et donc on a :
$k_1 - \overline{\mathscr{D}} = \{34,32,12,8,36,38,58,62\}$.
On remarque que 8 est divisible par 8 et non divisible par 16, et donc $d_0 = 27$ convient ($8 = 35 - 27$).

Donc on a :

$$\begin{array}{rcl} 869027 & = & (27\ \ 0\ \ 0\ \ \ldots \\ 706440 & = & (0\ \ 0\ \ 0\ \ \ldots \end{array}$$

Et on effectue :

$$\begin{array}{rcl} \frac{869027-27}{8} & = & 108625 \\ \frac{706440}{8} & = & 88305 \end{array}$$

Puis on note les bits faibles de chacun des restes, $B_{f,1} = B_{f,2} = 0$.
Il y a également une méthode rapide et efficace pour calculer les coefficients d_0 et d_0', on calcule $k_1 - \overline{\mathscr{D}}$ et $k_2 - \overline{\mathscr{D}}$, puis on cherche ω_{max} tel que dans chaque ensemble $k_i - \overline{\mathscr{D}}$, il y ait un nombre divisible par $2^{\omega_{max}}$ et non divisible par $2^{\omega_{max}+1}$.

Ici $k_1 = 17 \mod 64$ et $k_2 = 49 \mod 64$.
Ainsi $k_1 - \overline{\mathscr{D}} = \{16,14,58,54,18,20,40,44\}$.
Et $k_2 - \overline{\mathscr{D}} = \{48,46,26,22,50,52,8,12\}$.
La plus grande puissance de 2 qui divise un des éléments dans chacun des ensembles est 16 car il y a 16 dans le premier ensemble et 48 dans le second. Or 32 ne divise aucun de ces deux nombres. Donc $\omega_{max} = 4$.
Ainsi pour $d_0 = 1$ pour k_1 et $d'_0 = 1$ pour k_2, la *DoubleRDR* de (k_1, k_2) donne :

$$\begin{array}{rcl} 869027 & = & (27 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad \ldots \\ 706440 & = & (0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad \ldots \end{array}$$

Et on effectue :

$$\begin{array}{rcl} \frac{108625-1}{16} & = & 6789 \\ \frac{88305-1}{16} & = & 5519 \end{array}$$

On continue la méthode, on est dans le même cas que précédemment ($B_{f,1} = B_{f,2} = 0$).
Ainsi on réitère le processus :
Ici $k_1 = 5 \mod 64$ et $k_2 = 15 \mod 64$.
Ainsi $k_1 - \overline{\mathscr{D}} = \{4,2,46,42,6,8,28,32\}$.
Et $k_2 - \overline{\mathscr{D}} = \{14,12,56,52,16,18,38,42\}$.
Cette fois-ci, on remarque que $2^5 = 32$ divise un élément de $k_1 - \overline{\mathscr{D}}$ (pour $d_0 = -27$, $k_1 - d_0 = 32$) mais aucun élément de $k_2 - \overline{\mathscr{D}}$, donc $\omega_{max} < 5$.
Puis 16 divise un élement de $k_2 - \overline{\mathscr{D}}$, et le seul élément de $k_1 - \overline{\mathscr{D}}$ divisible par 16 est aussi divisible par 32, donc $\omega_{max} < 4$.
Donc on a 8 la plus grande puissance de 2 divisant à la fois un élément de $k_1 - \overline{\mathscr{D}}$ et un de $k_2 - \overline{\mathscr{D}}$, et telle que 16 ne divise pas ces éléments, donc $\omega_{max} = 3$.
On prend $d_0 = -3$ pour k_1 et $d'_0 = 23$ pour k_2.
Donc on a :

$$\begin{array}{rcl} 869027 & = & (27 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad -3 \quad 0 \quad 0 \quad \ldots \\ 706440 & = & (0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 23 \quad 0 \quad 0 \quad \ldots \end{array}$$

Et on effectue :

$$\begin{array}{rcl} \frac{6789+3}{8} & = & 849 \\ \frac{5519-23}{8} & = & 687 \end{array}$$

On obtient k_1 et k_2 impairs, comme précédemment. Ainsi on réitère le processus :
Ici $k_1 = 17 \mod 64$ et $k_2 = 47 \mod 64$.
Ainsi $k_1 - \overline{\mathscr{D}} = \{16,14,58,54,18,20,40,44\}$.
Et $k_2 - \overline{\mathscr{D}} = \{46,44,24,20,48,50,6,10\}$.
Ainsi, on a $\omega_{max} = 4$ en prenant $d_0 = 1$ pour k_1 et $d'_0 = -1$ pour k_2.
Donc on a :

$$\begin{array}{rcl} 869027 & = & (27 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad -3 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad \ldots \\ 706440 & = & (0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 23 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad \ldots \end{array}$$

Et on effectue :

$$\begin{array}{rcl} \frac{849-1}{16} & = & 53 \\ \frac{687+1}{16} & = & 43 \end{array}$$

Les deux exposants sont inférieurs à 64, il reste à finir le recodage en prenant la *RDR* de chaque reste d'exposants (ici 53 et 43).

On détermine la *RDR* de chaque exposant :

$$\begin{array}{rcl} 53 & = & (-27 \quad 0 \quad 0 \quad 0 \quad -3 \quad 0 \quad 0 \quad 1) \\ 43 & = & (27 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0) \end{array}$$

La *DoubleRDR* de (k_1, k_2) est :

$$\begin{array}{rcl} 869027 & = & (27 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad -3 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad -27 \quad 0 \quad 0 \quad 0 \quad -3 \quad 0 \quad 0 \quad 1) \\ 706440 & = & (0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 23 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 27 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0) \end{array}$$

Nombre de $\binom{0}{0}$: 15.

Remarque 3. *On remarque que le nombre de* $\binom{0}{0}$ *est bien plus élevé (*4 *de plus sur* 20 *bits), que pour la JSF ou pour la RDR appliquée à chaque exposant.*

4.3.2 Contraintes et Objectifs

La *DoubleRDR* est efficace, cependant il y a quelques contraintes pour programmer une telle méthode.

On donne les contraintes liées à la sécurité de l'algorithme dans un premier temps, puis celles liées à la performance dans un second temps.

Contraintes liées à la sécurité.

Il y a des contraintes liées à la résistance à une attaque par canaux auxiliaires, qui est l'objectif principal de la *DoubleRDR*.

- La première condition de la double représentation est l'utilisation de coefficients choisis aléatoirement (l'ensemble $\overline{\mathscr{D}}$).
- Il est important que les résultats ne dépendent pas des coefficients choisis (il faut que les résultats soient similaires lorsqu'on change l'ensemble $\mathscr{D}$).
- La disjonction de cas, lorsque les bits faibles sont égaux ou non, peut poser problème au niveau de la sécurité (car l'utilisation d'un "if" semble nécessaire).

Contraintes liées aux performances

- Le choix d'une taille de fenêtre maximale W_{max} est obligatoire pour choisir les coefficients de la *DoubleRDR*.
- Il faut pouvoir déterminer les prochains bits non nuls de chacun des nombres ($B_{f,1}$ et $B_{f,2}$), ceci peut être un problème du point de vue de la performance.
- Lorsque $B_{f,1} \neq B_{f,2}$, il peut arriver qu'il n'y ait pas de solution au problème :
  Trouver $d_0 \in \overline{\mathscr{D}}$ tel que :

$$\begin{array}{c} 2^{\omega} \mid (k_i - d_0) \mod (2^{W_{max}}) \\ 2^{\omega+1} \nmid (k_i - d_0) \mod (2^{W_{max}}) \end{array}$$

C'est un contrainte embarrassante au niveau de la performance (temps de calcul ET nombre de double zéros).

Objectifs et Définition

Pour rappel, on définit la *DoubleRDR* comme la représentation d'un couple (k_1, k_2) en base 2 vérifiant ces deux objectifs/conditions principaux :

Objectif 1 Les coefficients de k_1 et k_2 sont choisis tels qu'il y ait le même nombre de 0 consécutifs dans les représentations de k_1 et k_2.

Objectif 2 Le nombre de $\binom{0}{0}$ consécutifs, suivant un bit dont le coefficient est non nul est, maximal.

— 5 —

Théorie, Implémentation et Optimisation

Dans une première partie, on détaille un premier algorithme de recodage *DoubleRDR*.

Puis dans un second temps, on explique une implémentation plus efficace et otpimisée que la précédente.

Finalement, la partie 3 montre l'équivalence entre les deux implémentations de la *DoubleRDR*.

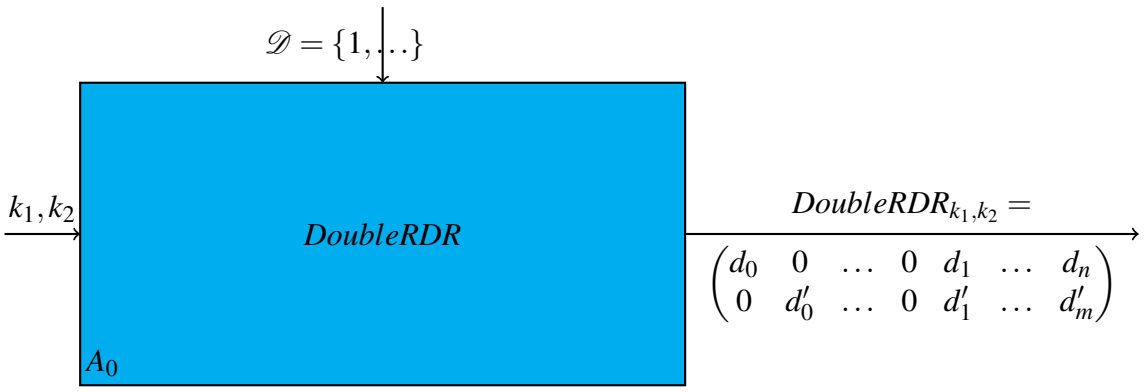
5.1 Implémentation de la *DoubleRDR*

Tout d'abord, afin de tester l'efficacité de la *DoubleRDR*, il est important de créer un algorithme permettant d'appliquer la méthode de recodage décrite dans la partie précédente.

Dans tout le chapitre, on note k_1 et k_2 les deux exposants à recoder et $\mathscr{D}$ l'ensemble des coefficients utilisés dans la représentation.

5.1.1 Squelette et organisation du programme

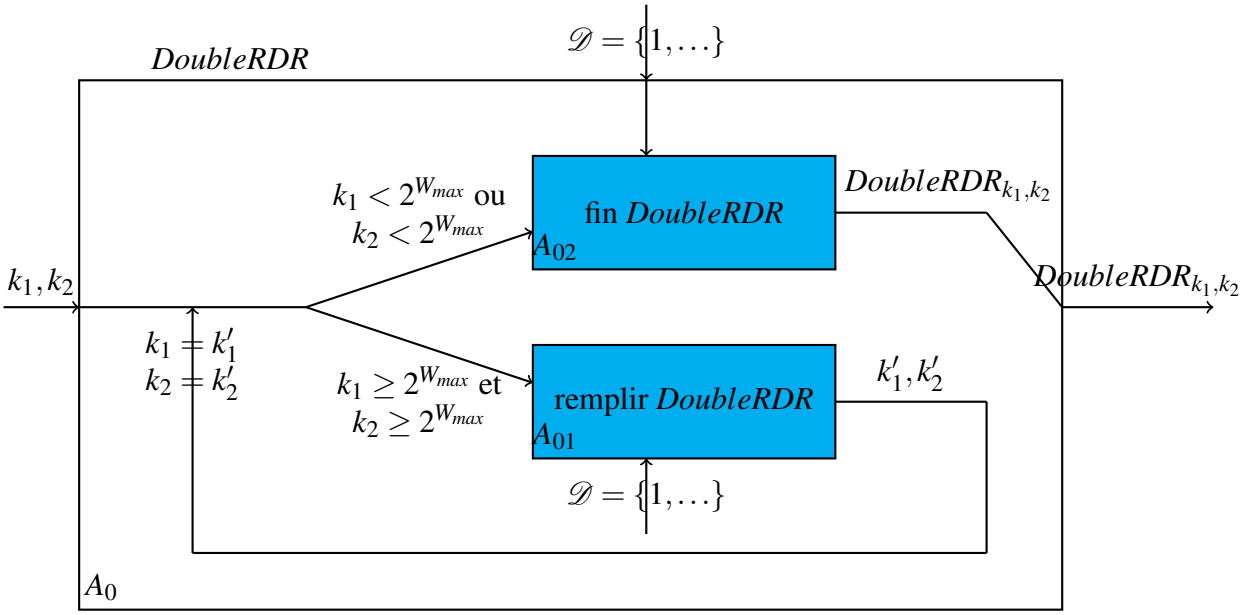
Voici le schéma *SADT* du programme *DoubleRDR* :



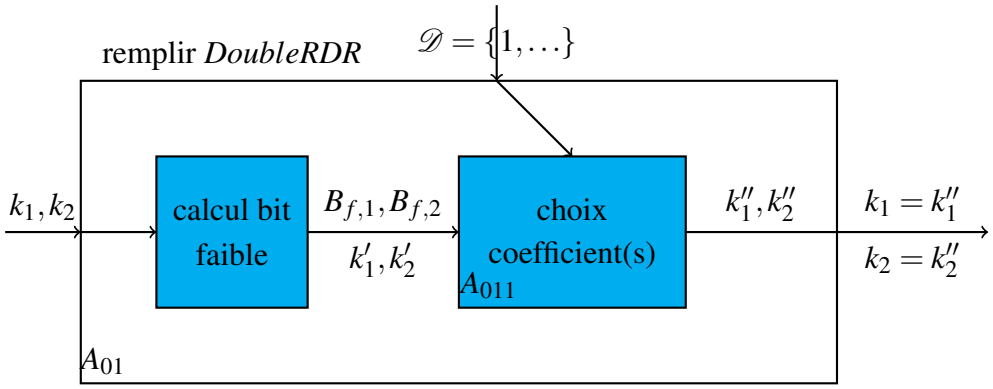
Les entrées du programme sont :

- les exposants k_1 et k_2,
- l'ensemble $\mathscr{D}$ des coefficients possibles.

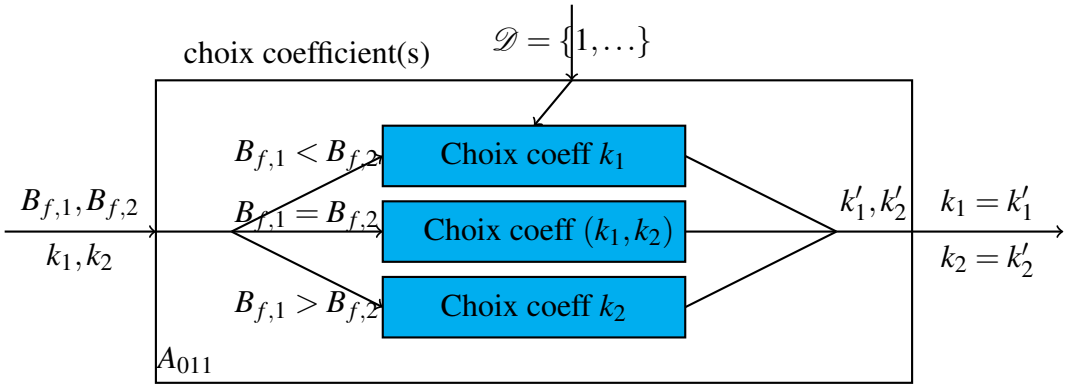
Le programme renvoie un tableau à deux lignes et environ la taille du plus grand nombre parmi k_1 et k_2.



On remarque qu'il faut fonctionner différemment pour la fin du recodage (ie $k_i < 2^{W_{max}}$), c'est pourquoi le programme applique la méthode *DoubleRDR* en recalculant k_1 et k_2 au fur et à mesure jusqu'à ce que l'un des deux exposants soit petit, on applique alors l'algorithme spécialement conçu pour ce cas particulier.



De la même manière que dans l'exemple, il est très important de calculer les bits faibles de chaque exposant avant de calculer le coefficient associé sous peine de choisir le mauvais coefficient.



On distingue, comme dans la méthode *DoubleRDR*, le cas où $B_{f,1} = B_{f,2}$ et les cas où $B_{f,1} \neq B_{f,2}$ (ie $B_{f,1} < B_{f,2}$ et $B_{f,1} > B_{f,2}$)

5.1.2 Les algorithmes de calculs

Dans cette partie, on explique l'algorithme de calcul de $B_{f,i}$, les algorithmes de choix des coefficients et l'algorithme de fin de la *DoubleRDR*.

Calcul de $B_{f,i}$

La première fonction détaillée est la fonction "calcul du bit de poids faible".
Pour ce faire, on divise par 2 l'exposant tant que celui-ci est pair, puis on récupère le nombre de division par 2 pour obtenir B_f, comme détaillé ci-dessous :

Algorithm 7 Algorithme de calcul de B_f pour k

Require: l'exposant k.
Ensure: le nombre B_f et la nouvelle valeur de l'exposant k, noté k'.

```
B_f ← 0
k' ← k
while k' > 1 do
    k' ← k'/2
    B_f ← B_f + 1
end while
return [B_f, k']
```

Choix des coefficients

On présente les 3 algorithmes de choix des coefficients.
L'algorithme pour le cas où $B_{f,1} < B_{f,2}$ est l'algorithme 8.

- L'algorithme calcule d'abord l'ensemble $k_1 - \overline{\mathscr{D}}$ (comme dans l'exemple),
- puis en fixant $pow_1 = 2^\omega$, l'algorithme cherche un coefficient d_0 tel que :
  $pow_1 \mid (k_1 - d_0)$ et $pow_2 = 2 \times pow_1 \nmid (k_1 - d_0)$,
- s'il y a une solution, le programme renvoie d_0, sinon on décremente ω jusqu'à trouver un coefficient d_0 vérifiant $pow_1 \mid (k_1 - d_0)$ et $pow_2 = 2 \times pow_1 \nmid (k_1 - d_0)$.

Pour le cas où $B_{f,1} > B_{f,2}$, il suffit d'utiliser le même algorithme en remplaçant k_1 par k_2 et en inversant les rôles de $B_{f,1}$ et $B_{f,2}$.

Algorithm 8 Algorithme de calcul de d_0 lorsque $B_{f,1} < B_{f,2}$

Require: l'exposant k_1, les bits $B_{f,1}$ et $B_{f,2}$, et l'ensemble $\overline{\mathscr{D}}$ avec Wn (borne).
Ensure: d_0 avec $d_0 \in \overline{\mathscr{D}}$.

```
for i allant de 0 à |D̄| − 1 do
    KmoinsDbarre[i] ← k1 − D̄[i] // l'ensemble k1 − D̄ de l'exemple
end for
ω ← B_{f,2} − B_{f,1}
pow1 ← 2^ω mod 2^{W_max} // on a pow1 = 2^ω, puis pow1 = 2^{ω−1} si on ne trouve pas de solution, etc ...
pow2 ← pow1 × 2 mod 2^{W_max}
c ← 0
while pow1 > 1 et c = 0 do
    i ← 0
    while i < |D̄| et c = 0 do
        S ← KmoinsDbarre[i] mod 2^{W_max}
        if S = 0 mod pow1 et S ≠ 0 mod pow2 then
            d0 ← D̄[i]
            c ← 1
        end if
        i ← i + 1
    end while
    pow1 ← pow1 / 2
    pow2 ← pow2 / 2
end while
return d0
```

L'algorithme pour le cas où $B_{f,1} = B_{f,2}$ est l'algorithme 9.
Pour se faire, on réutilise la méthode de l'exemple :

- L'algorithme calcule $k_1 - \overline{\mathscr{D}}$ et $k_2 - \overline{\mathscr{D}}$, puis on cherche ω_{max} correspondant à ces deux ensembles.
- Pour cela, on initialise ω à $Wn + 2$ (W_{max}) et on regarde dans le premier ensemble si 2^ω divise l'un des éléments, puis dans le deuxième ensemble.
- L'algorithme s'arrête uniquement au moment où 2^ω divise un élément de chacun des ensembles, et tel que $2^{\omega+1}$ ne divise pas ces éléments.
- La deuxième boucle (celle avec la variable i) correspond à la recherche d'un 2^ω divisant un élément de $k_1 - \overline{\mathscr{D}}$.
- La troisième boucle (celle avec la variable j) correspond à la recherche d'un 2^ω divisant un élément de $k_2 - \overline{\mathscr{D}}$.
- La première boucle (celle avec la variable pow) correspond à la recherche du w_{max} commun aux deux ensembles.

Algorithm 9 Algorithme de calcul de d_0 et d'_0 lorsque $B_{f,1} = B_{f,2}$

Require: les exposants k_1 et k_2 et l'ensemble $\overline{\mathscr{D}}$ avec W_n (borne).
Ensure: $[d_0, d'_0]$ avec $d_0, d'_0 \in \overline{\mathscr{D}}$.

```
for i allant de 0 à |D̄| − 1 do
    KmoinsDbarre1[i] ← k1 − D̄[i] // l'ensemble k1 − D̄
end for
for i allant de 0 à |D̄| − 1 do
    KmoinsDbarre2[i] ← k2 − D̄[i] // l'ensemble k2 − D̄
end for
pow ← 2^(Wn+2) // la variable pow correspond à w_max
c ← 0 // valeur d'arrêt qui arrête le programme quand on trouve les coefficients
while pow > 1 et c ≠ 1 do
    c1 ← 0 // valeur d'arrêt pour d0
    c2 ← 0 // valeur d'arrêt pour d'0
    i ← 0
    while i < |D̄| et c1 ≠ 1 do
        S ← KmoinsDbarre1[i]
        if S = 0 mod (pow) et S ≠ 0 mod (2 × pow) then
            c1 ← 1
        end if
        i ← i + 1
    end while
    j ← 0
    while j < |D̄| et c2 ≠ 1 do
        S ← KmoinsDbarre2[j]
        if S = 0 mod (pow) et S ≠ 0 mod (2 × pow) then
            c2 ← 1
        end if
        j ← j + 1
    end while
    pow ← pow/2
    c ← c1 × c2 // c vaut 1 uniquement lorsque c1 et c2 valent 1
end while
d0 ← D̄[i − 1]
d'0 ← D̄[j − 1]
return [d0, d'0]
```

5.2 De l'arithmétique à l'optimisation

Grâce à l'algorithme de la partie précédente, on remarque que la méthode *DoubleRDR* donne des résultats intéressants en terme de nombre de $\binom{0}{0}$ (voir tableau en annexe). On rappelle que l'algorithme de représentation d'un (ou plusieurs) exposant sert à améliorer la vitesse d'éxécution de l'exponentiation, ainsi il est nécessaire d'optimiser le programme précédent.

5.2.1 La recherche dans un tableau, procédé à éviter

De la même manière que pour la *RDR* (voir partie 1), on cherche à éviter la recherche dans un tableau (qui nécessite de parcourir tout le tableau pour chaque coefficient).
Il existe un moyen de contourner la recherche dans un tableau, il s'agit de créer un tableau *DoubleDmax*, de telle sorte que l'indice du tableau donne directement les coefficients d_0 et d'_0 de la *DoubleRDR*.

Afin de créer et remplir le tableau *DoubleDmax*, on rappelle les points suivants :

1) Dans la *RDR*, on utilise un tableau d'une ligne et de $2^{W_{max}}$ colonnes.

2) Les coefficients d_0 et d'_0 de la *DoubleRDR* de (k_1,k_2) sont déterminés uniquement avec $k_1 - \overline{\mathscr{D}}$ et $k_2 - \overline{\mathscr{D}}$, et donc uniquement avec k_1 et k_2 car $\overline{\mathscr{D}}$ ne change pas.

3) On observe les deux exposants sur une fenêtre maximale à chaque choix de coefficients, l'algorithme dépend uniquement des valeurs des exposants sur cette fenêtre.

5.2.2 Création et initialisation du tableau *DoubleDmax*

On constate que le tableau *DoubleDmax* est constitué de deux entrées, k_1 et k_2.
Les valeurs de k_1 et k_2 sur une fenêtre de taille $W_n + 2$ définissent un unique choix de coefficients.

Ainsi, on remarque qu'il y a 2^{Wn+2} possibilités pour k_1 et k_2, ce qui fait exactement 2^{2Wn+4} pour le couple (k_1,k_2), et la construction d'un tableau à 2^{Wn+2} lignes et 2^{Wn+2} colonnes et où chaque case contient un couple (d_0,d'_0) est appropriée.

De plus, dès que k_1 ou k_2 est pair, le coefficient associé d_0 est logiquement nul. Ainsi par un souci d'efficacité, on initialise *DoubleDmax* à $(0,0)$ partout, et on modifie ensuite les coefficients nuls pour lesquels k_1 et/ou k_2 est impair.
Il y a donc 2^{2Wn+3} doublets de coefficients à remplacer (exactement la moitié).

5.2.3 Remplissage du tableau *DoubleDmax*

Etant donné que chacun des algorithmes utilisent des boucles déterminées par la variable *pow* qui est liée au nombre de $\binom{0}{0}$ dans la représentation de (k_1,k_2), il est logique que la construction de *DoubleDmax* se fasse avec cette même boucle.
Ainsi, on utilise une boucle *while* dans laquelle la variable *pow* est initialisée à 2^{Wn+2}, puis décrémentée à 2^{Wn+1}, puis à 2^{Wn}, et ainsi de suite jusqu'à $pow = 1$.

On remplit les cases du tableau avec $\overline{\mathscr{D}}$ de cette manière,
$(d_0 = \overline{\mathscr{D}}[i] + pow, d'_0 = \overline{\mathscr{D}}[j] + pow)$ où $0 \leq i,j < |\overline{\mathscr{D}}|$, correspondant à la ligne $\overline{\mathscr{D}}[i] + pow$ et à

la colonne $\overline{\mathscr{D}}[j] + pow$.
Puis on fait de même avec $(d_0 = \overline{\mathscr{D}}[i] + 3 \times pow, d_0' = \overline{\mathscr{D}}[j] + 3 \times pow)$ où $0 \leq i, j < |\overline{\mathscr{D}}|$ On réitère le procédé ($1 \times pow$, $3 \times pow$, $5 \times pow$, $7 \times pow$, ...) jusqu'à retomber sur $1 \times pow$. Puis on décremente $pow \leftarrow \frac{pow}{2}$.

En réitérant le procédé jusqu'à ce que $pow = 2$, on remplit ainsi toutes les cases *impaires* du tableau (lignes et colonnes impaires).

Exemple 7. *Pour* $\mathscr{D} = \{1,3,23,27\}$*, on a :*

- *Pour* $pow = 64$*, on remplit les cases* $(1,1)$*,* $(1,3)$*,* $(1,23)$*,* $(1,27)$*,* $(1,37)$*,* $(1,41)$*, ...*
  puis on remplit les cases $(3,1)$*,* $(3,3)$*,* $(3,23)$ *et* $(3,27)$ ...
- *Pour* $pow = 32$*, on remplit les cases* $(1+32,1+32)$*,* $(1+32,3+32)$*, ...*
- *Pour* $pow = 16$*, on remplit les cases* $(1+16,1+16)$*,* $(1+16,3+16)$*, ... puis on remplit les cases* $(1+48,1+48)$*,* $(1+48,3+48)$ ...

Il reste à remplir les cases des lignes paires et des colonnes impaires, et inversement (c'est-à-dire le cas où $B_{f,1} < B_{f,2}$ et inversement).
On peut appliquer le même processus que précédemment en remplaçant l'un des coefficients de $\overline{\mathscr{D}}$ par 0.
En utilisant la même boucle *while*, on remplit les cases de cette manière,
$(d_0 = 0 + pow, d_0' = \overline{\mathscr{D}}[j] + pow)$ où $0 \leq j < |\overline{\mathscr{D}}|$, correspondant à la ligne pow et à la colonne $\overline{\mathscr{D}}[j] + pow$.
Puis on fait de même avec $(d_0 = 0 + 3 \times pow, d_0' = \overline{\mathscr{D}}[j] + 3 \times pow)$ où $0 \leq j < |\overline{\mathscr{D}}|$ On réitère le procédé ($1 \times pow$, $3 \times pow$, $5 \times pow$, $7 \times pow$, ...) jusqu'à retomber sur $1 \times pow$. Puis on décremente $pow \leftarrow \frac{pow}{2}$.

Exemple 8. *Pour* $\mathscr{D} = \{1,3,23,27\}$*, on a :*

- *Pour* $pow = 64$*, on remplit les cases* $(0,1)$*,* $(0,3)$*,* $(0,23)$*,* $(0,27)$*,* $(0,37)$*,* $(0,41)$*, ...*
- *Pour* $pow = 32$*, on remplit les cases* $(0+32,1+32)$*,* $(0+32,3+32)$*, ...*
- *Pour* $pow = 16$*, on remplit les cases* $(0+16,1+16)$*,* $(0+16,3+16)$*, ... puis on remplit les cases* $(0+48,1+48)$*,* $(0+48,3+48)$ ...

Finalement, avec cette méthode, toutes les cases ne sont pas remplies.
Par exemple, pour le cas où $\mathscr{D} = \{1,3,23,27\}$, la case $(32,17)$ est toujours initialisée à $(d_0, d_0') = (0,0)$.
Dans ce cas, il s'agit de remplir les cases manquantes en utilisant les propriétés du nombre impair (5 dans l'exemple), ainsi que la boucle *while* utilisée précédemment.
On procède ainsi,
on remplit la case $(d_0 = 0 + 2 \times pow, d_0' = \overline{\mathscr{D}}[j] + pow)$ si celle ci est encore initialisée à $(0,0)$, puis on remplit la case $(0 + 4 \times pow, d_0' = \overline{\mathscr{D}}[j] + pow)$, puis on continue avec $6 \times pow$ et ainsi de suite jusqu'à retomber sur $2 \times pow$.

Exemple 9. *Pour* $\mathscr{D} = \{1,3,23,27\}$*, on a :*

- *Pour* $pow = 16$*, on remplit les cases* $(32,1+16)$*,* $(32,3+16)$*,*
  puis $(32,1+48)$*,* $(32,3+48)$*, ...*

- *Pour pow* $= 8$*, on remplit les cases* $(16, 1+8)$*,* $(16, 3+8)$*, ...*
  puis $(16, 1+24)$*,* $(16, 3+24)$*, ...*
  puis $(32, 1+8)$*,* $(32, 3+8)$ *...*

5.2.4 Optimisation avancée

On remarque dans la partie précédente que le premier et le deuxième procédé de remplissage du tableau *DoubleDmax* fonctionne de la même manière, en ajoutant 0 à $\overline{\mathscr{D}}$, on peut fusionner les deux processus en un seul programme.

On peut également utiliser un compteur afin d'économiser un certain nombre de calculs. Il n'est pas nécessaire de calculer tous les ensembles $(\mathscr{D}+2^k, \mathscr{D}+2^k)$,
par exemple pour $\overline{\mathscr{D}} = \{1, 3, 23, 27, 37, 41, 61, 63\}$, le tableau *DoubleDmax* peut se remplir avec $(\mathscr{D}+2^k, \mathscr{D}+2^k)$, pour $k \geq 3$.

On observe finalement que les cases du tableau sont symmétriques (les cases $(43, 21)$ et $(21, 43)$ renvoient les mêmes coefficients), on peut diviser par deux la taille et les calculs du tableau *DoubleDmax* en associant le coefficient de la case (k_1, k_2) à k_1 et celui de la case (k_2, k_1) à k_2.

5.2.5 Algorithmes

Voici les différents algorithmes traitant la création et le remplissage du tableau *DoubleDmax*.

Création de *DoubleDmax*

Pour créer un tableau à 2^{W_n+2} lignes et 2^{W_n+2} colones, il suffit d'allouer une mémoire suffisante à un pointeur de pointeurs, puis de faire une boucle pour allouer la mémoire suffisante à tous les pointeurs.

Algorithm 10 Algorithme de création de *DoubleDmax*

Require: W_n.
Ensure: le tableau *DoubleDmax* vide.
  $pow \leftarrow 2^{W_n+2}$
  $**DoubleDmax = malloc(pow \times sizeof(*DoubleDmax))$ {création d'un tableau contenant *pow* lignes}
  for *i* allant de 0 à $pow - 1$ **do**
    $*DoubleDmax[i] = malloc(pow \times sizeof(**DoubleDmax))$ {allocation de chaque ligne d'une taille *pow* fois la taille d'un élément (un *int* dans notre cas)}
  end for
  return *DoubleDmax*

Remplissage de *DoubleDmax* : Première fonction

Voici le premier algorithme correspondant aux deux premiers procédés de la partie 5.2.3 (avec 0 ajouté au tableau $\overline{\mathscr{D}}$),

Algorithm 11 Algorithme de remplissage de *DoubleDmax*

Require: une puissance *pow*, l'ensemble $\overline{\mathscr{D}}$ avec Wn (borne) et le tableau *DoubleDmax*.
Ensure: *DoubleDmax* partiellement rempli.

```
for i allant de 0 à |𝒟̄| − 1 do
  d1 ← 𝒟̄[i]
  pow1 ← pow
  while pow1 ≤ 2^(Wn+2) do
    d1 ← d1 + pow1  mod 2^(Wn+2)
    for j allant de 0 à |𝒟̄| do
      d2 ← 𝒟̄[j]
      pow2 ← pow
      while pow2 ≤ 2^(Wn+2) do
        d2 ← d2 + pow2  mod 2^(Wn+2)
        if DoubleDmax[d1][d2] = 0 then
          if i < |D| then
            DoubleDmax[d1][d2] = D[i]
          else
            DoubleDmax[d1][d2] = −D[i − |D|]
          end if
          if j < |D| then
            DoubleDmax[d2][d1] = D[j]
          else
            DoubleDmax[d2][d1] = −D[j − |D|]
          end if
        end if
        pow2 ← pow2 + 2 × pow
      end while
    end for
    pow1 ← pow1 + 2 × pow
  end while
end for
return DoubleDmax
```

Remplissage de *DoubleDmax* : Deuxième fonction

Voici le deuxième algorithme permettant de traiter le troisième procédé de la partie 5.2.3,

5.3 Equivalence et Preuve

Les deux algorithmes expliqués précédemment semblent équivalents, il reste désormais à prouver cette équivalence.

Algorithm 12 Algorithme de complètion de *DoubleDmax*

Require: une puissance *pow*, l'ensemble $\overline{\mathscr{D}}$ avec Wn (borne).
Ensure: *DoubleDmax*.

```
pow1 ← 0
while pow1 ≤ 2^(Wn+2) do
  d1 ← pow1
  pow2 ← pow
  while pow2 ≤ 2^(Wn+2) do
    for j allant de 0 à |D̄| do
      d2 ← D̄[j]
      d2 ← d2 + pow2  mod 2^(Wn+2)
      if DoubleDmax[d1][d2] = 0 then
        if i < |D| then
          DoubleDmax[d1][d2] = D[i]
        else
          DoubleDmax[d1][d2] = −D[i − |D|]
        end if
        if j < |D| then
          DoubleDmax[d2][d1] = D[j]
        else
          DoubleDmax[d2][d1] = −D[j − |D|]
        end if
      end if
    end for
    pow2 ← pow2 + 2 × pow
  end while
  pow1 ← pow1 + 2 × pow
end while
return DoubleDmax
```

5.3.1 La définition de la *DoubleRDR*

Definition 1. *La DoubleRDR d'un couple d'exposant est une représentation de ce couple en base* 2 *de la forme*

$$\begin{array}{rcl} k_1 & = & (d_0 \quad \dots \quad d_k \quad \dots \quad d_n) \\ k_2 & = & (d'_0 \quad \dots \quad d'_k \quad \dots \quad d'_m) \end{array}$$

telle que :

- *Les coefficients d_i et d'_i appartiennent à $\mathscr{D} \cup -\mathscr{D} \cup \{0\}$ où $\mathscr{D}$ est prédéfini.*
- *Si $k_{1,i} = \frac{k_1 - d_0 - d_1 \dots - d_{i-1}}{2^i}$ est pair, alors $d_i = 0$.*
- *Si $k_{1,i}$ est impair (resp pair) et $k_{2,i}$ est pair (resp impair), il existe $w \leq W_{max}$ tel que :*

$$2^w \mid k_{2,i} \text{ et } 2^{w+1} \nmid k_{2,i}$$

alors d_i est tel que :

Il existe $m \leq w$ tel que $2^m \mid (k_{1,i} - d_i)$ et $2^{m+1} \nmid (k_{1,i} - d_i)$ et il n'existe pas d'élément $j > m$, pour lequel il existe d_i élément de $\overline{\mathscr{D}}$ vérifiant les conditions précédentes.

- *Si $k_{1,i}$ et $k_{2,i}$ sont pairs, alors d_i et d'_i sont tels que :*

Il existe $m \leq w$ tel que :
$2^m \mid (k_{1,i} - d_i)$ et $2^m \mid (k_{2,i} - d'_i)$,
$2^{m+1} \nmid (k_{1,i} - d_i)$ et $2^{m+1} \nmid (k_{2,i} - d'_i)$, et
il n'existe pas d'élément $j > m$, d_i et d'_i élément de $\overline{\mathscr{D}}$ vérifiant les conditions précédentes.

Remarque 4. *En d'autres termes, on choisit les coefficients d_i et d'_i tels que :*

1) *les prochains coefficients d_j et d'_j non nuls se situent sur le même bit,*
2) *il y ait un maximum de coefficients d_j et d'_j nuls consécutivement (le plus grand nombre de $\binom{0}{0}$ après $\binom{d_j}{d'_j}$)*

5.3.2 Algorithme et Mathématiques

Proposition 2. *Soit $\bar{k}_{1,i} = \frac{k_1 - d_0 - d_1 \ldots - d_{i-1}}{2^i} \mod 2^{W_n+2}$ et $\bar{k}_{2,i} = \frac{k_2 - d'_0 - d'_1 \ldots - d'_{i-1}}{2^i} \mod 2^{W_n+2}$.*

Le tableau DoubleDmax construit dans la partie précédente est telle que :
$d_i = DoubleDmax[\bar{k}_{1,i}][\bar{k}_{2,i}]$
$d'_i = DoubleDmax[\bar{k}_{2,i}][\bar{k}_{1,i}]$
où d_i et d'_i sont les i èmes coefficients de la DoubleRDR de (k_1, k_2).

Preuve 3. *Soient k_1 et k_2 deux exposants, soit i un entier plus petit que la taille de k_1 et k_2, soient d_i et d'_i les i ème coefficients dans la DoubleRDR de (k_1, k_2).*

- *Si $k_{1,i}$ et $k_{2,i}$ sont pairs, alors $\bar{k}_{1,i}$ et $\bar{k}_{2,i}$ sont pairs,*
  donc $DoubleDmax[\bar{k}_{1,i}][\bar{k}_{2,i}] = d_i = 0$ et
  $DoubleDmax[\bar{k}_{2,i}][\bar{k}_{1,i}] = d'_i = 0$.

- *Si $k_{1,i}$ est impair (resp pair) et $k_{2,i}$ est pair (resp impair), alors $\bar{k}_{1,i}$ est impair (resp pair) et $\bar{k}_{2,i}$ est pair (resp impair),*
  donc $DoubleDmax[\bar{k}_{2,i}][\bar{k}_{1,i}] = d'_i = 0$ et il existe w tel que $2^w \mid \bar{k}_{2,i}$ et $2^{w+1} \nmid \bar{k}_{2,i}$.
  Soit $d = DoubleDmax[\bar{k}_{1,i}][\bar{k}_{2,i}]$.
  Avec l'algorithme DoubleDmax, ils existent m et l tels que :

$$\bar{k}_{1,i} = d + l \times 2^m \mod 2^{W_n+2}, \text{ où } l \text{ impair.}$$

  On a donc, $2^m \mid (\bar{k}_{1,i} - d)$ et $2^{m+1} \nmid (\bar{k}_{1,i} - d)$,
  avec $m \leq w$ car $2^w \mid \bar{k}_{2,i}$.
  Il reste à montrer qu'il n'existe pas d'élément $j > m$, pour lequel il existe un élément d'' vérifiant les conditions précédentes.

Supposons qu'ils existent un tel j et un tel d'', on a $2^j \mid (\bar{k}_{1,i} - d'')$ et $2^{j+1} \nmid (\bar{k}_{1,i} - d'')$.
Donc il existe l impair tel que $\bar{k}_{1,i} = d'' + l \times 2^j \mod 2^{W_n+2}$.
Dans ce cas, d'après le fonctionnement de l'algorithme, on a $DoubleDmax[\bar{k}_{1,i}][\bar{k}_{2,i}] = d''$, ce qui est contradictoire.
Donc $d = d_i$ et $d' = d'_i$.

- *Si $k_{1,i}$ et $k_{2,i}$ sont impairs, alors $\bar{k}_{1,i}$ et $\bar{k}_{2,i}$ sont impairs.*
Soient $d = DoubleDmax[\bar{k}_{1,i}][\bar{k}_{2,i}]$ et $d' = DoubleDmax[\bar{k}_{2,i}][\bar{k}_{1,i}]$.
Avec l'algorithme DoubleDmax, ils existent m, l_1 et l_2 tels que :

$$\bar{k}_{1,i} = d + l_1 \times 2^m \mod 2^{W_n+2}, \text{ où } l_1 \text{ impair,}$$
$$\bar{k}_{2,i} = d' + l_2 \times 2^m \mod 2^{W_n+2}, \text{ où } l_2 \text{ impair.}$$

On a donc :

$$2^m \mid (k_{1,i} - d) \text{ et } 2^m \mid (k_{2,i} - d'),$$
$$2^{m+1} \nmid (k_{1,i} - d) \text{ et } 2^{m+1} \nmid (k_{2,i} - d')$$

Il reste à montrer qu'il n'existe pas d'élément $j > m$, pour lequel ils existent $d^{(2)}$ et $d^{(3)}$ dans $\overline{\mathscr{D}}$ vérifiant les conditions précédentes.

Supposons qu'ils existent un tel j et $d^{(2)}$ et $d^{(3)}$, on a $2^j \mid (\bar{k}_{1,i} - d^{(2)})$ et $2^{j+1} \nmid (\bar{k}_{1,i} - d^{(2)})$, et $2^j \mid (\bar{k}_{2,i} - d^{(3)})$ et $2^{j+1} \nmid (\bar{k}_{2,i} - d^{(3)})$.
Donc il existe l_1 impair tel que $\bar{k}_{1,i} = d^{(2)} + l_1 \times 2^j \mod 2^{W_n+2}$, et
il existe l_2 impair tel que $\bar{k}_{2,i} = d^{(3)} + l_2 \times 2^j \mod 2^{W_n+2}$.
Dans ce cas, d'après le fonctionnement de l'algorithme, on a $DoubleDmax[\bar{k}_{1,i}][\bar{k}_{2,i}] = d^{(2)}$ et $DoubleDmax[\bar{k}_{2,i}][\bar{k}_{1,i}] = d^{(3)}$, ce qui est contradictoire.
Donc $d = d_i$ et $d' = d'_i$.

— 6 —

Performances et Aléatoire : l'importance des coefficients

L'algorithme et la méthode sont optimisés, cependant un paramètre important n'a pas encore été discuté, quels sont les performances (en terme de $\binom{0}{0}$) de la *DoubleRDR* ?

L'objectif le plus important de ce recodage est d'être résistant à une attaque par canal auxiliaire. Pour ce faire, il est nécessaire que les résultats obtenus soient indépendants des coefficients choisis, car des variations trop importantes donneraient systématiquement des éléments sur l'exposant (et donc le secret !).

6.1 Le choix des coefficients

Nous allons voir dans cette partie, l'étude complète des coefficients de $\mathscr{D}$ pour certains paramètres de l'algorithme.

6.1.1 Deux choix , deux résultats

On remarque que le choix de l'ensemble $\mathscr{D}$ influe grandement sur les performances de l'algorithme.

Voici un exemple avec deux choix de coefficients particulièrement significatifs.
Pour 1000 tests par choix, des exposants de 4096 bits, et $|\mathscr{D}| = 4$.
Pour $\mathscr{D} = \{1,3,23,27\}$, le nombre moyen de $\binom{0}{0}$ est 2926.
Pour $\mathscr{D} = \{1,15,17,31\}$, le nombre moyen de $\binom{0}{0}$ est 1566.

Ce constat est problèmatique, deux questions se posent alors, quelle est l'influence des coefficients sur les performances de l'algorithme ? D'où vient cette disparité ?

La partie suivante s'attache à donner un élément de réponse à la première question.

6.1.2 Etude sur les coefficients

Pour effectuer les tests, on fixe certains paramètres afin d'observer l'influence des coefficients de $\mathscr{D}$ sur les résultats du recodage. La taille et la borne de $\mathscr{D}$ sont fixés à 4 et à 32 respective-

ment, et la taille des exposants est fixée à 4096. Une fois tous ces paramètres établis, on teste tous les coefficients, c'est-à-dire tous les ensembles $\mathscr{D}$ possibles, il y en a 455 (3 parmi 15).

Voici les résultats rangés en 5 catégories distinctes dans le tableau ci-dessous.

TABLE 6.1 : Performances suivant $\mathscr{D}$ (1000 tests)

Nombre de $\binom{0}{0}$	Pourcentages de $\binom{0}{0}$	Nombre de $\mathscr{D}$	Pourcentages de $\mathscr{D}$
$x > 2900$	$[70.8;100]$	63	13.8
$2900 > x > 2800$	$[68.4;70.8[$	193	42.4
$2800 > x > 2700$	$[65.9;68.4[$	96	21.1
$2700 > x > 2500$	$[61;65.9[$	68	14.9
$x < 2500$	$[0;61[$	35	7.7

6.1.3 Conséquences

Une fois ces résultats obtenus, on peut noter les difficultés que ceux ci engendrent et les solutions pour y remédier.

Les difficultés que nous avons listés sont les suivantes :

- Certains résultats sont en dessous des performances que l'on cherche à obtenir.
- Ne pas prendre toutes les possibilités de coefficients entraine une baisse de sécurité car il y a une baisse d'aléatoire.
- Il n'est pas certain que nous puissions identifier les coefficients par catégorie, encore moins par les catégories définies dans le tableau de la partie précédente.

Face à ces observations, nous avons listé plusieurs possibilités :

Piste 1 : La *DoubleRDR*, on randomise le choix des coefficients sans faire aucune sélection.

Avantage : Résistant à "Side Channel Attack".

Problème : Résultats très différents selon les coefficients, parfois une perte importante d'efficacité.

Piste 2 : La *Double semi − RDR*, on randomise le choix des coefficients en sélectionnant au préalable les coefficients qui fonctionnent le mieux, ou non , cette sélection définit ainsi le niveau de performance et le niveau de sécurité que l'on met dans l'algorithme.

Avantage : Adaptable, plus performant que la *DoubleRDR*.

Problème : Moins résistant à "Side Channel Attack" que la *DoubleRDR*.

Piste 3 : Le *Double − wNAF*, on ne randomise absolument rien, le choix des coefficients correspond à tous les nombres impairs entre 1 et la borne de $\mathscr{D}$, à la manière du *wNAF* pour l'exponentiation.

Avantage : Optimal au point de vue performance, pas d'appel à la fonction aléa.

Problème : Pas résistant à "Side Channel Attack".

La piste 2 propose plus de choix, et est expliquée dans la partie suivante.
Les pistes 1 et 3 n'ont pas été traitées.
Est-il possible de ranger les coefficients de $\mathscr{D}$ et si oui, comment ?

6.2 Liens entre les coefficients

Afin de répondre à la question de la partie précédente, on observe attentivement les résultats du tableau statistique sur les coefficients, et on note tous les ensembles $\mathscr{D}$ donnant des performances optimums.
En regardant attentivement ces coefficients, on remarque le point commun entre tous ces quadruplets.

6.2.1 Les coefficients "optimums"

Pour rappel, la borne de $\mathscr{D}$ est 32, sa taille à 4, et 1 est obligatoirement dans $\mathscr{D}$.

On commence par noter tous les $\mathscr{D}$ possibles pour chaque niveau de performance.

Pour le premier niveau, on obtient :

TABLE 6.2 : Ensembles $\mathscr{D}$ pour performances optimales

63 ensembles donnant les meilleurs résultats (>2900)				
1,3,5,9	1,3,5,23	1,3,5,25	1,3,7,11	1,3,7,21
1,3,7,27	1,3,9,11	1,3,9,21	1,3,9,27	1,3,11,23
1,3,11,25	1,3,21,23	1,3,21,25	1,3,23,27	1,3,25,27
1,5,7,13	1,5,7,19	1,5,7,29	1,5,9,13	1,5,9,19
1,5,9,29	1,5,13,23	1,5,13,25	1,5,19,23	1,5,19,25
1,5,23,29	1,5,25,29	1,7,11,13	1,7,11,19	1,7,11,29
1,7,13,21	1,7,13,27	1,7,19,21	1,7,19,27	1,7,21,29
1,7,27,29	1,9,11,13	1,9,11,19	1,9,11,29	1,9,13,21
1,9,13,27	1,9,19,21	1,9,19,27	1,9,21,29	1,9,27,29
1,11,13,23	1,11,13,25	1,11,19,23	1,11,19,25	1,11,23,29
1,11,25,29	1,13,21,23	1,13,21,25	1,13,23,27	1,13,25,27
1,19,21,23	1,19,21,25	1,19,23,27	1,19,25,27	1,21,23,29
1,21,25,29	1,23,27,29	1,25,27,29		

Tous ces quadruplets ont un point commun, lorsqu'on regarde la congruence modulo 16, on obtient systématiquement l'ensemble $\{\bar{1},\pm\bar{3},\pm\bar{5},\pm\bar{7}\}$.

Par exemple, $\{1,3,5,9\} \mod 16 = \{\bar{1},\bar{3},\bar{5},-\bar{7}\}$,
et $\{1,11,25,29\} \mod 16 = \{\bar{1},-\bar{5},-\bar{7},-\bar{3}\}$.

Ainsi pour définir le niveau optimal pour ces paramètres, il suffit de prendre $\mathscr{D} = \{1,d_1,d_2,d_3\}$ tel que :

$$\begin{aligned} d_1 &= \pm 3 \mod 2^4 \\ d_2 &= \pm 5 \mod 2^4 \\ d_3 &= \pm 7 \mod 2^4 \end{aligned}$$

On a donc 4 possibilités pour chaque coefficient d_1, d_2 et d_3 (par exemple 3, 13, 19 et 29 pour d_1), ce qui fait en tout 64 possibilités, au lieu de 63. Il s'agit de remarquer que $1,3,5,7$ n'est pas dans cette catégorie, du fait qu'aucun des éléments ne dépassent 8.

6.2.2 Les autres coefficients

La partie précédente a permis de remarquer qu'il y avait un lien arithmétique entre tous ces coefficients.

Level 3 : un intrus

Après le $Double-wNAF$ (Level 1), et la *DoubleRDR* avec les coefficients optimums (Level 2), il s'agit de trouver le lien entre les coefficients de la deuxième ligne du tableau ($2900 > x > 2800$), qui contient 193 possibilités pour $\mathscr{D}$, soit environ 42% de l'ensemble des $\mathscr{D}$ possibles.

Le premier quadruplet de l'ensemble est $1,3,5,7$, qui n'est pas très significatif car il correspond au quadruplet du $Double-wNAF$ pour $|\mathscr{D}| = 4$, c'est un cas très particulier.

Ensuite, on constate qu'à l'inverse des coefficients du niveau 2, il manque toujours un élément dans l'ensemble $\{\bar{1}, \pm\bar{3}, \pm\bar{5}, \pm\bar{7}\}$, tous sont susceptibles d'y être absents sauf $\bar{1}$ évidemment.

De par le constat précédent, on peut conclure que nécessairement deux des quatre coefficients ont la même valeur modulo 16 (au signe près).

Il suffit de choisir deux nombres parmi $\pm\bar{3}$, $\pm\bar{5}$ et $\pm\bar{7}$,
pour choisir le dernier coefficient, il suffit de prendre un nombre ayant la même valeur que l'un des trois coefficients précédents modulo 16 (au signe près).

Ainsi, pour résumer, il suffit de choisir un intrus parmi 3, 5 et 7, on applique les mêmes règles que précédemment pour les deux premiers coefficients (les deux restants après avoir eliminer l'intrus), puis finalement on fait $16 + \mathscr{D}[i]$ ou $16 - \mathscr{D}[i]$, où $i = 0, 1$ ou 2, pour obtenir le dernier coefficient.

Par cette technique, on obtient les coefficients de niveau 3, ce qui fait $4 \times 4 \times 6 \times 3 = 288$ possibilités, dont certaines se répètent ($1,3,5,13$ et $1,13,5,3$ par exemple). Le calcul exact est dans la remarque suivante.

Remarque 5. *Tout d'abord il y a trois possibilités pour l'intrus.*
*Pour chacun des deux premiers coefficients à choisir, il y a quatre possibilités (*3, 13, 19 *et* 29 *pour* 3*).*
Finalement pour le dernier coefficient, il y a 6 *possibilités (*$16-1$, $16+1$, $16-\mathscr{D}[1]$, $16+\mathscr{D}[1]$, $16-\mathscr{D}[2]$, $16+\mathscr{D}[2]$*).*
D'où ce résultat de $4 \times 4 \times 6 \times 3 = 288$*, cependant il y a des répétitions !*
L'exemple donné ($1,3,5,13$ *et* $1,13,5,3$*) est significatif, cela se produit pour la majorité des quadruplets, le seul cas où il n'y a pas répetition, c'est lorsque* $\mathscr{D}[2] = 16+1$ *ou* $16-1$*. En effet, dans ce cas la répétition est impossible, dans tous les autres cas les quadruplets sont en double.*
Donc le calcul exact est : $288 - \frac{288-4\times4\times2\times3}{2} = 192$*, le dernier quadruplet est évidemment* $1,3,5,7$.

Remarque 6. *Tous les quadruplets ne donnent pas forcément les mêmes résultats, certains sont proche de* 2850*, d'autres atteignent à peine les* 2800*, parfois même un petit peu inférieurs, mais ils ont tous ce lien arithmétique en commun.*

Level 4 : changement minimum pour un résultat bien différent

Le niveau 4 est très proche du niveau précédent.

Pour obtenir les quadruplets du niveau 4, il suffit d'appliquer la même méthode que le niveau 3 en modifiant une petite partie.

Tout d'abord, on choisit l'intrus, puis on choisit les deux premiers coefficients.
Le changement est dans la dernière étape, au lieu de choisir le dernier coefficient parmi $16-d$ et $16+d$, il suffit cette fois de choisir parmi $32-d$ où d est l'un des trois coefficients précédents.

Ce qui donne donc exactement $\frac{192}{2} = 96$ possibilités.

Level 5 : dernier casse-tête avant la fin

Le dernier niveau à décortiquer est le niveau 5, en observant attentivement les coefficients de $\mathscr{D}$ modulo 16, on remarque l'absence de $\pm\bar{7}$, c'est-à-dire qu'il n'y a ni 7, ni 9, ni 23 et ni 25 dans tous les quadruplets.

Le deuxième constat est le suivant, tous les quadruplets ont $\bar{1}$ en commun, et ils ont tous un élément de la forme $\pm\bar{3}$ (3, 13, 19 et 29) ou $\pm\bar{5}$ (5, 11, 21 ou 27).

En résumé les quadruplets vérifient cette règle simple : chaque quadruplet possède au moins élément non congru à à 1 modulo 16, ou bien cet élément est congru à $\pm\bar{3}$, ou bien il est congru à $\pm\bar{5}$. Si cet élément est congru à $\pm\bar{3}$, alors les deux éléments restants sont congrus à $\pm\bar{1}$ ou à $\pm\bar{3}$, sinon les deux éléments restants sont congrus à $\pm\bar{1}$ ou à $\pm\bar{5}$.

Remarque 7. *Pour compter le nombre des quadruplets de niveau* 5*, il suffit de distinguer trois cas simples :*

Cas 1 *: Il n'y a qu'un seul des coefficients qui est congru à* ± 3 *(ou* ± 5*).*
Dans ce cas, il est aisé de constater qu'il y a 4 *possibilités pour le coefficient non congru à* 1*.*
Pour les deux coefficients, il choisir de choisir parmi ces trois possibilités : 15*,* 17 *et* 31 *(*$\pm\bar{1}$*).*
Le choix des derniers coefficients offre 2 *parmi* 3 *(*$= 3$*) quadruplets possibles.*
Le cas 1 *regroupe donc* $2 \times 4 \times 3 = 24$ *quadruplets possibles.*

Cas 2 *: Tous les coefficients sont congrus à* ± 3 *(ou* ± 5*).*
Dans ce cas, le calcul est simple, le choix des 3 *coefficients se fait parmi les* 4 *possibilités suivantes :* 3*,* 13*,* 19 *et* 29*.*
Le cas 2 *dénombre* $2 \times \binom{4}{3} = 8$*.*

Cas 3 *: Il y a deux coefficients congrus à* ± 3 *(ou* ± 5*).*
Dans ce cas, il y a donc $\binom{4}{2} = 6$ *possibilités pour les deux premiers coefficients.*
Et pour le dernier coefficient, il faut choisir parmi 15*,* 17 *et* 31*.*
Le dernier cas offre donc $2 \times 6 \times 3 = 36$ *quadruplets supplémentaires.*

On obtient au total $24+8+36=68$ *choix potentiels.*

Statistiquement, on obtient exactement le même résultat, ce qui confirme le lien entre les quadruplets.

6.3 Performance et Sécurité

Pour réaliser l'étude précédente, certains paramètres ont été fixés. Dans cette partie, on s'intéresse à à l'influence des paramètres sur la performance et la sécurité de l'algorithme.

Nous avons effectué des tests de performance afin de déterminer l'influence de cette méthode de randomisation de la représentation sur la vitesse d'exponentiation.

6.3.1 Influence du choix de coefficient sur les performances.

Voici les résultats des tests de comparaison en termes de performance et de vitesse entre les différents niveaux de sécurité de l'algorithme *DoubleRDR* :

TABLE 6.3 : Comparaison des niveaux de sécurité pour $\#\mathscr{D}=4$ et $|k|=4096$ (1000 tests)

	Temps d'execution	Double Zéros (nombre)	Double Zéros (en %)
$Double-wNAF$	1458	2859	69.8
DoubleRDR Niveau 2	1767	2926	71.4
DoubleRDR Niveau 3	1815	2829	69.1
DoubleRDR Niveau 4	1822	2766	67.5
DoubleRDR Niveau 5	1806	2647	64.6

On remarque que le temps d'execution des différents niveaux de l'algorithme est assez proche pour ces paramètres.
Plus la sécurité de l'algorithme augmente (du haut vers le bas du tableau), plus les performances en terme de $\binom{0}{0}$ diminuent.

Dans la partie suivante, nous testons l'influence de la randomisation de $\mathscr{D}$ sur les performances.

6.3.2 Influence de l'aléatoire sur les performances.

Voici les résultats des tests de comparaison entre le $Double-wNAF$ (pas de randomisation) et la *DoubleRDR* de niveau 1 :

On constate que la *DoubleRDR* est très coûteuse en temps, ce qui est un problème important pour l'exponentiation.

Il existe une solution pour réduire le temps d'execution, il s'agit de diminuer l'appel à la fonction *random*, ce qui permet de réduire le nombre de constructions du tableau *DoubleDmax*. Pour 1000 recodages, on peut se contenter de changer 10 fois l'ensemble $\mathscr{D}$, afin d'effectuer 100 tests avec le même ensemble $\mathscr{D}$.

Dans la partie suivante, nous testons l'influence de la taille de $\mathscr{D}$ sur les performances.

TABLE 6.4 : Temps d'execution (1000 tests)

#D	|k|	$DoubleRDRNiveau1$	$Double-wNAF$	perte (en %)
1	256	57	55	0
1	4096	1415	1439	0
4	256	353	177	99
4	4096	1709	1365	25
8	1024	4900	162	47765
16	256	68620	29	∞
16	4096	70303	1366	50466

6.3.3 Influence de la taille de $\mathscr{D}$ sur les performances.

Dans cette partie, on a testé l'influence de la taille de l'ensemble $\mathscr{D}$ sur le temps d'execution et les performances.
Voici les résultats :

TABLE 6.5 : Temps d'execution (1000 tests)

#D	|k|	Double Zéros (nombre)	$DoubleRDRNiveau1$	Double Zéros (en %)
1	256	127	57	50
1	4096	2047	2047	50
2	4096	2518	1382	61
4	4096	2926	1709	71
8	4096	3178	5991	78
16	4096	3340	70303	82

On observe que lorsque $\#\mathscr{D}$ devient grand, le temps d'execution croît très vite, il est important de choisir une taille de $\mathscr{D}$ adéquate afin d'avoir un bon rapport "temps d'execution/performance".

Il faut rappeler que lorsque la taille de $\mathscr{D}$ augmente, la sécurité de l'algorithme (du secret) augmente également.

— 7 —
Conclusion

Le but de cette étude est l'amélioration de la sécurité de l'algorithme d'exponentiation modulaire en utilisant la randomisation de l'exposant. La solution proposée a montré qu'on ne peut améliorer la sécurité de cet algorithme sans affecter les performances de celui-ci. La sécurité et la performance sont étroitement liées et l'important est de choisir les paramètres adéquats en fonction de ses besoins, dans le but d'optimiser l'utilisation des protocoles.

Le deuxième objectif de l'article est la généralisation de la méthode *RDR* pour la double exponentiation.

Le plus grand atout de la méthode explicitée dans cet article est l'adaptabilité du procédé. Nous offrons des possibilités variées pouvant répondre à tous les besoins en fonction des utilisateurs, ceci rend la solution intéressante.

Une étude sur la façon de précalculer les x^{d_i} et les $x^{d_i} \times y^{d_j}$ permettrait de compléter cet article. Ils existent plusieurs manières d'effectuer les précalculs, et le temps d'execution de l'exponentiation modulaire dépend grandement de l'efficacité du précalcul.
La difficulté du précalcul dépend également du protocole étudié, et plus particulièrement du groupe dans lequel les calculs sont effectués.

Il est également possible de généraliser le concept de *RDR* à la multi-exponentiation en général. Il serait intéressant d'étendre la méthode à 3, 4 ou même n exposants.
Plus le nombre d'exposants est grand, plus les paramètres de l'algorithme influent les performances du recodage.

Bibliographie