



UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

---

# Rapport explicatif des étapes du projet de sécurité

---

Promotion : 2024-2025

*Réalisé par :*

Dehbia KETEB

## Introduction

Dans le cadre du module de sécurité informatique du second quadrimestre, nous avons abordé plusieurs thématiques liées à la sécurité et à la cryptographie. Le projet présenté ici porte sur la construction et l'exploitation de **tables de Hellman**

L'objectif est de générer un ensemble de tables couvrant un espace de recherche fixé à  $2^{38}$  valeurs. Chaque table s'appuie sur une **fonction de réduction** qui ramène la sortie de SHA-256 (256 bits) vers une valeur sur 38 bits, afin d'enchaîner les transformations  $R(f(x))$  et de construire des chaînes. Une bonne conception de cette réduction et un parcours efficace du domaine permettent de **limiter les collisions**, de **maximiser la couverture** et d'**améliorer les chances de retrouver un mot de passe** à partir de son haché pendant la phase *online*.

## 1 Analyse du sujet et exigences

Le travail demandé correspond à la **phase de pré-calcul** des tables de Hellman. Un squelette de projet était fourni. Les exigences essentielles sont :

- **Jusqu'à 255 fonctions de réduction distinctes** : une table par valeur de rotation. Dans mon implémentation, la réduction effectue une *rotation logique à droite* du mot de 256 bits. Étant donné que je représente les bits en ordre LSB→MSB dans un vecteur, cela se traduit par une **rotation à gauche** du vecteur de 256 positions, paramétrée par le numéro de la table.
- **Extraction de 38 bits en little-endian** : après rotation, on conserve les **38 bits de poids faible** (ce qui correspond aux octets d'indices 0 à 4, plus les 6 bits faibles de l'octet d'indice 5, en représentation little-endian). Ces 38 bits sont empaquetés dans un entier sur au moins 6 octets (ici un u64).
- **Génération multi-tables avec métadonnées** : chaque table est écrite dans un fichier distinct (ex. 1.txt, 2.txt, ...) dont la première ligne contient nchains, ncolumns et la valeur de rotation redu (le numéro de la table).
- **Optimisation par multithreading** : la génération des tables est parallélisée avec `thread::spawn`, chaque table étant construite dans un *thread* indépendant. Aucun échange n'est requis entre threads ; une simple synchronisation finale par `join()` suffit.
- **Taille du programme** : le code (hors commentaires et lignes vides) respecte la contrainte d'environ une centaine de lignes.

## Algorithmes

L'objectif opérationnel est de développer une **fonction de réduction** fiable et de produire des **valeurs de départ sur 38 bits** servant de points initiaux de chaînes.

### Fonction `generated_aleatoire_passwords`

---

**Algorithm 1** Génération de mots de passe aléatoires de 38 bits

---

**Require:** `nchains` : entier strictement positif

**Ensure:** `result` : tableau de `nchains` entiers uniques de 38 bits

```

1: Initialiser un générateur rng
2: result  $\leftarrow$  tableau vide
3: while result.len < nchains do
4:   candidate  $\leftarrow$  entier aléatoire dans  $[0, 2^{38} - 1]$ 
5:   if candidate  $\notin$  result then
6:     Ajouter candidate à result
7: return result

```

---

### Fonction `reduction_function`

---

**Algorithm 2** Réduction d'un SHA-256 en un entier de 38 bits (LSB  $\rightarrow$  MSB, LE)

---

**Require:** `password` : tableau de 32 octets, `rotation` :  $0 \leq r < 256$

**Ensure:** `u64` contenant les 38 bits de poids faible après rotation

```

1: bits  $\leftarrow$  tableau vide
2: for chaque octet b de password do
3:   for j = 0 à 7 do                                      $\triangleright$  LSB  $\rightarrow$  MSB dans chaque octet
4:     Ajouter  $((b \gg j) \& 1)$  à bits
5: k  $\leftarrow r$                                               $\triangleright$  taille totale = 256
6: rotated  $\leftarrow$  tableau de 256 zéros
7: for i = 0 à 255 do                                        $\triangleright$  rotation à gauche du vecteur = rotation à droite du mot
8:   rotated[i]  $\leftarrow$  bits[(i + k) mod 256]
9: value  $\leftarrow$  0
10: for p = 0 à 37 do                                        $\triangleright$  empaquetage little-endian des 38 LSB
11:   value  $\leftarrow$  value |  $((\text{rotated}[p]) \ll p)$ 
12: return value

```

---

## Fonction main (génération parallèle des tables)

---

**Algorithm 3** Construction parallèle des tables de Hellman

---

```
1: Lire les arguments Args (ntables, nchains, ncolumns, path)
2: if path n'existe pas then
3:   Créer path
4: entries ← generated_aleatoire_passwords(nchains)
5: handles ← liste vide
6: for  $i = 1$  à ntables do
7:   Capturer path, entries, nchains, ncolumns, i
8:   Lancer un thread avec thread::spawn :
9:     Ouvrir path/i.txt, écrire l'en-tête (nchains, ncolumns, redu=i)
10:  for chaque start dans entries do
11:     $m \leftarrow \text{start}$ 
12:     $\text{hash} \leftarrow \text{SHA256}(\text{to\_le\_bytes}(m))$ 
13:     $m \leftarrow \text{reduction\_function}(\text{hash}, i)$ 
14:    for  $c = 1$  à ncolumns-1 do
15:       $\text{hash} \leftarrow \text{SHA256}(\text{to\_le\_bytes}(m))$ 
16:       $m \leftarrow \text{reduction\_function}(\text{hash}, i)$ 
17:    Écrire start et m dans le fichier
18:  Ajouter le handle à handles
19: for chaque h dans handles do
20:  h.join()
```

---

## Validation de la solution

J'ai privilégié une approche volontairement **simple et lisible** (boucles, structures de données de base) afin de faciliter la vérification et la maintenance. La rotation est réalisée *bit par bit* pour être correcte pour tout  $r$  (y compris lorsque  $r$  n'est pas multiple de 8). L'empaquetage des 38 bits suit strictement la **convention little-endian** : l'octet d'indice 0 (après rotation) correspond au poids le plus faible.

La génération des tables est parallélisée par **thread**, chaque table étant indépendante. La synchronisation se limite à un `join()` final après la création de toutes les tables.

## Résultats d'exécution

Les figures suivantes illustrent l'exécution et la production des fichiers de tables :

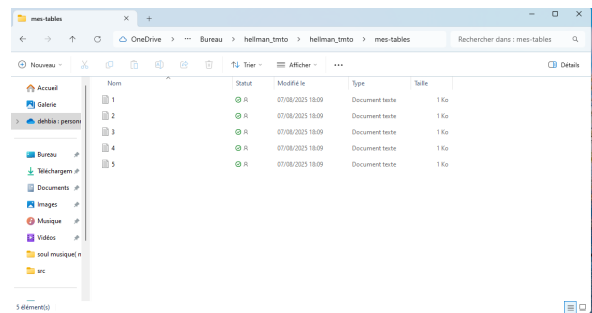
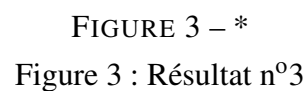
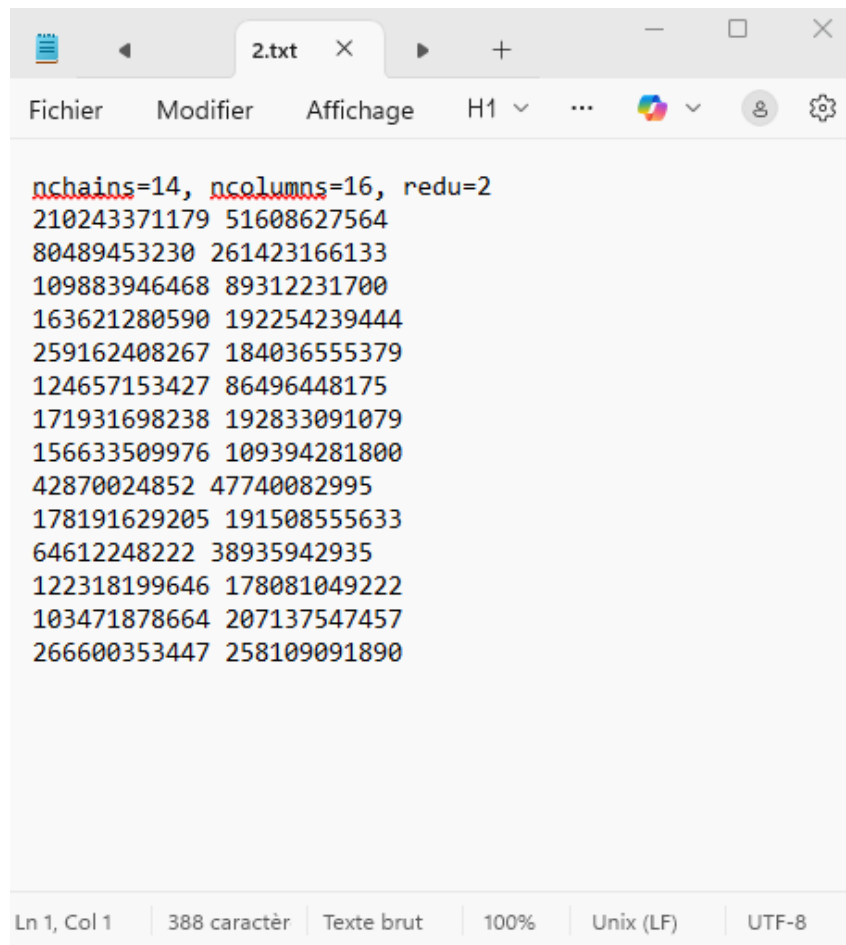


FIGURE 2 – \*

Figure 2 : Résultat n°2





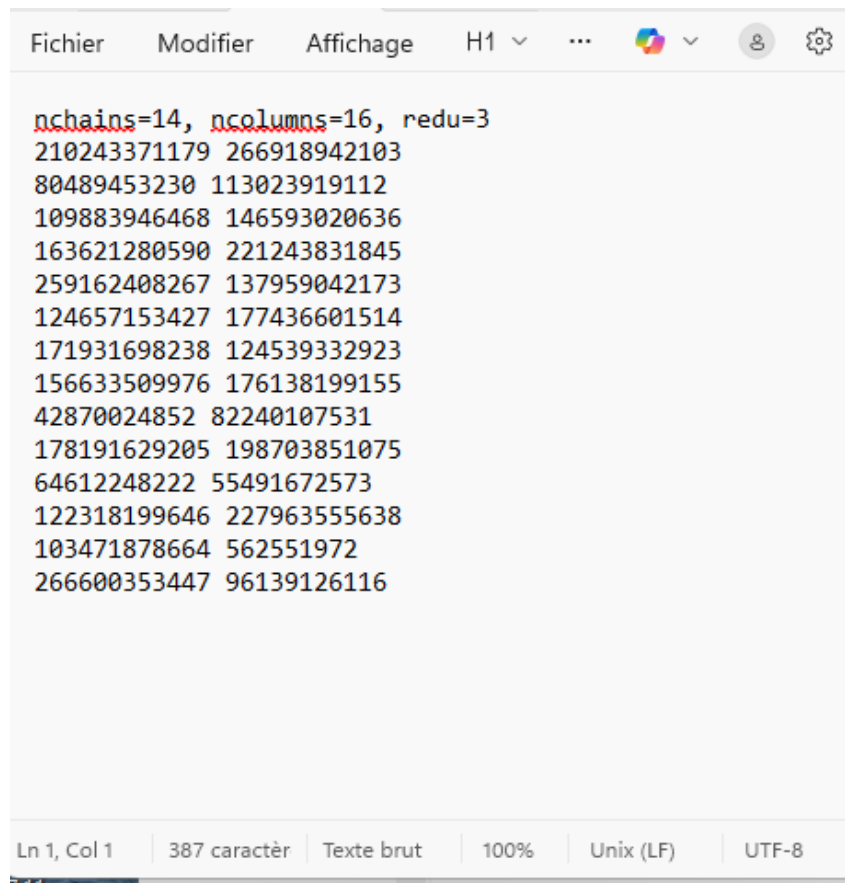
The screenshot shows a text editor window titled '2.txt'. The menu bar includes 'Fichier', 'Modifier', 'Affichage', 'H1', and several icons. The text content is as follows:

```
nchains=14, ncolums=16, redu=2
210243371179 51608627564
80489453230 261423166133
109883946468 89312231700
163621280590 192254239444
259162408267 184036555379
124657153427 86496448175
171931698238 192833091079
156633509976 109394281800
42870024852 47740082995
178191629205 191508555633
64612248222 38935942935
122318199646 178081049222
103471878664 207137547457
266600353447 258109091890
```

The status bar at the bottom indicates: 'Ln 1, Col 1', '388 caractères', 'Texte brut', '100%', 'Unix (LF)', and 'UTF-8'.

FIGURE 4 – \*

Figure 4 : Résultat n°4

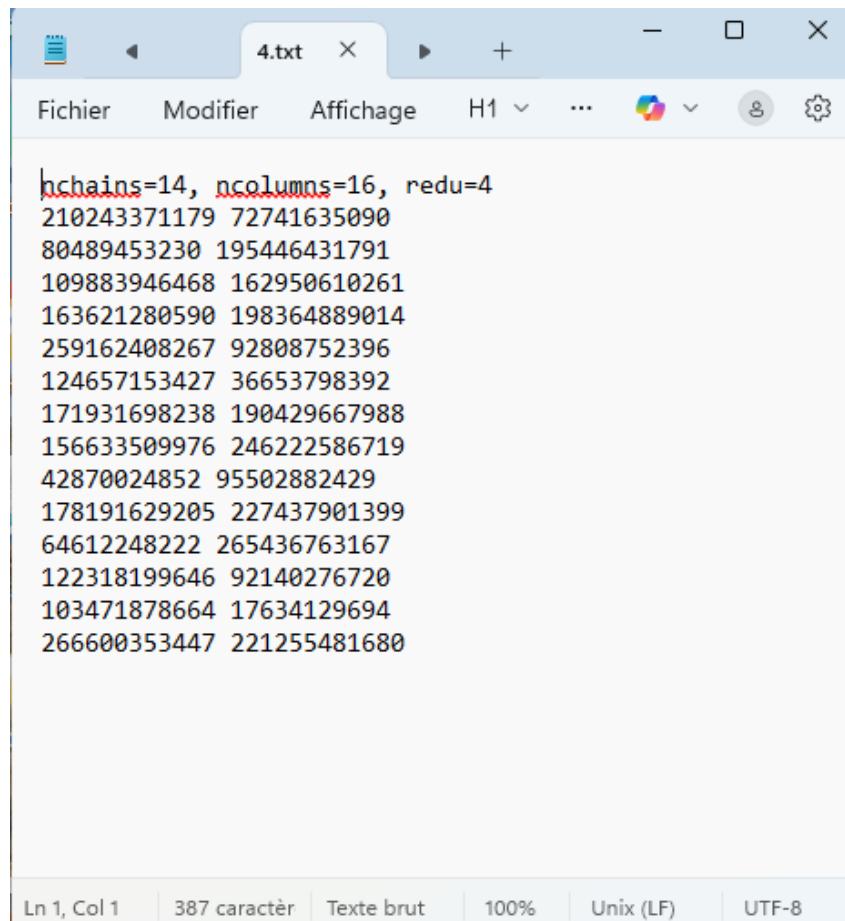


```
nchains=14, ncolums=16, redu=3
210243371179 266918942103
80489453230 113023919112
109883946468 146593020636
163621280590 221243831845
259162408267 137959042173
124657153427 177436601514
171931698238 124539332923
156633509976 176138199155
42870024852 82240107531
178191629205 198703851075
64612248222 55491672573
122318199646 227963555638
103471878664 562551972
266600353447 96139126116
```

Ln 1, Col 1 | 387 caractèr | Texte brut | 100% | Unix (LF) | UTF-8

FIGURE 5 – \*

Figure 5 : Résultat n°5

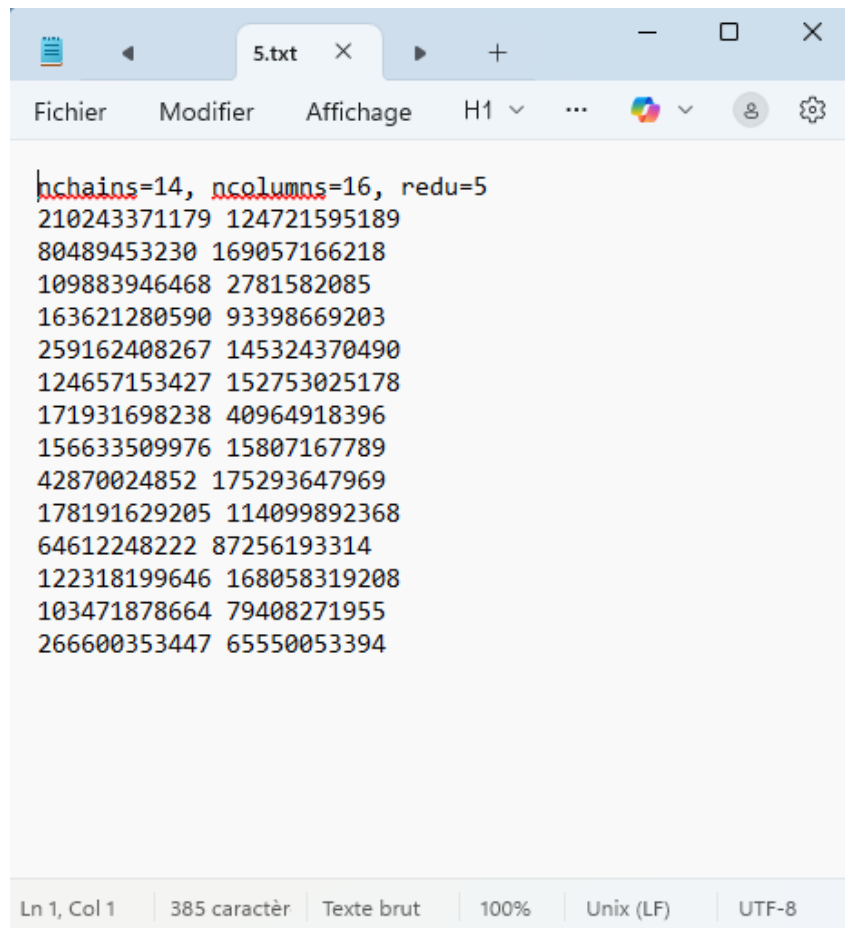


```
hchains=14, ncolumns=16, redu=4
210243371179 72741635090
80489453230 195446431791
109883946468 162950610261
163621280590 198364889014
259162408267 92808752396
124657153427 36653798392
171931698238 190429667988
156633509976 246222586719
42870024852 95502882429
178191629205 227437901399
64612248222 265436763167
122318199646 92140276720
103471878664 17634129694
266600353447 221255481680
```

FIGURE 6 – \*

Figure 6 : Résultat n°6





```
hchains=14, ncolums=16, redu=5
210243371179 124721595189
80489453230 169057166218
109883946468 2781582085
163621280590 93398669203
259162408267 145324370490
124657153427 152753025178
171931698238 40964918396
156633509976 15807167789
42870024852 175293647969
178191629205 114099892368
64612248222 87256193314
122318199646 168058319208
103471878664 79408271955
266600353447 65550053394
```

FIGURE 7 – \*

Figure 7 : Résultat n°7