SAÉ 3.01 Rapport de cryptographie

Jules CHIRON, Matis RODIER, Thomas GODINEAU | INF2 FI A

4 janvier 2024

Table des matières

Introduction

Le but de cette SAÉ est de réaliser une **application web** de ticketing en PHP. Chaque utilisateur de cette plateforme peut créer un compte et se connecter. Afin de sécuriser les mots de passe des utilisateurs, nous les chiffrons avec l'algorithme **RC4**.

La **première partie** de ce rapport contient le code PHP commenté des modules de chiffrement et de déchiffrement. La **deuxième partie** contient une recherche documentaire sur les fonctions de hachage cryptographique, l'algorithme MD5 et les applications de ces algorithmes en cryptographie.

1 Algorithme RC4

Permutation et suite chiffrante

Pour ce module, nous avons utilisés les algorithmes écrit en pseudo-code fournis dans le sujet. Il y a juste un changement dans la fonction *gen* (ligne 11).
Dans le cadre de ce projet, nous utilisons une clé générée aléatoirement de 40 caractères.

Fonction de génération de la permutation

```php
function ksa($k){
    // On crée un tableau de caractères à partir de la clé
    $k = str_split($k);

    for ($i = 0; $i < count($k); $i++){
        // On récupère le code ASCII de chaque caractère
        $k[$i] = intval(ord($k[$i]));
    }

    $s = array();
    for ($i = 0; $i < 256; $i++){
        // On crée un tableau de 0 à 255
        $s[] = $i;
    }

    $j = 0;
    for ($i = 0; $i < 256; $i++){
        // On mélange le tableau de manière à obtenir
        // une permutation des 256 valeurs
        $j = ($j + $s[$i] + $k[$i % count($k)]) % 256;
        $temp = $s[$i];
        $s[$i] = $s[$j];
        $s[$j] = $temp;
    }

    return $s;
}
```

Fonction de génération de la suite chiffrante

```php
function gen($s, $n){
    $j = 0;
    $k = array();

    for ($i = 0; $i < $n; $i++){
        // On initialise le tableau de sortie avec n 0
        $k[] = 0;
    }

    $i = 0;
    for ($l = 0; $l < $n; $l++){ // /!\ Changement dans le sujet
        $i = ($i + 1) % 256;
        // On récupère la valeur du tableau à l'indice i
        // et on l'ajoute à j
        $j = ($j + $s[$i]) % 256;

        // On échange les valeurs de s[i] et s[j]
        $temp = $s[$i];
        $s[$i] = $s[$j];
        $s[$j] = $temp;

```

```
22          // On modifie la valeur de k à l'indice l
23          // avec la valeur de s à l'indice (s[i] + s[j]) % 256
24          $k[$l] = $s[($s[$i] + $s[$j]) % 256];
25      }
26
27      return $k;
28  }
```

Chiffrement et déchiffrement

Fonction de chiffrement

```
1   function cypher($m, $k){
2       $m = str_split($m);
3
4       // On génère la suite chiffrante
5       $s = gen(ksa($k), 128);
6
7       for ($i = 0; $i < count($m); $i++){
8           // On récupère le code ASCII de chaque caractère du message
9           $m[$i] = intval(ord($m[$i]));
10      }
11
12      $c = array();
13      for ($i = 0; $i < count($m); $i++){
14          // On effectue un XOR entre le message et la suite
15
16          // On convertit le résultat en hexadécimal
17          // et on l'ajoute au tableau de sortie
18          if (strlen(dechex($m[$i] ^ $s[$i])) == 2)
19              $c[] = dechex($m[$i] ^ $s[$i]);
20          else
21              // Si le code ASCII est inférieur à 16,
22              // on ajoute un 0 devant le résultat
23              $c[] = '0'.dechex($m[$i] ^ $s[$i]);
24      }
25
26      // On retourne le résultat sous forme de chaîne de caractères
27      return implode('', $c);
28  }
```

Fonction de déchiffrement

```
1   function decypher($c, $k){
2       $c = str_split($c, 2);
3       $s = gen(ksa($k), 128);
4
5       for ($i = 0; $i < count($c); $i++){
6           // On convertit chaque caractère hexadécimal en décimal
7           $c[$i] = hexdec($c[$i]);
8       }
```

```
9
10      $m = array();
11      for ($i = 0; $i < count($c); $i++){
12          // On effectue un XOR entre le message chiffré et la suite
13          $m[] = chr($c[$i] ^ $s[$i]);
14      }
15
16      return implode('', $m);
17  }
```

2 Travail de recherche

2.1 Fonction de hachage cryptographique

Définition

Une fonction de hachage cryptographique est une fonction qui associe à un message d'entrée (chaîne de caractères, entier ...) une valeur bien précise appelée "empreinte".

Propriétés

Une fonction de hachage doit disposer de certaines propriétés afin d'être suffisamment sûre et efficace. Premièrement, la taille des empreintes doit être **indépendante de la longueur** du message. Deuxièmement, une fonction de hachage doit renvoyer **une seule et unique empreinte** pour chaque message possible.

À l'inverse, elle doit renvoyer **deux empreintes différentes** pour deux messages différents. Si ce n'est pas le cas, il y a alors des **collisions** et l'algorithme n'est plus sûr car deux messages différents peuvent être utilisés alors qu'un seule est le bon (s'ils ont la même empreinte). Si ces messages ne diffèrent même que très peu, les empreintes des deux messages doivent être **très différentes** afin de ne pas pouvoir s'approcher de plus en plus du message en essayant de décrypter une empreinte. Enfin, il ne doit **pas être possible de retrouver un message** à partir de son empreinte (du moins sans la clé).

2.2 Chiffrement MD5

Présentation

La **fonction de hachage MD5** est une fonction de hachage cryptographique. MD5 est l'acronyme de Message Digest 5 (équivalent d'*empreinte* en français et *5* car il succède au **MD4**). Cette fonction a été inventée par **Ronald Rivest** (un des inventeurs du *RSA*) en 1991, c'est l'amélioration de l'algorithme **MD4**. Cependant, de graves failles de sécurité ont été découvertes en 1996, et des collisions ont été trouvées en 2004 par une équipe chinoise. Ainsi, l'**algorithme MD5** n'est plus considéré comme sûr et est déconseillé pour des applications cryptographiques. D'autres algorithmes plus récents et plus sûrs ont été créés et sont recommandés pour des applications dans ce domaine, comme **SHA-256**. Cependant, MD5 est encore utilisé pour vérifier l'intégrité de fichiers (pour vérifier qu'ils n'ont pas été modifiés) lors de téléchargements.

Fonctionnement

L'empreinte de l'**algorithme MD5** est une chaîne de 128 bits (16 octets donc 16 caractères). La **fonction MD5** travaille sur des blocs de 512 bits (soit 64 caractères). Si la longueur du dernier bloc du message n'est pas un multiple de 448 bits, il est complété :

— On ajoute un 1 à la fin du message

— On ajoute des 0 jusqu'à ce que la longueur du dernier bloc soit égale à 448 bits

On ajoute ensuite la taille du message (**sur 64 bits**) à la fin de celui-ci. La longueur du message est donc maintenant un multiple de 512 bits ($448 + 64 = 512$). Le message est donc ensuite découpé en bloc de 512 bits.

Suite à cela, on calcule 64 constantes de 32 bits (*64 mots*) que l'on stocke dans un tableau à partir de la fonction sinus :

$$\forall i \in [1, 64],\ K[i] = \lfloor 2^{32} \times |\sin(i)| \rfloor$$

On définit également un tableau de 64 cases contenant des valeurs qui seront utilisées pour effectuer des rotations de bits :

$$S = [[7, 12, 17, 22] \times 4, [5, 9, 14, 20] \times 4, [4, 11, 16, 23] \times 4, [6, 10, 15, 21] \times 4]$$

On choisit 4 valeurs arbitraires de 32 bits (L, M, N et O) et on définit les 4 fonctions suivantes :

— $F(X, Y, Z) = (X \wedge Y) \vee (\bar{X} \wedge Z)$

— $G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \bar{Z})$

— $H(X, Y, Z) = X \oplus Y \oplus Z$

— $I(X, Y, Z) = Y \oplus (X \vee \bar{Z})$

En francais cela donne :

— $F(X, Y, Z) = (X\ ET\ Y)\ OU\ (NON\ X\ ET\ Z)$

— $G(X, Y, Z) = (X\ ET\ Z)\ OU\ (Y\ ET\ NON\ Z)$

— $H(X, Y, Z) = X\ XOR\ Y\ XOR\ Z$

— $I(X, Y, Z) = Y\ XOR\ (X\ OU\ NON\ Z)$

Puis, **chaque bloc** est traité de la manière suivante :

— On initialise 4 variables de 32 bits (A, B, C et D) avec les valeurs initiales

— On effectue 64 tours de boucle (on effectue 4×16 opérations) avec les fonctions F, G, H et I (fonctions F pour les 16 premières, puis G pour les 16 suivantes ...) (voir Figure 1) :

- On applique la fonction (*F, G, H ou I*) aux variables B, C et D
- On ajoute le résultat de la fonction à la variable A
- On ajoute aussi à A la valeur du message à l'indice i (le tour de boucle) et $K[i]$
- On effecture une rotation de A de $S[i]$ bits vers la gauche
- On ajoute B à A
- On applique modulo 2^{32} à A

- Enfin, On effectue une rotation des variables A, B, C et D ($A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D,\ D \rightarrow A$)

— On ajoute les valeurs des variables A, B, C et D aux valeurs initiales ($L = L + A$, $M = M + B$, ...)

— On recommence pour chaque bloc

Enfin, on concatène L, M, N et O pour obtenir **l'empreinte finale du message** ($4 \times 32 = 128$ *bits*).

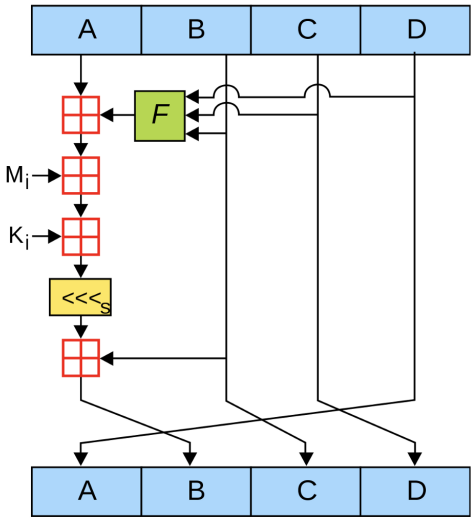

FIGURE 1 – Représentation d'une des 64 itérations de l'algorithme MD5

2.3 Utilisation dans la cryptographie

Les **fonctions de hachage** telles que **MD5** trouvent un intérêt dans la cryptographie. Effectivement, ces fonctions transforme un **message** (chaîne de caractères, entier ...) en une **empreinte** (chaîne de caractères représentant une suite hexadécimale) dont la taille ne dépend pas de la taille du message.
De plus, on peut observer que si on modifie un caractère du message, l'empreinte finale change complètement. On ne peut donc pas déchiffrer un message à partir de son empreinte (*à tatillon*).
Enfin, (*pour l'algorithme MD5*) il existe théoriquement **plus de 3×10^{38}** empreintes différentes pouvant être possiblement générées ($2^{128} = 3,402823669 \times 10^{38}$). Il est donc très peu probable de trouver deux messages différents ayant la même empreinte (*outre les collisions découvertes en 2004 (cf 2.2)*).

Ainsi, des fonctions telles que **MD5** ou encore **SHA-256** vérifient les différentes propriétés que nous avons énoncées dans la partie 2.1. C'est pourquoi de telles fonctions sont utilisées dans la cryptographie.