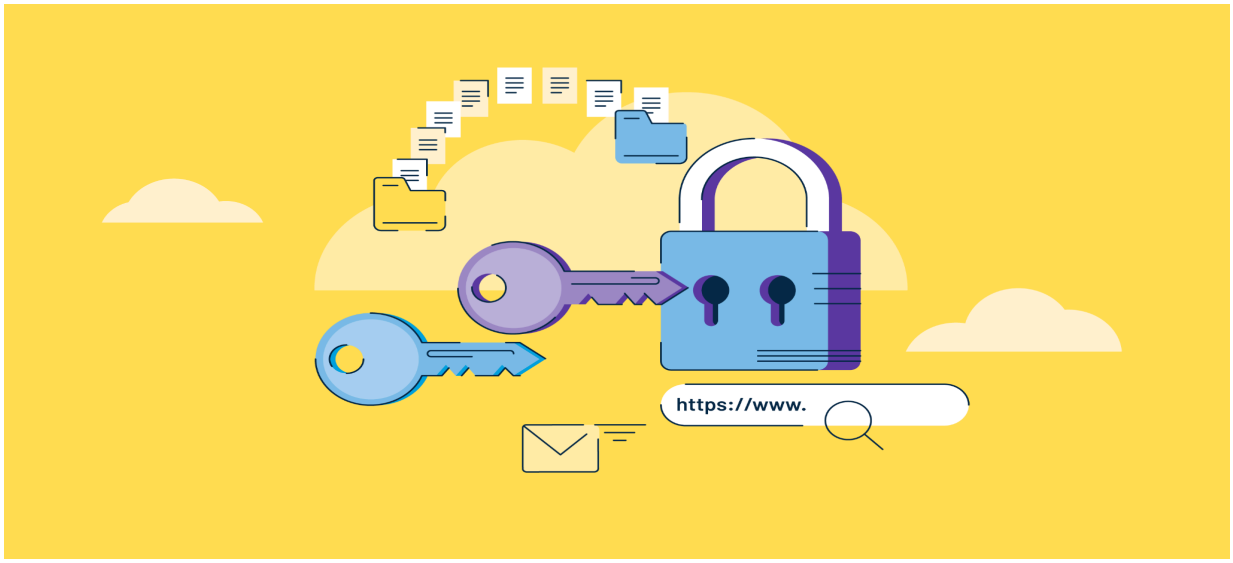
Cryptographie avancé

Compte Rendu

ALGORITHME MINI-DES



Année Universitaire: 2023/2024

LEKOUAGHET imad eddine

Génération des deux clé :

On utilisera dans ce le code deux fonctions définie comme suit:

Fonction déplacement:

```
def deplacement(bits, deplacement):

    return bits[deplacement:] + bits[:deplacement]
```

La fonction "deplacement" effectue une rotation circulaire des bits d'un bloc de données vers la gauche ou vers la droite, selon la valeur de "deplacement". Elle prend les "deplacement" premiers bits du bloc et les place à la fin, tandis que les bits restants sont déplacés au début, créant ainsi un décalage circulaire.

Fonction permutation clé:

```
def permutation_cle(cle, table_de_permutation):

    cle_permutee = [cle[index - 1] for index in table_de_permutation]

    return ''.join(cle_permutee)
```

La fonction "permutation_cle" réalise une permutation des bits de la clé selon une table de permutation spécifique. Pour chaque index dans la table de permutation, elle récupère le bit correspondant de la clé et les concatène pour former la clé permutée. Enfin, elle retourne cette clé permutée.

```
cle_0 = "1010000010"

table_P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]

table_P8 = [6, 3, 7, 4, 8, 5, 10, 9]
```

Table de permutation P10 :

Cette table est utilisée pour réorganiser les bits de la clé initiale de 10 bits, "cle_0".

Chaque index dans la table P10 indique la position du bit correspondant dans la clé initiale.

Par exemple, le premier élément de la table P10 est 3, ce qui signifie que le bit à la position 3 dans la clé initiale sera le premier bit de la clé permutée.

En utilisant cette table, la clé de départ "cle_0" est permutée pour former une nouvelle clé de 10 bits.

Table de permutation P8 :

Cette table est utilisée pour réduire la taille de la clé permutée de 10 bits à 8 bits.

Chaque index dans la table P8 indique la position du bit correspondant dans la clé permutée de 10 bits.

Par exemple, le premier élément de la table P8 est 6, ce qui signifie que le sixième bit dans la clé permutée de 10 bits sera le premier bit de la clé finale de 8 bits.

En utilisant cette table, la clé permutée de 10 bits est encore une fois réorganisée pour former une clé finale de 8 bits.

```
cle_1 = permutation_cle(cle_0, table_P10)

L = cle_1[:5]

R = cle_1[5:]
```

Dans ces lignes de code, nous utilisons la fonction permutation_cle() pour permuter la clé cle_0 en utilisant la table de permutation P10. Voici ce qui se passe ensuite :

La fonction permutation_cle(cle_0, table_P10) est appelée, prenant la clé cle_0 (qui est initialement de 10 bits) et la table de permutation P10 en argument.

La fonction réorganise les bits de la clé cle_0 selon la table de permutation P10, produisant ainsi une nouvelle clé de 10 bits, que nous appelons cle_1.

Après la permutation, nous divisons la clé permutée cle_1 en deux parties : la partie gauche (L) et la partie droite (R).

Les 5 premiers bits de la clé permutée cle_1 constituent la partie gauche L, tandis que les 5 derniers bits constituent la partie droite R.

Ainsi, L représente les 5 premiers bits de la clé permutée cle_1, et R représente les 5 derniers bits. Ces parties de la clé sont utilisées plus tard dans l'algorithme de chiffrement DES pour différentes opérations.

```
L_decale = deplacement(L, 1)

R_decale = deplacement(R, 1)

cle = L_decale + R_decale
```

Dans ces lignes de code, nous effectuons un décalage circulaire vers la gauche des parties gauche (L) et droite (R) de la clé.

Nous appelons la fonction deplacement(L, 1) pour déplacer la partie gauche (L) de la clé cle_1 d'une position vers la gauche.

De même, nous appelons la fonction deplacement(R, 1) pour déplacer la partie droite (R) de la clé cle_1 d'une position vers la gauche.

Ces décalages circulaires sont effectués pour chaque partie de la clé afin de créer une nouvelle clé décalée.

Enfin, nous concaténons les parties gauche et droite décalées pour former une nouvelle clé, que nous stockons dans la variable cle.

Ces décalages circulaires sont une étape importante dans l'algorithme de génération des sous-clés du chiffrement DES, où les parties gauche et droite de la clé sont déplacées pour créer une nouvelle clé décalée.

```
cle_1 = permutation_cle(cle, table_P8)

L_decale = deplacement(L, 3)

R_decale = deplacement(R, 3)

cle = L_decale + R_decale



cle_2 = permutation_cle(cle, table_P8)
```

On réalise la meme chose avec la clé_2.

Compilation:

Aprés la compilation du code on obtient les deux clés comme suit:

Encryption du MINI-DES:

```
def lookup_Sbox(bits, Sbox):

    row = int(bits[0] + bits[3], 2)

    col = int(bits[1:3], 2)

    return Sbox[row][col]
```

Cette fonction est utilisée pour effectuer une recherche dans une boîte de permutation (S-box) en utilisant une partie spécifique du bloc de données en entrée. Voici comment elle fonctionne :

Extraction des indices de ligne et de colonne :

La fonction prend en entrée une partie spécifique de bits bits du bloc de données et une S-box Sbox.

Elle extrait les deux premiers bits (bits[0] et bits[3]) pour déterminer l'indice de la ligne dans la S-box.

Elle convertit ces deux bits en un entier en utilisant la fonction int(bits[0] + bits[3], 2). Ici, le 2 indique que la chaîne binaire est convertie en entier.

Extraction de l'indice de colonne :

Les bits du milieu (bits[1:3]) sont extraits pour déterminer l'indice de colonne dans la S-box.

Ils sont également convertis en un entier en utilisant int(bits[1:3], 2).

Recherche dans la S-box :

Avec les indices de ligne et de colonne extraits, la fonction accède à la S-box en utilisant ces indices.

Elle récupère la valeur stockée dans la case correspondante de la S-box.

La valeur récupérée est retournée en tant que résultat de la fonction.

```
def fonction1(texte, cle):
    table_EP = [4, 1, 2, 3, 2, 3, 4, 1]
    table_P4 = [2, 3, 1, 4]
    SBOX0 = [
        ['01', '00', '11', '10'],
        ['11', '10', '01', '00'],
        ['00', '10', '01', '11'],
        ['11', '01', '11', '10']
    ]

    SBOX1 = [
        ['00', '10', '10', '11'],
        ['10', '00', '01', '11'],
```

```
        ['11', '00', '10', '00'],
        ['10', '01', '00', '11']
    ]
```

table_EP : Table d'expansion, utilisée pour étendre la partie droite du bloc de données à 4 bits à 8 bits.

table_P4 : Table de permutation à 4 bits, utilisée pour permuter les résultats des boîtes S avant l'opération de XOR.

SBOX0/SBOX1 : Deux boîtes de permutation (S-boxes), utilisées pour effectuer des substitutions non linéaires

```
    L0 = texte[:4]
    R0 = texte[4:]
```

Le bloc de données d'entrée est divisé en deux parties, L et R.

```
R_expanded = permutation(R0, table_EP)
resultat_xor = xor(R_expanded, cle)
```

Expansion de R :

La partie droite R est étendue de 4 bits à 8 bits en utilisant la table d'expansion table_EP.

Opération XOR avec la clé :

Le résultat de l'expansion de R est combiné avec la clé en utilisant une opération XOR.

```
    resultat_S0 = lookup_Sbox(resultat_xor[:4], SBOX0)
    resultat_S1 = lookup_Sbox(resultat_xor[4:], SBOX1)
```

En résumé, ces lignes de code effectuent des substitutions non linéaires en utilisant deux boîtes de permutation (S-boxes) distinctes, SBOX0 et SBOX1, sur les parties gauche et droite du résultat de l'opération XOR, respectivement. Les valeurs substituées sont ensuite

utilisées dans les étapes suivantes du chiffrement DES pour former le bloc de données chiffré final.

```
S_combine = resultat_S0 + resultat_S1

   resultat_P4 = permutation(S_combine, table_P4)

   R1 = xor(resultat_P4, L0)

   SW = R0 + R1
```

Concaténation des résultats des boîtes S :

Les résultats des substitutions dans les boîtes S, resultat_S0 et resultat_S1, sont concaténés pour former une chaîne de 4 bits S_combine. Cela est nécessaire car les résultats des boîtes S sont de longueur différente, mais ils doivent être combinés pour être utilisés dans la permutation suivante.

Permutation à l'aide de la table P4 :

La chaîne de 4 bits S_combine est permutée à l'aide de la table de permutation P4, table_P4. Cette étape réorganise les bits de S_combine selon la table de permutation P4 pour obtenir resultat_P4. Cette permutation est importante pour introduire une certaine confusion dans les données chiffrées.

Opération XOR avec L0 pour obtenir R1 :

L'opération XOR est effectuée entre resultat_P4 et la partie gauche L0 du bloc de données d'entrée. Cela introduit une autre transformation non linéaire dans le processus de chiffrement.

Le résultat de cette opération XOR donne la partie droite R1 du bloc de données chiffré.

Formation du bloc de données chiffré final (SW) :

Le bloc de données chiffré final est formé en concaténant la partie droite d'origine R0 et la nouvelle partie droite R1. Cela complète le processus de chiffrement du bloc de données

Compilation du code :

Initialisation du message texte et des clés :

Le message texte à chiffrer est défini comme "01110010".

Deux clés sont également définies : cle_1 comme "10100100" et cle_2 comme "01000011".

Permutation initiale (PI) du message texte :

Le message texte est permuté initialement en utilisant la table de permutation table_PI. Cela réorganise les bits du message texte selon un schéma prédéfini.

Application de la fonction de chiffrement DES :

La fonction fonction1() est appelée avec le message texte permuté et la première clé cle_1. Cela chiffre le message texte avec la première clé.

Le résultat est stocké dans la variable SW.

Chiffrement avec la deuxième clé :

La fonction fonction1() est à nouveau appelée, cette fois avec le résultat chiffré précédent SW et la deuxième clé cle_2. Cela chiffre à nouveau le message texte, mais cette fois-ci avec une deuxième clé.

Le résultat est stocké dans la variable C.

Permutation finale (IP) du texte chiffré :

Le texte chiffré est permuté une dernière fois en utilisant la table de permutation table_IP. Cela réorganise les bits du texte chiffré dans un ordre spécifique pour obtenir le texte chiffré final.

Affichage du texte chiffré final :

```
message crypté est:  11011101
```