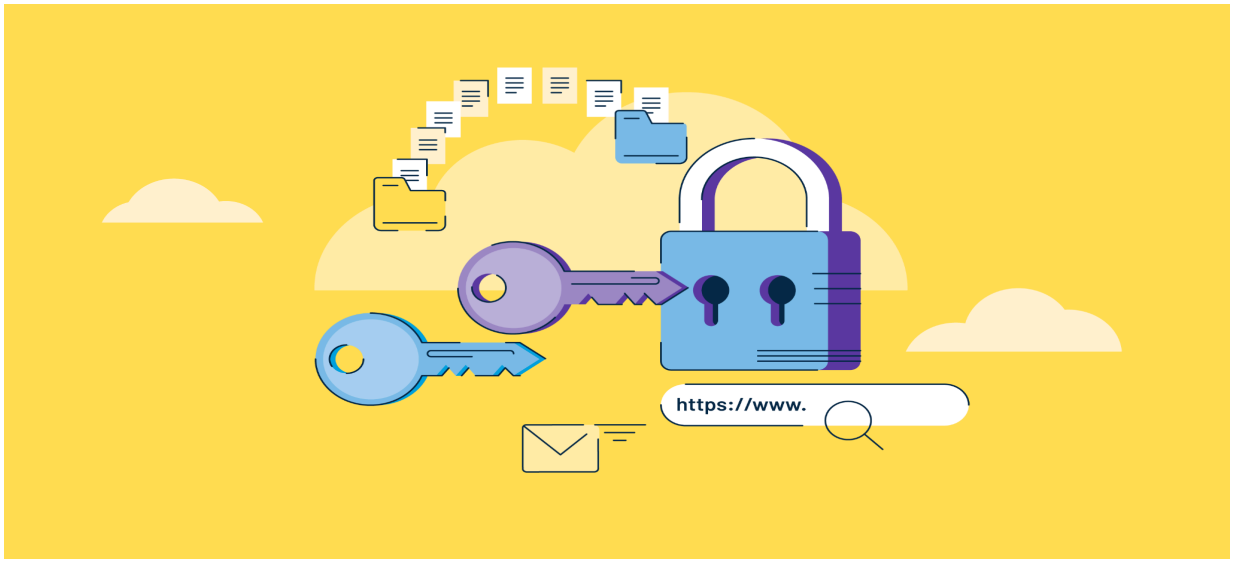
Cryptographie avancé

Compte Rendu
ALGORITHME MINI-AES



Année Universitaire: 2023/2024

LEKOUAGHET imad eddine

Génération des deux clé :

Le code présent propose une implémentation d'une méthode de génération de clés symétriques basée sur des opérations XOR et une table de substitution (S-box). Plus spécifiquement, il présente deux fonctions, calculate_K1 et calculate_K2, qui génèrent respectivement les clés K1 et K2 à partir d'une clé initiale K0. Ce rapport analyse en détail le fonctionnement de la fonction calculate_K1, décomposant chaque étape de son processus de génération de clé et discutant de son rôle dans un système de chiffrement.

Fonction XOR et S-box :

La fonction xor_binary réalise une opération XOR bit à bit entre deux chaînes binaires de même longueur.

s_box est une table de substitution (S-box) en format binaire. Elle est utilisée pour substituer des valeurs lors du chiffrement.

```
def xor_binary(binary1, binary2):

    result = ''.join(str(int(bit1) ^ int(bit2)) for bit1, bit2 in zip(binary1, binary2))
    return result

s_box = {
    '0000': '1110', '0001': '0100', '0010': '1101', '0011': '0001',
    '0100': '0010', '0101': '1111', '0110': '1011', '0111': '1000',
    '1000': '0011', '1001': '1010', '1010': '0110', '1011': '1100',
    '1100': '0101', '1101': '1001', '1110': '0000', '1111': '0111'
}
```

Fonction split_key :est chargée de diviser une clé binaire en plusieurs mots de 4 bits

```
def split_key(K0):
    # Split K0 into 4-bit words
    w0 = K0[:4]
    w1 = K0[4:8]
    w2 = K0[8:12]
    w3 = K0[12:]

    return w0, w1, w2, w3
```

La fonction prend en entrée une clé binaire K0. Cette derniére est divisée en quatre partieségales, chaque partie représentant un mot de 4 bits.Les quatre mots sont

déterminés en prenant successivement les 4 premiers, les 4 suivants, etc., jusqu'à la fin de la chaîne.

Fonction calculate_K1 :

La fonction calculate_K1 joue un rôle fondamental dans le processus de génération de clés symétriques en chiffrement. Initialement, la clé binaire K0 est subdivisée en quatre mots de 4 bits, notés w0, w1, w2, et w3. L'étape suivante implique une opération XOR entre le premier mot w0 et la constante binaire 0001. Cette opération modifie le premier mot pour introduire une variation cruciale dans la clé.

```
def calculate_K1(K0):
    # Split K0 into 4-bit words
    w0, w1, w2, w3 = split_key(K0)

    # XOR the first word with 0001
    w0 = xor_binary(w0, '0001')

    # Calculate w4 to w7 for K1
    w4 = xor_binary(w0, s_box[w3])
    w5 = xor_binary(w1, w4)
    w6 = xor_binary(w2, w5)
    w7 = xor_binary(w3, w6)

    # Combine w4 to w7 into K1
    K1 = w4 + w5 + w6 + w7


    return K1
```

Par la suite, la fonction exploite une table de substitution, la S-box, pour transformer le dernier mot w3. En utilisant w3 comme indice dans la S-box, la valeur correspondante est extraite et XORée avec le mot modifié précédemment (w0 XOR 0001). Le résultat de cette opération, noté w4, devient le deuxième mot de la nouvelle clé K1.

Les étapes suivantes de la fonction calculent les mots restants de la clé K1 (w5, w6, w7) en effectuant des opérations XOR successives entre les mots précédents. Enfin, ces mots sont assemblés pour former la clé K1 complète.

Fonction calculate_K2 :est une extension de calculate_K1, générant une deuxième clé K2 à partir de la première clé K1. Elle divise d'abord la clé K1 en mots de 4 bits, puis effectue

des opérations XOR et utilise une S-box pour chaque mot afin de produire K2. Cette approche garantit que K2 est une clé unique et sécurisée, prête à être utilisée dans le processus de chiffrement.

Algorithme d'encryption Mini-AES :

Ce code implémente une version simplifiée d'un chiffrement symétrique basé sur l'algorithme AES (Advanced Encryption Standard). L'AES est un algorithme de chiffrement largement utilisé pour sécuriser les données sensibles dans diverses applications.

xor_binary(binary1, binary2) :

Cette fonction prend deux chaînes binaires en entrée (binary1 et binary2) et effectue une opération XOR bit à bit entre elles.Elle itère sur chaque bit des deux chaînes binaires et effectue l'opération XOR correspondante.Le résultat est une chaîne binaire représentant le résultat de l'opération XOR.

```
def xor_binary(binary1, binary2):
    result = ''.join(str(int(bit1) ^ int(bit2)) for bit1, bit2 in zip(binary1, binary2))
    return result
```

multiply_binary(bin1, bin2) :

Cette fonction prend deux chaînes binaires (bin1 et bin2) en entrée.Elle convertit ces chaînes binaires en entiers, effectue une multiplication entre ces entiers, puis convertit le résultat en une chaîne binaire.Le résultat est une chaîne binaire représentant le produit des deux nombres binaires en entrée.

```
def multiply_binary(bin1, bin2):
    # Convert binary strings to integers
    num1 = int(bin1, 2)
    num2 = int(bin2, 2)

    # Perform multiplication in decimal
    result_decimal = num1 * num2

    # Convert the result back to binary string
    result_binary = str(bin(result_decimal))[2:]

    return result_binary
```

split_message(M) :

Cette fonction prend une chaîne binaire M en entrée.Elle divise cette chaîne en quatre parties égales, chaque partie représentant un mot de 4 bits.Le résultat est un tuple contenant les quatre mots de 4 bits.

```
def split_message(M):
    P0 = M[:4]
    P1 = M[4:8]
    P2 = M[8:12]
    P3 = M[12:]

    return P0, P1, P2, P3
```

substitution(P) :

Cette fonction prend un mot de 4 bits P en entrée.Elle utilise ce mot comme clé pour accéder à une table de substitution appelée S-box, où chaque entrée est associée à une autre valeur de 4 bits.Le résultat est la valeur correspondante à la clé P dans la S-box.

```
def substitution(P):
    B = s_box[P]

    return B
```

create_matrix(M) :

Cette fonction prend une chaîne binaire M en entrée.Elle utilise la fonction split_message pour diviser M en quatre mots de 4 bits.Pour chaque mot, elle applique la substitution de bytes à l'aide de la fonction substitution.Le résultat est une matrice 2x2 contenant les valeurs substituées pour chaque mot.

```
def create_matrix(M):
    P0, P1, P2, P3 = split_message(M)
    B0 = substitution(P0)
    B1 = substitution(P1)
    B2 = substitution(P2)
    B3 = substitution(P3)
    matrix = [[B0, B2], [B1, B3]]

    return matrix
```

create_matrixk(M) :Cette fonction est similaire à create_matrix, mais elle ne fait que diviser M en mots de 4 bits sans appliquer la substitution de bytes.Elle est utilisée pour créer la matrice de clé de tour.

```
def create_matrixk(M):
    P0, P1, P2, P3 = split_message(M)

    matrix = [[P0, P2], [P1, P3]]

    return matrix
```

shiftRows(matrix) :

Cette fonction effectue un décalage circulaire des éléments de la deuxième ligne d'une matrice 2x2. Les éléments sont déplacés d'une position vers la gauche, et le résultat est renvoyé en sortie.

```
def shiftRows(matrix):
    matrix[1][0], matrix[1][1] = matrix[1][1], matrix[1][0]

    return matrix
```

mixColumns(matrix) : Cette fonction réalise l'opération MixColumns de l'algorithme AES sur une matrice 2x2. Elle combine chaque colonne de la matrice avec une matrice constante spécifique à l'aide d'opérations de multiplication binaire et de XOR. Le résultat est renvoyé en sortie.

```python
def mixColumns(matrix):

    matrix_constant = [
        ['11', '10'],
        ['10', '11']
    ]

    # Perform MixColumns operation
    D0= xor_binary(multiply_binary(matrix_constant[0][0], matrix[0][0]), multiply_binary(matrix_constant[0][1], matrix[1][0]))
    D1= xor_binary(multiply_binary(matrix_constant[0][0], matrix[0][1]), multiply_binary(matrix_constant[0][1], matrix[1][1]))
    D2= xor_binary(multiply_binary(matrix_constant[1][0], matrix[0][0]), multiply_binary(matrix_constant[1][1], matrix[1][0]))
    D3= xor_binary(multiply_binary(matrix_constant[1][0], matrix[0][1]), multiply_binary(matrix_constant[1][1], matrix[1][1]))
    result= [[D0, D2], [D1, D3]]
    return result
```

addRoundKey(matrix, key) : Cette fonction combine une matrice 2x2 avec une matrice de clé de tour en effectuant une opération XOR entre chaque élément correspondant des deux matrices. Le résultat est renvoyé en sortie.

```python
def addRoundKey(matrix, key):
    # Perform AddRoundKey operation
    result = [[xor_binary(matrix[row][col], key[row][col]) for col in range(2)] for row in range(2)]

    return result
```

matrixToString(matrix) : Cette fonction convertit une matrice 2x2 en une chaîne binaire en concaténant les éléments de la matrice. Le résultat est renvoyé sous forme de chaîne binaire représentant la matrice en texte.

```python
def matrixToString(matrix):
    return ''.join(matrix[0]) + ''.join(matrix[1])
```

L'exécution du code :

```
M_binary = '1100101010111101'
print('text en claire ',M_binary)
matrix = create_matrix(M_binary)
shifted_matrix = shiftRows(matrix)
mixed_matrix = mixColumns(shifted_matrix)
K1="1100111000001010"
result = matrixToString(addRoundKey(mixed_matrix, create_matrixk(K1)))
print('apres round key 1',result)
matrix = create_matrix(result)
shifted_matrix = shiftRows(matrix)
mixed_matrix = mixColumns(shifted_matrix)
K2="1000011001101100"
result = addRoundKey(mixed_matrix, create_matrixk(K2))
print('le text chiffré est ' ,matrixToString(result))
```

Ce script chiffre un message binaire en deux tours en utilisant une version simplifiée de l'algorithme AES. Il divise d'abord le message en une matrice 2x2, puis applique des opérations de substitution, de décalage de lignes, de mélange de colonnes et d'addition de clés de tour. Chaque tour utilise une clé de tour spécifique pour ajouter une couche de sécurité. Enfin, le texte chiffré est affiché.

```
le texte en clair 1100101010111101
apres le round 1 1010011110111111
le text encrypté  1111000001111100
```