Ecole supérieure en science et technologie de l'informatique et du numérique ESTIN

Compte rendu TP 01

Mini DES

Réalisé par :

Benmedakhene Souha (G2) .

Encadré par :

Dr . Bouchoucha Lydia

Année universitaire : 2023/2024

Introduction :

La sécurité des données est un enjeu majeur dans le monde numérique actuel. Le chiffrement est une technique essentielle pour protéger la confidentialité des informations sensibles lors de leur transmission ou stockage. Dans ce contexte, le TP visait à explorer et à implémenter l'algorithme de chiffrement Mini DES, une version simplifiée du célèbre standard de chiffrement DES (Data Encryption Standard).
Le Mini DES utilise une clef de 10 bits pour chiffrer des blocs de texte de 8 bits.

Objectifs :

1. Comprendre les principes fondamentaux du chiffrement :
    - Appréhender les concepts de base du chiffrement, y compris la substitution, la permutation et les opérations logiques.
    - Comprendre l'importance du chiffrement dans la protection des données sensibles contre les accès non autorisés.
2. Acquérir une expérience pratique dans l'implémentation d'un algorithme de chiffrement :
    - Se familiariser avec le processus de traduction des principes théoriques du chiffrement en code informatique.
    - Renforcer ses compétences en programmation en utilisant un langage de programmation tel que Python pour mettre en œuvre un algorithme de chiffrement.
3. Explorer les différentes étapes du processus de chiffrement Mini DES :
    - Analyser les étapes de génération de clés, de permutation initiale, de substitution , etc.
    - Comprendre comment ces étapes contribuent à la sécurité et à la robustesse de l'algorithme de chiffrement.
4. Tester le chiffrement en utilisant des exemples concrets :
    - Utiliser des exemples de textes en clair et de clés pour tester l'implémentation de l'algorithme de chiffrement.

Implémentation :

Etape 1: Génération des clés

L'étape de génération des clés dans l'algorithme Mini DES est cruciale car elle permet de créer les clés nécessaires pour chiffrer et déchiffrer les données. Cette étape se divise en plusieurs sous-étapes :

1. **Génération de la clé initiale (K0) :**
   - La clé initiale K0 est fournie en entrée ou générée aléatoirement. Dans Mini DES, la clé est représentée par une séquence de 10 bits.
   - Cette clé initiale est ensuite soumise à une permutation spécifique appelée permutation initiale (P10). Cette permutation réarrange les bits de la clé initiale selon une table de permutation prédéfinie. Le résultat est une clé intermédiaire de 10 bits .

2. **Dérivation des clés K1 et K2 :**

   - Une fois que la clé initiale est obtenue, elle est divisée en deux parties égales, généralement désignées comme étant L0 et R0. Chaque partie contient 5 bits.
   - Ensuite, chaque partie subit un décalage circulaire vers la gauche. Pour K1, L0 et R0 sont décalés d'un bit vers la gauche, tandis que pour K2, ils sont décalés de deux bits vers la gauche.
   - Après les décalages, les deux parties sont réassemblées et soumises à une autre permutation spécifique appelée permutation de choix (P8). Cette permutation réduit la taille de la clé de 10 bits à 8 bits en réorganisant les bits selon une table de permutation prédéfinie.
   - Le résultat final est la clé de chiffrement K1 pour le premier round de chiffrement, et la clé K2 pour le second round.

Et voici le code en Python :

```
mini_DES > generate_keys
# Permutation box P10
P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]

# Permutation box P8
P8 = [6, 3, 7, 4, 8, 5, 10, 9]

def permute(key, permutation_box):
    permuted_key = [key[i - 1] for i in permutation_box]
    return ''.join(permuted_key)

def left_circular_shift(bits):
    return bits[1:] + bits[0]

def generate_keys(K0):
    # Apply P10 permutation
    K0_permuted = permute(K0, P10)

    # Split K0 into two parts
    L0 = K0_permuted[:5]
    R0 = K0_permuted[5:]

    # Left circular shift on L0 and R0 for K1
    L1 = left_circular_shift(L0)
    R1 = left_circular_shift(R0)

    # Concatenate L1 and R1 for K1
    result1 = L1 + R1

    # Apply P8 permutation to get K1
    K1 = permute(result1, P8)

    # Left circular shift on L1 and R1 by 2 bits for K2
    L2 = left_circular_shift(left_circular_shift(L1))
    R2 = left_circular_shift(left_circular_shift(R1))

    # Concatenate L2 and R2 for K2
    result2 = L2 + R2
```

```
    # Apply P8 permutation to get K2
    K2 = permute(result2, P8)

    return K1, K2

# Take input for K0
K0 = input("Enter the 10-bit key K0: ")

# Generate K1 and K2
K1, K2 = generate_keys(K0)

print("K1:", K1)
print("K2:", K2)
```

Etape 2: Chiffrement

L'étape de chiffrement dans l'algorithme Mini DES est le cœur du processus, où les données en clair sont transformées en données chiffrées à l'aide des clés de chiffrement générées précédemment (K1 et K2). Cette étape est composée de plusieurs sous-étapes, chacune contribuant à la sécurité et à la confidentialité des données chiffrées. Voici une explication détaillée de chaque aspect de l'étape de chiffrement :

1. **Permutation initiale (PI) :**

- Au début de l'étape de chiffrement, les données en clair sont soumises à une permutation initiale (PI). Cette permutation réorganise les bits du texte en clair selon une table de permutation spécifique appelée permutation initiale (PI). L'objectif de cette permutation est de mélanger les bits du texte en clair de manière à améliorer la diffusion et la confusion, deux propriétés essentielles pour renforcer la sécurité du chiffrement.

2. **Rounds de chiffrement (**en utilisant la fonction F(Ki) **) :**
    - Le chiffrement dans Mini DES se déroule en deux rounds, chacun utilisant une clé de chiffrement différente (K1 pour le premier round et K2 pour le deuxième round).
    - Dans chaque round, les blocs L et R sont soumis à une série d'opérations, y compris l'expansion, la substitution, la permutation et le XOR avec la clé de chiffrement correspondante.
    - L'expansion consiste à étendre le bloc R0 en utilisant une table de permutation spécifique (EP), qui ajoute des bits supplémentaires au bloc pour augmenter la complexité du chiffrement.
    - Ensuite, le bloc étendu est combiné avec la clé de chiffrement (K1 ou K2) en utilisant l'opération de XOR (OU exclusif).
    - Après le XOR, le résultat est divisé en deux parties, qui sont ensuite soumises à des S-boxes (boîtes de substitution) . Les S-boxes remplacent chaque groupe de bits par un autre groupe de bits en utilisant des tables de substitution spécifiques (S0 pour la première partie et S1 pour la deuxième partie).
    - Les résultats des S-boxes sont permutés à l'aide d'une autre table de permutation spécifique (P4), ce qui réorganise les bits pour obtenir le texte chiffré.
    - Après on va faire un XOR entre le résultat et le " L" en entrée .
    - Enfin , à la fin de F(K1) on va faire un SW (les blocs de sortie sont combinés dans un ordre spécifique) et la même chose pour F(K2) .

3. **Permutation finale (PI -1) :**
    - Une fois les rounds de chiffrement terminés, les blocs de sortie sont soumis à une permutation finale (PI -1) (c'est la matrice inverse de PI). Cette permutation inverse la permutation initiale et restaure l'ordre original des bits, produisant ainsi le texte chiffré final.

Et voici le code en Python :

```python
PI = [2, 6, 3, 1, 4, 8, 5, 7]

def initial_permutation(text):
    return permute(text, PI)


EP = [4, 1, 2, 3]

S0 = [
    ['01', '00', '11', '10'],
    ['11', '10', '01', '00'],
    ['00', '10', '01', '11'],
    ['11', '01', '00', '10']
]

S1 = [
    ['00', '10', '10', '11'],
    ['10', '00', '01', '11'],
    ['11', '00', '01', '00'],
    ['10', '01', '00', '11']
]

P4 = [2, 4, 3, 1]

def xor(a, b):
    # Perform bitwise XOR between two binary strings of equal length
    return ''.join(str(int(x) ^ int(y)) for x, y in zip(a, b))


def s_box(bits, sbox):
    row = int(bits[0] + bits[3],2)
    col = int(bits[1]+ bits[2], 2)
    return sbox[row][col]

```

```python
def f(R0, K1):
    # Expand R0 using EP
    expanded_R0 = permute(R0, EP)
    # XOR with K1
    xor_result = xor(expanded_R0, K1)
    # Split the result into two 4-bit halves
    left_half = xor_result[:4]
    right_half = xor_result[4:]
    print("Left half:", left_half)
    print("Right half:", right_half)  # Add this line for debugging
    # Apply S-boxes
    s0_result = s_box(left_half, S0)
    s1_result = s_box(right_half, S1)
    # Concatenate the results
    concatenated_result = s0_result + s1_result
    # Apply permutation P4
    permuted_result = permute(concatenated_result, P4)
    return permuted_result

def swap_and_permute(R0, R1):
    return R1 + R0


def f_with_k2(R0, K2):
    return f(R0, K2)


def swap_and_permute_final(R2, R1):
    return R2 + R1


IP_1 = [4, 1, 3, 5, 7, 2, 8, 6]

def final_permutation(text):
    return permute(text, IP_1)

```

```python

def encrypt(plaintext, K1, K2):
    # Initial permutation
    permuted_text = initial_permutation(plaintext)
    # Split into L0 and R0
    L0 = permuted_text[:4]
    R0 = permuted_text[4:]
    # Round 1
    R1 = f(R0, K1)
    # Swap and permute
    swapped_text = swap_and_permute(R0, R1)
    # Round 2
    R2 = f_with_k2(swapped_text[:4], K2)
    # Swap and permute final
    final_text = swap_and_permute_final(R2, swapped_text[4:])
    # Final permutation
    encrypted_text = final_permutation(final_text)
    return encrypted_text


plaintext = input("Enter the plaintext (8 bits): ")


# Encrypt and print the result
encrypted_text = encrypt(plaintext, K1, K2)
print("Encrypted text:", encrypted_text)
```

Etape 3: Test et Validation

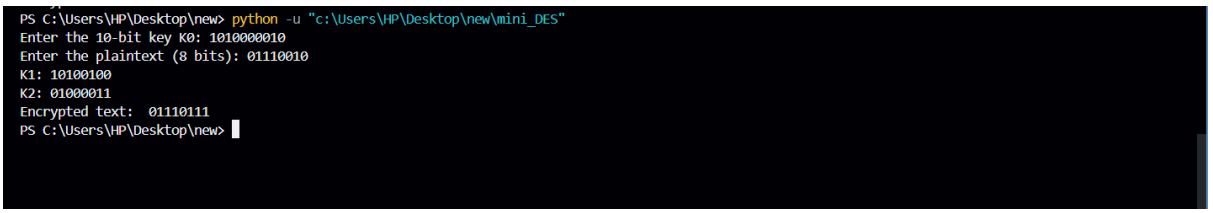
La validation est une étape critique dans le processus de développement d'un algorithme de chiffrement, car elle permet de s'assurer que le chiffrement fonctionne comme prévu et offre un niveau de sécurité adéquat .

Pour valider le chiffrement, différents exemples de texte en clair et de clés sont utilisés pour tester l'implémentation.

Text en claire : 01110010 et la clé K0 : 1010000010 .

```
PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\mini_DES"
Enter the 10-bit key K0:
```

```
PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\mini_DES"
Enter the 10-bit key K0: 1010000010
Enter the plaintext (8 bits):
```



Conclusion :

Dans le cadre de ce TP, nous avons eu l'opportunité d'approfondir notre compréhension des concepts fondamentaux du chiffrement, ainsi que de nous familiariser avec l'algorithme Mini DES. En mettant en œuvre cet algorithme de manière pratique, nous avons acquis des connaissances approfondies sur son fonctionnement interne, notamment sur les différentes étapes telles que la génération des clés .. Cette expérience pratique nous a permis de mieux appréhender les défis techniques associés à l'implémentation d'un algorithme de chiffrement, notamment en ce qui concerne la manipulation des données binaires et la gestion des opérations de permutation et de substitution.