# 作業三

## 利用DES加密器舉出崩塌效應案例

AN4111071_資訊系_葉治嘉

# 報告順序

程式碼展示及說明(圖片註解)

過程中遇到的錯誤、校正及理解

```python
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
import binascii

def des_encrypt(key, plaintext):
    cipher = DES.new(key, DES.MODE_ECB)
    return cipher.encrypt(plaintext)

def hamming_distance(str1, str2):
    return sum(bin(b1 ^ b2).count('1') for b1, b2 in zip(str1, str2))

def difference_ratio(hamming_distance, length):
    return (hamming_distance / length) * 100

# 給予設定好的明文、金鑰(只是為了展示崩塌效應，因此隨便給予)。
binary_key1 = '0001001000110100010101100111100010011010101111001101111011110000'
binary_key2 = '0001001000110100010101100110100010011010101111001101111011110000' # 使binary_key2與binary_key1只有1bit的不同(在第28個bit)
key1 = int(binary_key1, 2).to_bytes(8, byteorder='big')
key2 = int(binary_key2, 2).to_bytes(8, byteorder='big')  # 將key轉成16進位儲存

binary_plaintext1 = '0110000101100010011000110110010001100101011001100110011101101000'
binary_plaintext2 = '0110000101100010011000110110011001100101011001100110011101101000' # 使binary_plaintext2與binary_plaintext1只有1bit的不同(在第31個bit)
plaintext1 = int(binary_plaintext1, 2).to_bytes(8, byteorder='big')
plaintext2 = int(binary_plaintext2, 2).to_bytes(8, byteorder='big')  # 將plaintext轉成16進位儲存

# 利用內建函式加密第一個案例:不同key(相差1bit)，相同明文
ciphertext1_case1 = des_encrypt(key1, plaintext1)
ciphertext2_case1 = des_encrypt(key2, plaintext1)

# 利用內建函式加密第二個案例:不同明文(相差1bit)，相同key
ciphertext1_case2 = des_encrypt(key1, plaintext1)
ciphertext2_case2 = des_encrypt(key1, plaintext2)

# 利用內建函式計算兩個案例的漢明距離跟差異比例
distance_case1 = hamming_distance(ciphertext1_case1, ciphertext2_case1)
ratio_case1 = difference_ratio(distance_case1, len(ciphertext1_case1) * 8)

distance_case2 = hamming_distance(ciphertext1_case2, ciphertext2_case2)
ratio_case2 = difference_ratio(distance_case2, len(ciphertext1_case2) * 8)

# 顯示結果。
#可以清晰地看到1bit不同卻會導致大約50%的差異，清晰地感受到崩塌效應。
print("案例1:")
print(f"Ciphertext 1: {binascii.hexlify(ciphertext1_case1)}")
print(f"Ciphertext 2: {binascii.hexlify(ciphertext2_case1)}")
print(f"漢明距離: {distance_case1}")
print(f"差異比例: {ratio_case1:.2f}%")
print("\n案例2:")
print(f"Ciphertext 1: {binascii.hexlify(ciphertext1_case2)}")
print(f"Ciphertext 2: {binascii.hexlify(ciphertext2_case2)}")
print(f"漢明距離: {distance_case2}")
print(f"差異比例: {ratio_case2:.2f}%")
```

```
PS C:\Users\qoo09> python -u "c:\User
案例1:
Ciphertext 1: b'4003060e8db0d26f'
Ciphertext 2: b'7c670c69454e9f7d'
漢明距離: 30
差異比例: 46.88%

案例2:
Ciphertext 1: b'4003060e8db0d26f'
Ciphertext 2: b'ad28e582c810c8b8'
漢明距離: 32
差異比例: 50.00%
```

# 錯誤與修正:

若助教有乖乖看我雜亂ㄉ程式碼、註解，會發現上一頁
的key怪怪的，雖然是64bits卻沒符合奇偶校驗的規則。

```
# 給予設定好的明文、金鑰(只是為了展示崩塌效應，因此隨便給予)。
binary_key1 = '0001001000110101010101100111100010011010101111011101111011110000'
binary_key2 = '0001001000110101010101100110100110011010101111011101111011110000'
# 使binary_key2與binary_key1只有1bit的不同(在第28個bit)，其他不同則是奇偶校驗的結果，因為最後會被捨棄因此無所謂
```

因為我一開始疏忽了，只是亂給了一個key。
不過因為這次的大意，讓我再次確定了一件事，就是
奇偶校驗所加的bits完全不會影響答案
(更正前後output值仍然相同)

```
案例1:
Ciphertext 1: b'4003060e8db0d26f'
Ciphertext 2: b'7c670c69454e9f7d'
漢明距離: 30
差異比例: 46.88%

案例2:
Ciphertext 1: b'4003060e8db0d26f'
Ciphertext 2: b'ad28e582c810c8b8'
漢明距離: 32
差異比例: 50.00%
```

原因主要為: 雖然在實際的硬體中，不遵循這個規則的金鑰會被拒絕。然而，在許多軟體中（包括一些加密庫），這個規則不一定會強制執行，因為它屬於錯誤檢測，而不是安全性的一部分。