NSUCRYPTO2024

Problem 4: Weak key schedule for DES

October 21, 2024

Solution

The secret round key (in decimal): 13696557016106

The recovered message: **It is better to be in chains with friends, than to be in a garden with strangers**

Below is our step to solve the problem.

DES algorithm implementation

We start by re-implementing the DES algorithm using Python, following the guidelines from the note https://emadalsuwat.github.io/cryptography/DES.pdf. Our focus is on analyzing the encrypt function, where Alice mistakenly uses the same round key for all rounds due to an indexing error.

We will implement the DES algorithm by splitting it into multiple functions. Each function's purpose will be explained below, and we will test the implementation to verify correctness.

```python
def create_subkeys(key: int) -> list[str]:
    K, Ks = bin(key)[2:].zfill(64), []
    # apply permutation
    K_plus = [K[i - 1] for i in pc1]
    K_plus = ''.join(K_plus)
    Cs, Ds = [K_plus[:28]], [K_plus[28:]]
    for i, r in enumerate(ROTATES):
        Cs.append(Cs[i][r:] + Cs[i][:r])
        Ds.append(Ds[i][r:] + Ds[i][:r])

    for Ci, Di in zip(Cs, Ds):
        K = Ci + Di
        K = [K[i - 1] for i in pc2]
        K = ''.join(K)
        Ks.append(K)

    return Ks
```

Figure 1: Create Subkeys Function

The `create_subkeys` function generates 16 subkeys from the 64-bit DES key by applying permutations and rotations. It first splits the key into two halves, applies the PC1 permutation, then uses the specified number of rotations before applying PC2 to generate the subkeys for each round.

```python
def f(R: str, K: str) -> str:
    R = [R[i - 1] for i in EXPANSION]
    R = ''.join(R)
    RK = strxor(R, K)
    B = [RK[i:i+6] for i in range(0, len(RK), 6)]
    B = [SB[i][int(b[0] + b[-1], 2) * 16 + int(b[1:-1], 2)] for i, b in enumerate(B)]
    B = [bin(b)[2:].zfill(4) for b in B]
    B = ''.join(b for b in B)
    B = [B[i - 1] for i in P]
    return ''.join(B)
```

Figure 2: Round Function

The `f` function is the round function for DES. It expands the 32-bit right half R to 48 bits, XORs it with the round key K, and applies the S-box transformation. The result is permuted by the permutation P.

The `encode` function performs the full DES encryption. It applies the initial permutation to the message M, then runs 16 rounds of DES using the subkeys. After the final round, the halves are swapped and the inverse permutation is applied to produce the final ciphertext.

To ensure that we have implemented the algorithm correctly, we will test it using a

```python
def encode(M: int, Ks: list):
    M = bin(M)[2:].zfill(64)
    M = [M[i - 1] for i in IP]
    Ls, Rs = [''.join(M[:32])], [''.join(M[32:])]
    for n in range(1, 17):
        Ln = Rs[n - 1]
        Rn = strxor(Ls[n - 1], f(Rs[n - 1], Ks[n]))
        Ls.append(Ln)
        Rs.append(Rn)

    RL16 = Rs[-1] + Ls[-1]
    RL16 = ''.join([RL16[i - 1] for i in IP_inv])
    RL16 = int(RL16, 2)
    return hex(RL16)[2:].zfill(16)
```

Figure 3: Encode Function

known key and plaintext. The test case uses the key `"0E329232EA6D0D73"` and plaintext `"8787878787878787"`. The correct ciphertext should be `"0000000000000000"`.

```python
key = int('0E329232EA6D0D73', 16)
Ks = create_subkeys(key)
M = int('8787878787878787', 16)
print(encode(M, Ks) == "0" * 16) # True
```

Figure 4: Test DES Implementationn

The expected output is `True`, indicating that the encryption matches the expected result.

We are now ready to look at Alice's mistake, focusing on the `encode` function. Instead of using `Ks[n]` for the n-th round, Alice mistakenly used `Ks[1]` (the first round key) for all 16 rounds:

```
Rn = strxor(Ls[n - 1], f(Rs[n - 1], Ks[n]))
-> Rn = strxor(Ls[n - 1], f(Rs[n - 1], Ks[1]))
```

Exploit using Slide attack

Due to Alice's mistake, our goal now is to recover the first round key (Ks_1). We first find a plaintext pair (after applying the initial permutation) P_0 and P_1 such that:

$$P_0 = (L_0, R_0)$$
$$P_1 = (R_0, L_0 \oplus f(R_0, Ks_1))$$

This means P_1 could be the output of P_0 after one round of encryption, allowing us to find back Sk_1.

```
pt = open("Book.txt", "rb").read()[:-1]
ct = open("Book_cipher.txt", "rb").read()

pt_blocks = [pt[i:i + 8] for i in range(0, len(pt), 8)]
ct_blocks = [ct[i:i + 8] for i in range(0, len(ct), 8)]

pt_bits = [[int(x) for x in bin(int(y.hex(), 16))[2:].zfill(64)]
                    for y in pt_blocks]

pt_bits = [[x[i - 1] for i in IP] for x in pt_bits]

ct_bits = [[int(x) for x in bin(int(y.hex(), 16))[2:].zfill(64)]
                    for y in ct_blocks]

ls = [''.join(str(y) for y in x[:32]) for x in pt_bits]
rs = [''.join(str(y) for y in x[32:]) for x in pt_bits]

print(set(ls).intersection(set(rs)))



"""
Output potential R_0 values:
{'00000000101111101000001001000100',
'00000000111111110010001000000010',
'00000000111111110110000010000010'}
"""
```

Figure 5: Finding collision

From here, we can easily determine P_0 and P_1. One such valid pair is P_0 being the first 64-bit chunk from the plaintext:

$$P_0 = 1101111101000101000110011001100000000000010111110100000100100100$$

$$P_1 = 0000000010111110100000100100010000000000111111111000001000000010$$

Thus, P_0 represents the initial state, while P_1 could be the result after one round of encryption using Alice's faulty key.

```python
def find_key(P0, P1, CT):
    L0, R0 = P0[:32], P0[32:]
    L1, R1 = P1[:32], P1[32:]
    assert R0 == L1, "nope"
    R1 = strxor(L0, R1) # R1 = L0 + f(R0, K), remove L0 to get f(R0, K)
    # Apply Pinv to remove permutation P
    R1 = ''.join([R1[i - 1] for i in Pinv])
    # Next split in block of 4 bits
    R1 = [R1[i:i+4] for i in range(0, len(R1), 4)]
    # Find mapping from SB
    xs = []
    for i, y in enumerate(R1):
        y = int(y, 2)
        res = []
        for j in range(len(SB[i])):
            if SB[i][j] == y:
                pos = bin(j // 16)[2:].zfill(2)
                res.append(pos[0] + bin(j % 16)[2:].zfill(4) + pos[1])
        xs.append(res)
    combinations = list(itertools.product(*xs))

    # M = apply IP_inv on P0, use M and key to encrypt 16 rounds, If it maches CT -> found Alice's key
    M = ''.join([P0[i - 1] for i in IP_inv])
    M = int(M, 2)

    R_expanded = ''.join([R0[i - 1] for i in EXPANSION])

    for combo in combinations:
        possible_k = ''.join(x for x in combo)
        possible_k = strxor(R_expanded, possible_k)
        check = encode(M, [possible_k], mode=0) # mode = 0 to encrypt 1 round only
        assert check == P1
        if (bin(int(encode(M, [possible_k]), 16))[2:].zfill(64) == CT):
            print(f"found key: {possible_k}")
            return possible_k
```

Figure 6: Find key

We use to script above to recover the first-round key Ks_1. The ciphertext CT corresponds to the first 64 bits of the actual ciphertext. The process involves reversing the function f from the encryption process. Since the S-box maps 6 bits to 4 bits, multiple candidate keys arise (up to 4 possibilities per value/S-box). So in total, we have to try 65536 candidate keys (since we have 8 S-Boxes).

For each candidate key, we perform a full encryption on M. The correct key will produce the ciphertext CT, confirming the key's validity.

After that, we can decrypt the challenge's message using the encryption process

```python
P0 = '11011111010001010001100110011000000000001011111010000010010000100'
P1 = '00000000101111101000001001000100000000001111111110000010000000010'
CT = '00010001011101000010110111101100011111011100001101111001101010011'

key = find_key(P0, P1, CT)

Ks = [key]

msg = """86991641D28259604412D6BA88A5C0A6471CA7222C52482BF2D0
E841D4343DFB877DC8E0147F3D5F20FC18FF28CB5C4DA8A0F4694861AB5E98F37ADBC2D69B35779D9001BB4B648518FE6EBC00B2AB10""".replace('\n', '')

msg_block = [msg[i:i+16] for i in range(0, len(msg), 16)]

recover_msg = b''

for msg_b in msg_block:
    recover_msg += bytes.fromhex(encode(int(msg_b, 16), Ks))

print(recover_msg.decode())

"""
found key: 0000110001110100111110100110101001100100000101010
It is better to be in chains with friends, than to be in a garden with strangers
"""
```

Figure 7: Decrypt message

Please refer to the solution script for more details on NSUCRYPTO2024 Problem 4.