Ministerul Educației, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Raport

pentru lucrare de laborator Nr. 4
la cursul "*Cifruri bloc. Algoritmul DES*"

A efectuat: Arteom KALAMAGHIN, FAF-211
A verificat: Aureliu ZGUREANU

Chișinău - 2023

Subject: Cryptanalysis of polyalphabetic ciphers

Tasks:

Studiați materiale didactice plasate pe ELSE. De elaborat un program în unul din limbajele de programare preferate pentru implementarea unui element al algoritmului DES. Sarcina se va alege în conformitate cu numărul n de ordine al studentului din lista grupei, în conformitate cu formula: nr_sarcina = n mod 11 (16 mod 11 = 5). Pentru fiecare sarcină să fie afișate la ecran tabelele utilizate și toți pașii intermediari. Datele de intrare să fie posibil de introdus de utilizator sau de generat în mod aleatoriu. Atenție! La susținerea lucrării vor fi puse întrebări despre lucrul întregului algoritm!!!

Theoretical notes:

The Data Encryption Standard (DES) is a historic and widely recognized symmetric-key block cipher encryption algorithm. Developed in the early 1970s by IBM and adopted as a federal standard in the United States, DES has played a crucial role in securing data for several decades. It operates on 64-bit blocks of data and uses a 56-bit encryption key. DES employs a 16-round Feistel network structure, involving complex permutation and substitution operations, which provide confusion and diffusion. While it was once considered highly secure, the relatively short key length became a vulnerability over time, leading to the development of more advanced encryption standards. Today, DES is primarily of historical significance, having been succeeded by more robust encryption algorithms like AES. In DES, the key scheduling algorithm takes a 64-bit encryption key and produces 16 subkeys, each 48 bits in length, which are used in the 16 rounds of the DES encryption process. The following steps can be used to summarize key scheduling. First, a permutation (PC-1) is applied to the 64-bit encryption key, which rearranges the bits (making the key into two 28-bit halves and reducing the 64-bit key to 56 bits after eight parity bits are removed). To generate a new 56-bit key for each of the 16 rounds, the two 28-bit halves of the key are independently shifted (either one or two positions leftward, depending on the round number). This process creates a unique subkey for every round. After the 56-bit key has been rotated, it is again subjected to a permutation (PC-2) to choose 48 bits out of the 56 bits; the data encryption for that round uses these 48-bit subkeys.

Implementation:

I've started from implementing PC-1:

```
def initial_permutation(key):
      pc1 = [57, 49, 41, 33, 25, 17, 9,
            1, 58, 50, 42, 34, 26, 18,
            10, 2, 59, 51, 43, 35, 27,
            19, 11, 3, 60, 52, 44, 36,
            63, 55, 47, 39, 31, 23, 15,
            7, 62, 54, 46, 38, 30, 22,
            14, 6, 61, 53, 45, 37, 29,
            21, 13, 5, 28, 20, 12, 4]
      key_permuted = [key[pc1[i] - 1] for i in range(56)]
      return key_permuted
...
key = initial_permutation(key)
```

The steps that follow are split and left circular shift:

```
def left_circular_shift(key, shift):
      return key[shift:] + key[:shift]
...
C, D = key[:28], key[28:]
...
shift = 2 if i in [0, 1, 8, 15] else 1
C = left_circular_shift(C, shift)
D = left_circular_shift(D, shift)
```

Next comes concatenation and PC-2:

```
def pc2_permutation(key):
      pc2 = [14, 17, 11, 24, 1, 5, 3, 28,
            15, 6, 21, 10, 23, 19, 12, 4,
            26, 8, 16, 7, 27, 20, 13, 2,
            41, 52, 31, 37, 47, 55, 30, 40,
            51, 45, 33, 48, 44, 49, 39, 56,
            34, 53, 46, 42, 50, 36, 29, 32]
      round_key = [key[pc2[i] - 1] for i in range(48)]
      return round_key
...
combined_key = C + D
round_key = pc2_permutation(combined_key)
```

On the next page I provide an ex-le of the detailed out. generated by the program for the initial 64-bit - 0111101000010101010100011100110111101000010101010100011100110111…

```
Initial key (64b):            0111101000010101010100011100110111101000010101010100011100110111  *initial-perm. (PC-1)
Initial permutation (56b):  00011000011111011001000110101100000111101010000110010111               *split
Intial C and D (28b):         0001100001111101100100011010, 1100000111101010000110010111

Round -  1, shift = 2 @                                                                            *C and D left-circular shift
   ~ Shifted C and D (28b):   0110000111110110010001101000, 0000011110101000011001011111           *concatenation
   ~ Combined key (56b):      01100001111101100100011010000000011110101000011001011111            *compression-perm. (PC-2)
   ~ Round key (48b):         101000101001101001000001110111000000001111101100

Round -  2, shift = 2 @                                                                            *C and D left-circular shift
   ~ Shifted C and D (28b):   1000011111011001000110100001, 0001111010100001100101111100           *concatenation
   ~ Combined key (56b):      1000011111011001000110100001000111101010000110010111100            *compression-perm. (PC-2)
   ~ Round key (48b):         000010010111101001110110010100001111101011001001

Round -  3, shift = 1 @                                                                            *C and D left-circular shift
   ~ Shifted C and D (28b):   0000111110110010001101000011, 0011110101000011001011111000           *concatenation
   ~ Combined key (56b):      0000111110110010001101000011001111010100001100101111000            *compression-perm. (PC-2)
   ~ Round key (48b):         001001011100011001011100011010001010110011001101

Round -  4, shift = 1 @                                                                            *C and D left-circular shift
   ~ Shifted C and D (28b):   0001111101100100011010000110, 0111101010000110010111110000           *concatenation
   ~ Combined key (56b):      0001111101100100011010000110011110101000011001011110000            *compression-perm. (PC-2)
   ~ Round key (48b):         101001000111010111011000011100101011010000111001

Round -  5, shift = 1 @                                                                            *C and D left-circular shift
   ~ Shifted C and D (28b):   0011111011001000110100001100, 1111010100001100101111100000           *concatenation
   ~ Combined key (56b):      0011111011001000110100001100111101010000110010111100000            *compression-perm. (PC-2)
   ~ Round key (48b):         010001100101000110010110101010101101010010011111

...

Round - 14, shift = 1 @                                                                            *C and D left-circular shift
   ~ Shifted C and D (28b):   0010001101000011000011111011, 0011001011111000001111010100           *concatenation
   ~ Combined key (56b):      0010001101000011000011111011001100101111100000111010100            *compression-perm. (PC-2)
   ~ Round key (48b):         000100111011100001111000111110010001011000001001

Round - 15, shift = 1 @                                                                            *C and D left-sircular shift
   ~ Shifted C and D (28b):   0100011010000110000111110110, 0110010111110000011110101000           *concatenation
   ~ Combined key (56b):      0100011010000110000111110110011001011111100000111010100            *compression-perm. (PC-2)
   ~ Round key (48b):         100100001110100010011101001110111001011011100100

Round - 16, shift = 2 @                                                                            *C and D left-sircular shift
   ~ Shifted C and D (28b):   0001101000011000011111011001, 1001011111000001111010100001           *concatenation
   ~ Combined key (56b):      0001101000011000011111011001100101111100000111010100001            *compression-perm. (PC-2)
   ~ Round key (48b):         000101010010011100010110000110001100110110100111
```

Conclusion:

In conclusion, the implementation of the key scheduling part of the DES cipher has provided valuable insights into the intricate process of key generation in this encryption algorithm. Through the initial permutation, the splitting of the key into two 28-bit halves, and the subsequent rotations and subkey generation, we've observed how DES creates distinct and evolving subkeys for each of its 16 rounds. This exercise has highlighted the significance of key scheduling in enhancing the security of the DES algorithm and has underlined the importance of using unique subkeys in each round to thwart potential attacks. The knowledge gained from this lab work is fundamental in understanding DES and its historical significance in the realm of cryptography.

To check out the source code access my gitHub repo:
https://github.com/Starlight-Crusader/CS-Lab