Kryptografia

Tomasz Perła

26.03.2025

Spis Treści

1 Algorytm DES (Data Encryption Standard)

1.1 Generowanie podkluczy dla poszczególnych rund

1. 64-bitowy klucz poddaje się permutacji usuwającej bity parzystości (z każdego bajtu klucza) $PC_1(K_{64})$

$$PC_{1\,L} = \begin{bmatrix} 57 & 49 & 41 & 33 & 25 & 17 & 9 \\ 1 & 58 & 50 & 42 & 34 & 26 & 18 \\ 10 & 2 & 59 & 51 & 43 & 35 & 27 \\ 19 & 11 & 3 & 60 & 52 & 44 & 36 \end{bmatrix} \quad PC_{1\,R} = \begin{bmatrix} 63 & 55 & 47 & 39 & 31 & 23 & 15 \\ 7 & 62 & 54 & 46 & 38 & 30 & 22 \\ 14 & 6 & 61 & 53 & 45 & 37 & 29 \\ 21 & 13 & 5 & 28 & 20 & 12 & 4 \end{bmatrix}$$

$$PC_1 = \begin{bmatrix} PC_{1\,L} \\ PC_{1\,R} \end{bmatrix}$$

2. 56-bitową część klucza K_{56} dzieli się na dwie 28-bitowe połówki
3. Każdą połówkę poddaje się operacji rotacji bitowej w lewo przy czym:
   (a) Dla 1, 2, 9 i 16 rundy podklucze powstają poprzez ROL o 1 pozycję
   (b) Dla pozostałych rund ROL o 2 pozycje
4. Dwie zrotowane połówki ponownie łączymy w 56-bitową całość
5. Powstałe w ten sposób 16 podkluczy poddaje się działaniu permutacji kompresującej do 48-bitów $PC_2(K_{i\,56})$

$$PC_2 = \begin{bmatrix} 14 & 17 & 11 & 24 & 1 & 5 & 3 & 28 \\ 15 & 6 & 21 & 10 & 23 & 19 & 12 & 4 \\ 26 & 8 & 16 & 7 & 27 & 20 & 13 & 2 \\ 41 & 52 & 31 & 37 & 47 & 55 & 30 & 40 \\ 51 & 45 & 33 & 48 & 44 & 49 & 39 & 56 \\ 34 & 53 & 46 & 42 & 50 & 36 & 29 & 32 \end{bmatrix}$$

1.2 Szyfrowanie tekstu jawnego

1. 64-bitowy tekst jawny jest permutowany zgodnie z permutacją początkową $IP(X)$

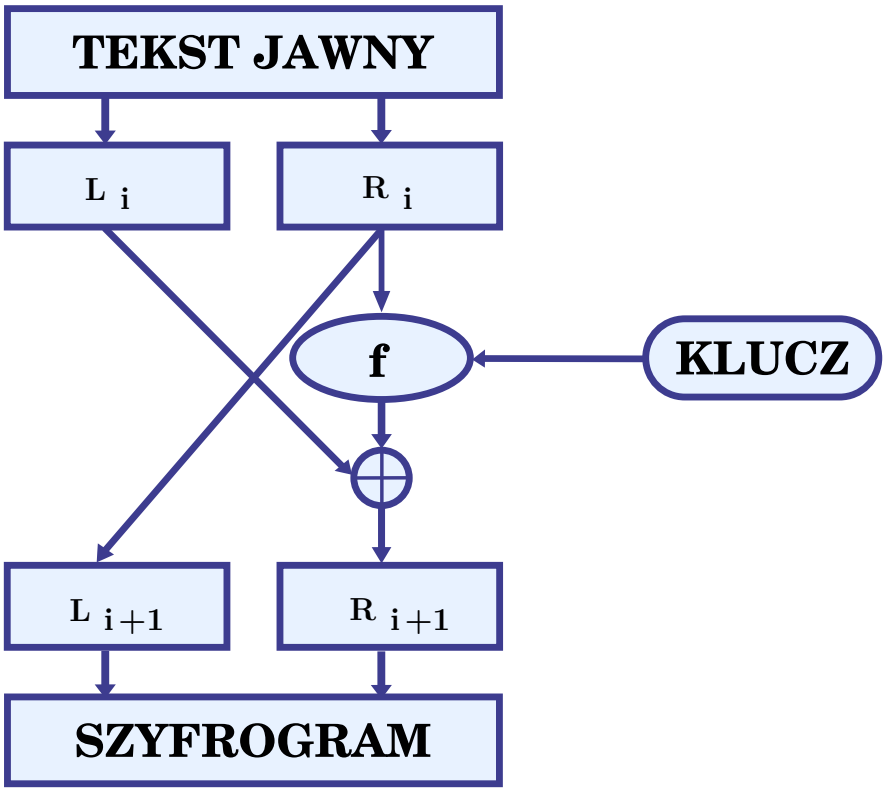
$$IP = \begin{bmatrix} 58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 \\ 60 & 52 & 44 & 36 & 28 & 20 & 12 & 4 \\ 62 & 54 & 46 & 38 & 30 & 22 & 14 & 6 \\ 64 & 56 & 48 & 40 & 32 & 24 & 16 & 8 \\ 57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 \\ 59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 \\ 61 & 53 & 45 & 37 & 29 & 21 & 13 & 5 \\ 63 & 55 & 47 & 39 & 31 & 23 & 15 & 7 \end{bmatrix}$$

2. Następuje podział X na dwie połówki 32-bitowe: L_i oraz R_i

3. Dla rundy $i+1$ zachodzą następujące zależości:

$$L_{i+1} = R_i \tag{1}$$

$$R_{i+1} = L_i \oplus f(R_i, K_{i+1}) \tag{2}$$



Sieć Feistela: https://pl.wikipedia.org/wiki/Sieć_Feistela

4. Po 16 rundach połówki R_{16} i L_{16} są łączone w 64-bitowy blok $X_{16} = R_{16}L_{16}$

5. Blok X_{16} jest poddawany permutacji końcowej (odwrotnej do początkowej) $IP^{-1}(X_{16})$

$$IP^{-1} = \begin{bmatrix} 40 & 8 & 48 & 16 & 56 & 24 & 64 & 32 \\ 39 & 7 & 47 & 15 & 55 & 23 & 63 & 31 \\ 38 & 6 & 46 & 14 & 54 & 22 & 62 & 30 \\ 37 & 5 & 45 & 13 & 53 & 21 & 61 & 29 \\ 36 & 4 & 44 & 12 & 52 & 20 & 60 & 28 \\ 35 & 3 & 43 & 11 & 51 & 19 & 59 & 27 \\ 34 & 2 & 42 & 10 & 50 & 18 & 58 & 26 \\ 33 & 1 & 41 & 9 & 49 & 17 & 57 & 25 \end{bmatrix}$$

1.3 Funkcja Feistela

1. Poddanie 32-bitowej R_i działaniu permutacji z rozszerzeniem do 48 bitów $P_{ex}(R_i)$

$$P_{ex} = \begin{bmatrix} 32 & 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 9 & 10 & 11 & 12 & 13 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 16 & 17 & 18 & 19 & 20 & 21 \\ 20 & 21 & 22 & 23 & 24 & 25 \\ 24 & 25 & 26 & 27 & 28 & 29 \\ 28 & 29 & 30 & 31 & 32 & 1 \end{bmatrix}$$

2. 48 bitów poddaje się operacji XOR z odpowiednim podkluczem $P_{ex}(R_i) \oplus K_i$

3. Otrzymane 48 bitów dzielone jest na 8 grup po 6 bitów. Każda grupa poddawana jest działaniu S-boksu S_j (kodującego 6-bitowe wartości na 4-bitowe). Bity 1 i 6 (licząc od najmniej znaczącego bitu) określają wiersz w S_j natomiast bity 2, 3, 4 i 5 określają kolumnę w S_j. W wyniku czego otrzymujemy 32-bitowy blok.

.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	4	6	2	10	13	11	8	3	15	5	1	14	7	9	0
1	14	3	11	6	8	15	1	10	5	13	0	7	4	12	9	2
2	7	10	8	15	13	2	11	1	14	9	3	4	12	5	0	6
3	9	5	2	11	3	12	7	14	10	8	6	13	1	4	0	15

Tabela 1: Przykładowy S-boks

4. Otrzymane 32 bity są na koniec permutowane P_{box}

$$P_{box} = \begin{bmatrix} 16 & 7 & 20 & 21 & 29 & 12 & 28 & 17 \\ 1 & 15 & 23 & 26 & 5 & 18 & 31 & 10 \\ 2 & 8 & 24 & 14 & 32 & 27 & 3 & 9 \\ 19 & 13 & 30 & 6 & 22 & 11 & 4 & 25 \end{bmatrix}$$

1.4 Implementacja w języku Java

```
1 private long des(byte[] keyB, byte[] msgB, boolean decrypt) {
2     //zamiana bajtow klucza na liczbe 64-bitowa
3     long key64 = bytesToLong(keyB);
4     //wybranie odpowiednich bajtow z klucza:
5     //usuniecie bitow parzystosci (najmniej znaczanych bitow z
          kazdego bajtu klucza oraz jego permutacja)
6     long key56 = selectBits(key64, this.pc1, 64);
7     //podzial 56 bitow klucza na dwie 28-bitowe polowy
```

```
8       int key28l = (int) (key56 >> 28);
9       int key28r = (int) (key56 & 0xfff_ffff);
10      long[] keys56 = new long[16];
11      //rotacja kazdej polowy o okreslona liczbe pozycji:
12      //dla 1,2,9 i 16 rundy o jedna pozycje
13      //dla pozostalych rund o dwie pozycje
14      for (int i = 0; i < 16; i++) {
15          key28l = rotLeft(key28l, this.shift[i]);
16          key28r = rotLeft(key28r, this.shift[i]);
17          //laczenie polowek w 56-bitowa calosc
18          keys56[i] = ((long) key28l << 28) | key28r;
19      }
20      long[] keys48 = new long[16];
21      //operacja permutacji kompresujacej do 48 bitow
22      for (int i = 0; i < 16; i++) {
23          keys48[i] = selectBits(keys56[i], this.pc2, 56);
24      }
25      //zamiana 64-bitowego bloku tekstu jawnego na liczbl
            64-bitowl
26      long msg = bytesToLong(msgB);
27      //permutacja poczatkowa bitow tekstu jawnego
28      long msg64 = selectBits(msg, this.ip, 64);
29      //podzial 64 bitow tekstu jawnego na dwie 32-bitowe polowy
30      int msg32l = (int) (msg64 >> 32);
31      int msg32r = (int) (msg64);
32      //wykonanie 16 rund
33      for (int i = 0; i < 16; i++) {
34          int next = 0;
35          //wykonujemy permutacje z rozszerzeniem do 48 bitow
36          //oraz operacje XOR z odpowiednim kluczem 48-bitowym
37          //(kolejnosc kluczy jest odwrotna dla deszyfrowania)
38          long k = selectBits(msg32r, this.ex, 32) ^
                keys48[decrypt ? 15 - i : i];
39          //dzielenie 48 bitow na 8 grup 6-bitowych
40          for (int j = 0; j < 8; j++) {
41              //6-bitowa grupa
42              int bit6 = (int) ((k >> j * 6) & 0b11_1111);
43              //wiersz w j-tym S-boksie
44              int row = (((bit6 & 0b10_0000) >> 4) & 0b10) |
                    (bit6 & 1);
45              //kolumna w j-tym S-boksie
46              int col = (bit6 & 0b01_1110) >> 1;
47              //dodanie 4-bitowego wyniku z S-boksa
48              next |= ((int) this.sbox[7 - j][16 * row + col])
                    << (j * 4);
49          }
```

```
50          //permutacja koncowa
51          int f = (int) selectBits(next, this.pbox, 32);
52          int lxor = msg32l ^ f;
53          msg32l = msg32r;
54          msg32r = lxor;
55      }
56      //zlaczenie lewej i prawej polowy w odwrotnej kolejnosci
57      long msg64rl = ((long) msg32r << 32) | ((long) msg32l &
            0xffff_ffffL);
58      //permutacja odwrotna do poczatkowej (permutacja koncowa)
59      long finalMsg = selectBits(msg64rl, this.ip1, 64);
60      return finalMsg;
61  }
```

Aplikacja szyfrująca DESX: https://github.com/matthiasoo/DESX_GUI.git

1.5 Odszyfrowywanie

Z zależności (1) i (2) wynika:

$$R_{i-1} = L_i \tag{3}$$

$$L_{i-1} = R_i \oplus f(R_{i-1}, K_i) = R_i \oplus f(L_i, K_i) \tag{4}$$

Zatem znając R_i, L_i i K_i można obliczyć R_{i-1} i L_{i-1}. Zatem podczas deszyfrowania wykonywane są te same operacje z wykorzystaniem podkluczy w odwrotnej kolejności.

2 DESX

2.1 Teoria

Do zaszyfrowania 64-bitowego bloku P_{64} można użyć trzech kluczy:
$K_{int} = K_1$, $K_{DES} = K_2$ oraz $K_{ext} = K_3$

$$X = K_{ext} \oplus DES(K_{DES},\ P \oplus K_{int}) \tag{5}$$

$$X = K_3 \oplus DES(K_2,\ P \oplus K_1) \tag{6}$$

gdzie: $DES(K,\ \Sigma)$ ooznacza szyfrowanie algorytmem DES 64-bitowego bloku Σ z kluczem K.

2.2 Odszyfrowywanie

W celu odszyfrowania szyfrogramu zaszyfrowanego algorytmem DESX należy użyć trzech kluczy w odwrotnej kolejności:

$$P = K_{int} \oplus DES(K_{DES},\ X \oplus K_{ext}) \tag{7}$$

$$P = K_1 \oplus DES(K_2,\ X \oplus K_3) \tag{8}$$

2.3 Implementacja

```
1  private long desx(byte[][] keys, byte[] msgB, boolean decrypt)
       {
2      //zamiana bajtow kluczy na liczby 64-bitowe
3      //przy odszyfrowywaniu klucze sa uzywane w odwrotnej
           kolejnosci
4      long key1 = bytesToLong(keys[decrypt ? 2 : 0]);
5      long key3 = bytesToLong(keys[decrypt ? 0 : 2]);
6      //zamiana 64-bitowego (8 bajtowego) bloku tekstu jawnego
           na liczbe 64-bitowa
7      long msg = bytesToLong(msgB);
8      //pierwszy etap: operacja XOR tekstu jawnego i pierwszego
           klucza
9      long pxork = msg ^ key1;
10     //drugi etap: uaycie algorytu DES z drugim kluczem
11     long des = des(keys[1], longToBytes(pxork), decrypt);
12     //trzeci etap: operacja XOR wyniku dzialania algorytmu DES
           i trzeciego klucza
13     long finalMsg = des ^ key3;
14     return finalMsg;
15 }
```