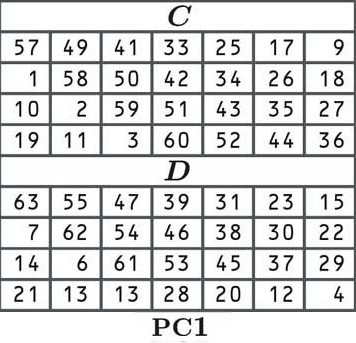
**PODSTAWY KRYPTOGRAFII – LABORATORIUM**  
Zadanie – Implementacja algorytmu 3DES

1. **Opis generowania 16 podkluczy**

Dany jest 64-bitowy klucz (jego wielkość jest zgoda z wielkością 64-bitowego bloku). Dokonujemy permutacji klucza zgodnie z następującą tablicą

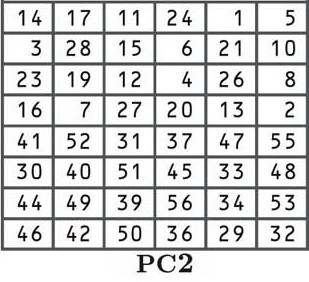


**5**

Permutacja polega na tym, że zawartość klucza ulega przemieszaniu zgodnie z położeniami bitów poddanych w tabeli PC1. Następuje konwersja z 64-bitów na 56-bitów. Wynika z tego, że 8 bitów zostaje pomiętych. W/w pominięcie wynika z historycznego używania bitów parzystości. Klucz 56-bitowy dzielimy na dwie części po 28-bitów każda. W zależności od numeru rundy dokonujemy przesunięcia w lewo o jednej lub dwa obu części klucza. Przesunięcie odbywa się zgodnie z następującą tabelą

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Numer rundy | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Liczba przesunięć w lewo | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

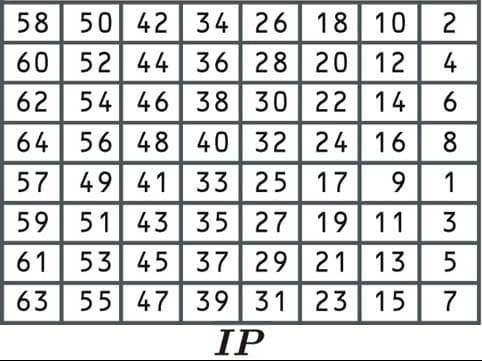
Dzięki operacji otrzymujemy 16 różnych podkluczy. Następnie dokonujemy permutacji w/w podkluczy zgodnie z tabelą PC2.



Zauważamy, że wynikiem jest zestaw szesnastu 48-bitowych podkluczy. Stąd wynika nazwa przeprowadzonego procesu – permutacja kompresująca.

1. **Opis algorytmu DES**

Dane zostają podzielone na 64-bitowe bloki. Dokonujemy permutacji bloku danych zgodnie z tabelą IP.



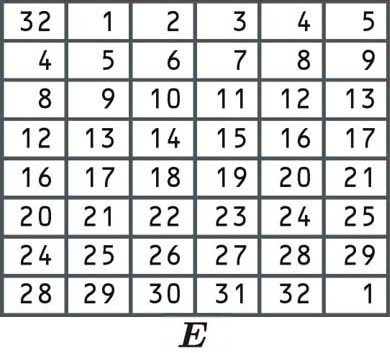
Następnie blok przechodzi przed 16 opisanej niżej rund.

Dzielimy dane na dwie 32-bitowe części – lewą i prawą. Prawa część zostaje poddana operacjom 4-krokowej F funkcji:

* permutacja rozszerzająca,
* mieszanie z kluczem,
* podstawienie,
* permutacja.

Permutacja rozszerzająca

Wynikiem ekspansywnej permutacji jest 48-bitów (długość równoliczna długości każdemu z 16 wygenerowanych podkluczy), które powstaje na podstawie permutacji zgodnej z tabelą E.



Mieszanie z kluczem

Na otrzymanych 48-bitach wykonywana jest operacja XOR z jednym z podkluczy (zgodnym z numerem rundy), gdzie XOR daje następujące przypadki:

0 + 0 = 0

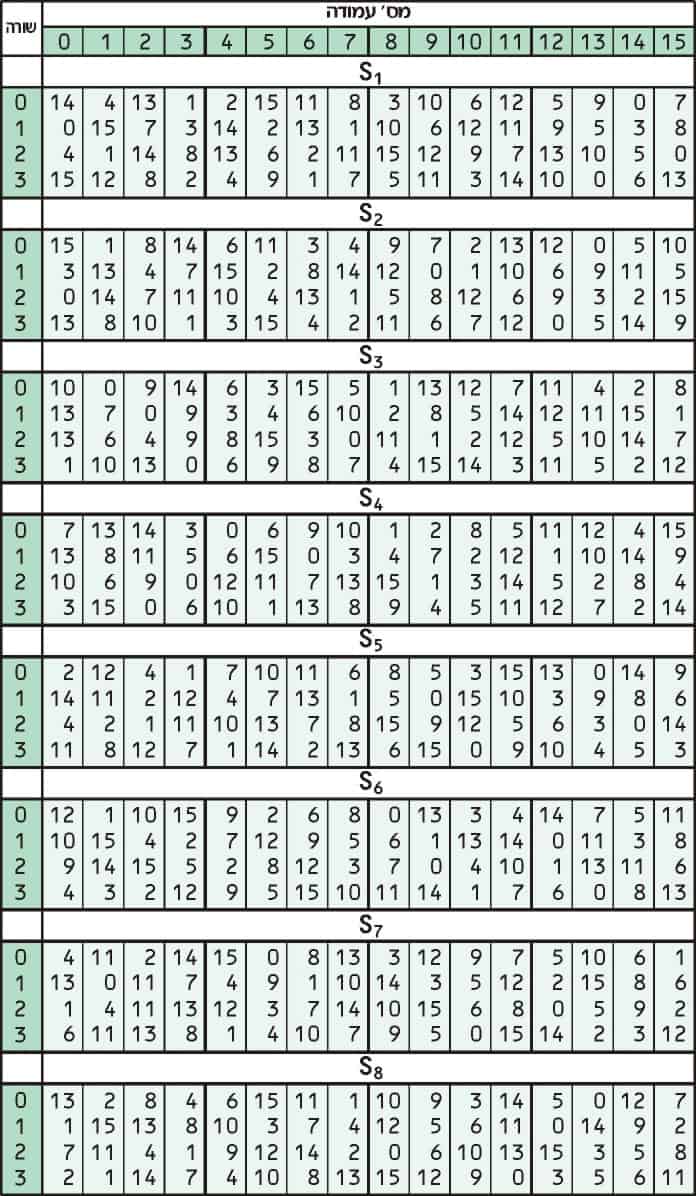
1 + 0 = 1

1 + 1 = 0

W wyniku operacji danej mamy 48 bitów.

Podstawienie

Dane 48-bitów dzielone jest na 8 grup po 6 bitów każda. Każda z grup poddawana jest działaniu S-boxa postaci



S-box służy translacji wejściowych 6 bitów na 4 bity wyjściowe. Dzieje się to w następujący sposób:

* Bierzemy wartość pierwszego i ostatniego bitu z 6 bitowej grupy, złączamy je   
  i odczytujemy dziesiątkową wartość otrzymanej liczby binarnej. Kombinacje są następujące:

|  |  |  |  |
| --- | --- | --- | --- |
| Bit pierwszy | Bit ostatni | Liczba binarna | Liczba dziesiętna |
| 0 | 0 | 00 | 0 |
| 1 | 0 | 10 | 2 |
| 0 | 1 | 01 | 1 |
| 1 | 1 | 11 | 3 |

W wyniku operacji otrzymujemy numer wiersza S-boxa (wartości z przedziału 0-3).

* Bierzemy cztery środkowe bity z 6 grupy i działamy analogicznie. W wyniku operacji otrzymujemy numer kolumny S-boxa (wartości z przedziału 0-15).

Pierwsza szóstka bitów odpowiada części tabeli S1, druga S2, itd. Odczytujemy wartość S-boxa o podanych współrzędnych i zapisujemy binarnie. Otrzymujemy 8 grup po 4 bity każda, co daje sekwencję 32 bitów.

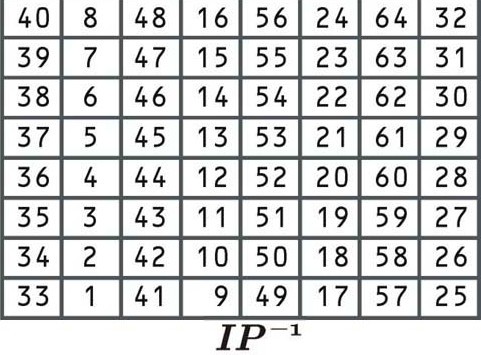
Permutacja

Otrzymaną w poprzednim kroku sekwencję 32-bitów poddajemy permutacji zgodnie z tabelą P.



Następnie prawą część danych (po wyżej opisanych operacjach – wciąż 32-bitową) część danych poddajemy operacji XOR z lewą częścią. Wynik rundy staję się prawą częścią kolejnej rundy, a oryginalna prawa część (przed funkcją F) staję się lewą częścią kolejnej rundy. W rundzie finałowej zamiast wyżej opisanej zamiany bloków są one łączone w 64-bitowy blok.

Blok 64-bitowy poddany jest finałowej permutacji zgodnej z tabelą IP-1:



otrzymując także 64-bitowy blok. W celu deszyfrowania danych używa się kluczy w przeciwnej kolejności.

1. **Opis algorytmu 3DES**

3DES używa trzykrotnie algorytmu DES z trzema różnymi kluczami 56-bitowymi. Pierwszy klucz służy zwykłemu zaszyfrowaniu danych. Drugi klucz deszyfruje dane zaszyfrowane kluczem pierwszym. Trzeci klucz szyfruje dane odszyfrowane kluczem drugim. Deszyfrowanie przebiega następująco: deszyfrowanie przy pomocy klucza pierwszego, szyfrowanie za pomocą klucza drugiego, deszyfrowanie za pomocą klucza trzeciego.

1. **Implementacja**

Działanie algorytmu 3DES zostało zaimplementowane z języku Java. Do funkcjonalności aplikacji należy generowanie kluczy (pojedynczo, jak i generowanie zestawu 3 kluczy), zapis zestawu 3 kluczy do pliku, wczytanie zestawu kluczy z pliku. Klucze początkowe są 64-bitowe, generowane jako 8 znaków UTF-8. Szyfrowanie i deszyfrowanie możliwe jest wyłącznie, gdy wszystkie 3 klucze zostały wygenerowane, w przeciwnym razie otrzymamy komunikat o braku klucza z wyszczególnieniem o który klucz chodzi. Komunikat otrzymamy także w wyniku próby zaszyfrowania pustej wiadomości. Tekst jawny oraz szyfrogram jesteśmy w stanie zapisać do pliku oraz wczytać z pliku. Wczytywany tekst jawny i szyfrogram automatycznie pojawia się w polu tekstowym. Analogiczna operacja ma miejsce z rezultatem szyfrowania i deszyfrowania. Interfejs graficzny użytkowania jest następującej postaci:

****

Kod źródłowy programu podzielony został na dwa moduły: moduł odpowiedzialny za widok oraz moduł odpowiedzialny za merytoryczną realizację operacji dla algorytmu 3DES.

Podstawową używaną w implementacji strukturą jest *Block*, który przedstawiany jest jako ośmioelementowa tablica bajtów. Do jej przeciążeń używanych w implementacji należą *Block8* – bajt, *Block32* – czteroelementowa tablica bajtów, *Block48* – sześcioelementowa tablica bajtów, *Block56* – siedmioelementowa tablica bajtów.

W celu dokonania szyfrowania konieczne jest wygenerowanie trzech ośmiobajtowych kluczy, każdy z nich stosowany jest do jednej z trzech operacji DES. Dla jednej operacji DES należy w następujący sposób utworzyć szesnaście podkluczy:

* stworzenie tablicy 16 elementowej, która zawartością będzie blok 48-bitowy,
* redukcja 64-bitowego klucza do 56-bitowego z pomocą permutacji,
* podzielenie 56-bitowego klucza na dwa 28-bitowe połowy (dopełnione do pełnych bajtów – 32 bitów)
* przesunięcie w lewo każdej połowy w zależności od numeru rundy
* złączenie w klucz 56-bitowy
* redukcja 56-bitowego klucza do 48-bitowego z pomocą permutacji,
* zapamiętanie wyniku jako i-ty element 16-elementowej tablicy.

private **Block48**[] generateRoundKeys(**Key** key) {  
 **Block48**[] generatedKeys = new Block48[*numberOfRounds*];  
 **Block56** key56 = dropParityBits(new Block(key.getBytes()));  
 for (int i = 0; i < *numberOfRounds*; i++)  
 {  
 for (int j = 0; j < *numberOfKeyShiftsPerRound*[i]; j++)  
 rotateBitsLeft(key56);  
  
 generatedKeys[i] = compressKey(key56);  
 }  
 return generatedKeys;  
}

Pojedynczy algorytm DES z pomyślnością działa na 64-bitowym bloku danych, dlatego dla większych plików konieczne jest podzielnie plików na 64-bitowe bloki danych i wykonanie na każdym z nich operacji algorytmu DES. Kolejnymi operacjami algorytmu DES są:

* wstępna permutacja,
* podział bloku 64-bitowego na 32-bitowe połowy,
* operacje na rundzie (analogiczne dla każdej z 16 rund):

private void round(**BlockHalf** leftBlock32, **BlockHalf** rightBlock32, **Block48** key)  
{  
 **BlockHalf** copyRightBlock32 = new BlockHalf();  
 copyRightBlock32.blocks = rightBlock32.blocks.clone();  
 FFunction(rightBlock32, key);  
  
 for (int i = 0; i < 4; i++)  
 leftBlock32.blocks[i] ^= rightBlock32.blocks[i];  
  
 rightBlock32.blocks = leftBlock32.blocks.clone();  
 leftBlock32.blocks = copyRightBlock32.blocks.clone();  
}

* + operacje funkcji F na prawej połowie 64-bitowego bloku danych:

private void FFunction(**BlockHalf** block32, **Block48** key48)  
{  
 **Block48** block48 = expandTo48Bits(block32);  
 for (int i = 0; i < block48.blocks.length; i++)  
 block48.blocks[i] ^= key48.blocks[i];  
  
 **Block8**[] block6Table = split48BitBlockTo8GroupsPer6BitBlocks(block48);  
 **Block8**[] block4Table = new Block8[8];  
 for (int i = 0; i < block4Table.length; i++)  
 {  
 block4Table[i] = new Block8();  
 block4Table[i].blocks[0] = substitution(i, block6Table[i]).blocks[0];  
 }  
 merge4BitBlocksInto32BitBlock(block4Table, block32);  
 permutationP(block32);  
}

* + - rozszerzenie 32-bitowgo bloku danych do 48 bitów,
    - operacja xor 48-bitowego bloku danych i 48-bitowego podklucza,
    - podział 48-bitów na 8 grup po 6-bitów w celu obliczenia wartości na podstawie S-boxów,
    - permutacja
  + operacja xor 32-bitowej lewej części z 32-bitową prawą częścią po operacjach F-funkcji,
* rozpoczęcie kolejnej rundy, gdzie wynik poprzedniej to jego prawa połowa, a oryginalna prawa część to jego lewa część.

Funkcjonalności programu pozwalają na szyfrowanie/deszyfrowanie plików z zapisem szyfrogramu/tekstu jawnego do pliku oraz szyfrowanie/deszyfrowanie wiadomości tekstowych w oknach tekstowych graficznego interfejsu użytkownika.

1. **Wnioski**

W porównywaniu do algorytmu DES algorytm 3DES przy prostym w implementacji zwiększeniu ilości wykonywanych operacji daje korzystne efekty względem trudności odszyfrowania. Jednak ze względu za zwielokrotnienie tego samego algorytmu szyfrującego staje się podatny na ataki meet in the middle. W przypadku użycia takiego samego klucza dla 3 etapów działania algorytmu DES, tworzących 3DES, algorytm 3DES zrównuje się pod względem bezpieczeństwa ze standardowym algorytmem DES, dlatego warta uwagi jest kontrola reperacji w trzech kluczach wejściowych. Ryzyko to jednak sprawia, że algorytm 3DES jest kompatybilny z algorytmem DES.