Jakub Mielczarek 236602
Hubert Tasarz 236674

Podstawy Kryptografii

8:30-10:00 Środa
21.06.2022r

1. Cel ćwiczenia

Celem ćwiczenia było wykonanie kodu realizującego szyfrowanie i deszyfrowanie algorytmem DESX.

2. Realizacja

Algorytm szyfrujący DESX jest algorytmem bazującym na algorytmie DES z tą różnicą, że dodatkowo dane są szyfrowane jeszcze 2 razy(na wejściu i wyjściu z algorytmu DES`a). Dzięki wykorzystaniu aż 3 kluczy znacznie utrudniamy złamanie klucza używając ataku brute-force.
Algorytm DESX:

- Szyfruje 64bitowe bloki za pomocą 56bitowego klucza
- Przed pierwszą i po ostatniej rundzie bity są permutowane
- Żeby uzyskać podklucze usuwamy z 64bitowego klucza 8 bitów parzystości
- Z pozostałych 56bitów tworzymy 16 podkluczy – każdy 48bitowy, każdy inny
- Blok danych dzielimy na 2 części – prawą i lewą po 32bity
- Prawą stronę modyfikujemy przez funkcję f
    - 32 bity prawej strony permutowane do 48bitów
    - Xor z kluczem
    - Otrzymane 48 bitów dzielone na 8 grup po 6
    - Przejście przez SBoxy każdej z grupy
    - Z każdego SBoxa wychodzą 4 bity
    - Złączenie 8 wyników daje nam 32bitową prawą stronę
- Prawą xorujemy z lewą i wynik staje się nową prawą stroną, poprzednia prawa staje się lewą.
- Przechodzimy w taki sposób przez 16 rund
- Po 16 iteracji zamieniamy ze sobą strony a otrzymany blok jest podawany permutacji końcowej czyli takiej odwrotnej do początkowej co pozwoli nam później na deszyfrowanie
- Deszyfrowanie polega na wykonaniu tych samych operacji, zmienia się jedynie kolejność kluczy.

3. Wygląd programu

Nasz program ma interfejs graficzny, możemy wpisać ręcznie fragment tekstu lub wybrać plik który chcemy zaszyfrować. Po zaszyfrowaniu plik jest zapisywany w katalogu programu z nazwą „NazwaSzyfrowanegoPlikuEncrypted”. Później możemy wybrać plik do deszyfrowania, który jest zapisywany w katalogu programu z nazwą „NazwaSzyfrowanegoPlikuDecrypted”.

4. Kod

Xorowanie

```
public static byte[] XORBytes(byte[] a, byte[] b)
{
    byte[] result = new byte[a.length];
    for (int i = 0; i < a.length; i++)
    {
        result[i] = (byte) (a[i] ^ b[i]);
    }
    return result;
}
```

Generowanie podkluczy

```
public byte[][] getSubkeys()
{
    byte[] activeKey = Permutation.permutation(this.key, this.PC1);
    int halfKeySize = this.PC1.length / 2;  //28 bitów
    byte[] c = BitOperations.selectBits(activeKey, 0, halfKeySize);
//pierwsza połowa permutowanego klucza
    byte[] d = BitOperations.selectBits(activeKey, halfKeySize,
halfKeySize); //druga połowa permutowanego klucza
    byte[][] subKeysLocal = new byte[16][];  // tablica 16 podkluczy
    for (int k = 0; k < 16; k++)
    {
        c = BitOperations.shiftLeft(c, this.SHIFTS[k]);
        d = BitOperations.shiftLeft(d, this.SHIFTS[k]);
        byte[] cd = BitOperations.joinTabBytes(c, d);
        try {
            subKeysLocal[k] = Permutation.permutation(cd, this.PC2);
        } catch (RuntimeException e)
        {
            System.out.println(e.getCause());
        }

    }
    return subKeysLocal;
}
```

DES

```
//szyfrowanie bloku 64 bitowego
public byte[] crypt(byte[] block, byte [][] subkeys, boolean ifE)
{
    //Permutowanie danych wejściowych i rozdzielenie ich na dwa bloki 32
bitowe
    byte[] encryptedBlock = Permutation.permutation(block, IP);
    int halfKeySize = this.IP.length / 2;  //32 bity
    byte[] L = BitOperations.selectBits(encryptedBlock, 0, halfKeySize);
//pierwsza połowa permutowanego bloku
    byte[] R = BitOperations.selectBits(encryptedBlock, halfKeySize,
halfKeySize); //druga połowa permutowanego bloku


    for (int k = 0; k < 16; k++)
```

```
    {
        byte[] R48 = Permutation.permutation(R, E);  //Dopełniamy prawy
blok do 48 bitów
        if (ifE == true)
            R48 = XOR.XORBytes(R48, subkeys[k]);
        else
            R48 = XOR.XORBytes(R48, subkeys[15 - k]);
        R48 = sBlocks(R48);
        R48 = Permutation.permutation(R48, P);
        R48 = XOR.XORBytes(L,R48);
        L = R;
        R = R48;
    }
    byte[] RL = BitOperations.joinTabBytes(R, L);
    return Permutation.permutation(RL, IP1);
}


public byte[] desEncode(byte[] message, byte [][] subkeys)
{
    byte[] result = tabTransformation.fillMessage(message);
    byte[] tempBlock;
    // Pętla tnąca i kodowanie bloków
    for (int i = 0; i < (result.length / 8); i++)
    {
        tempBlock = crypt(BitOperations.oneBlock(result, i * 8),subkeys,
true);
        System.arraycopy(tempBlock, 0, result, i * 8, 8);
    }
    return result;
}


public byte[] desDecode(byte[] encrypted, byte [][] subkeys)
{
    byte[] tmpResult = new byte[encrypted.length];
    byte[] tempBlock;
    for (int i = 0; i < (encrypted.length / 8); i++)
    {
        tempBlock = crypt(BitOperations.oneBlock(encrypted, i * 8),subkeys,
false);
        System.arraycopy(tempBlock, 0, tmpResult, i * 8, tempBlock.length);
    }
    return tabTransformation.cutDecrypted(tmpResult);

}
```

DESX

```
public byte[] DESX(byte[] message, byte [] key1, byte [] key2, byte []
key3, boolean ifE) {
    Des des = new Des();
    byte[] step1;
    KeysGenerator subkeysGen = new KeysGenerator(key2);
    byte[][] subkeys = subkeysGen.getSubkeys();
    byte[] step2;
    byte[] result;

    if (ifE == true) {
        //xorowanie z kluczem I
```

```
        step1 = XOR.xorEncode(message, key1);
        //szyfrowanie z kluczem II
        step2 = des.desEncode(step1, subkeys);
        //xorowanie z kluczem II
        result = XOR.xorEncode(step2, key3);
    }
    else
    {
        //xorowanie z kluczem I
        step1 = XOR.xorDecode(message, key1);
        //deszyfrowanie z kluczem II
        step2 = des.desDecode(step1, subkeys);
        //xorowanie z kluczem II
        result = XOR.xorDecode(step2, key3);
    }
    return result;
}
```

SBOXy

```
private byte[] sBlocks(byte[] data)
{
    byte row;
    byte col;
    data = BitOperations.create6BitData(data);
    byte[] result = new byte[data.length / 2];  //4 bajty
    byte lowerHalfByte = 0;
    byte halfByte;
    for (int b = 0; b < data.length; b++)  // b to numer sBoxa
    {
        row = (byte) (((data[b] >> 6) & 2) | ((data[b] >> 2) & 1));
        col = (byte) ((data[b] >> 3) & 15);
        halfByte = sBox[64 * b + 16 * row + col];  //z każdego sBoxa
bierzemy liczbę znajdującą się w danym rzędzie i danej kolumnie
        if (b % 2 == 0)
            lowerHalfByte = halfByte;
        else
            result[b / 2] = (byte) (16 * lowerHalfByte + halfByte);
    }
    return result;
}
```

5. Podsumowanie

   Udało nam się stworzyć kod, który poprawnie szyfruje i deszyfruje zarówno wiadomości tekstowe jak i różnego rodzaju pliki.