Kryptografia - lista 2

Oskar Makowski 236554

25 Kwietnia 2020

1 DES vs AES

DES jest szyfrem blokowym wykonującym 16 rund sieci Feistela o rozmiarze bloku 64 bity i kluczu długości 54 bitów. Sieć Feistela wykorzystuje wewnętrzną funkcję f, która nie jest odwracalna. W każdej rundzie używa się tej samej funkcji f, zmieniając klucz rundy, każdy długości 48 bitów, stanowiący podzbiór klucza głównego.

AES korzysta z bloku rozmiaru 128 bitów, długości klucza mogą wynosić 128, 192 lub 256 bitów. Używa sieci permutacyjno-podstawieniowej(ang. *substitution-permutation network*). Operacje przeprowadzane są na macierzy stanu wymiaru $4x4$ bajty. Liczba rund zależy od długości użytego klucza. Każda runda składa się z 4 kroków:

- Dodaj klucz rundy
- Zamiana bajtów
- Zamiana wierszy
- Mieszanie kolumn

2 Tryby pracy szyfrów blokowych

2.1 CBC

Wymagane jest stworzenie losowego wektora początkowego(IV) długości n. Pierwszy blok szyfrogramu powstaje przez zastosowanie pseudolosowej permutacji do $IV \oplus m_1$, czyli XORa pierwszego bloku tekstu jawnego i IV. Oznaczmy $c_0 = IV$. Wtedy i-ty blok szyfrogramu powstaje według wzoru: $c_i = F_k(c_{i-1} \oplus m_i)$. Szyfrogram jest postaci $< IV, c_1, \ldots, c_l >$. Aby było możliwe przeprowadzenie deszyfrowania, potrzebne jest zawarcie IV w postaci wynikowej.

Szyfrowanie w tym trybie musi odbywać się sekwencyjnie, ponieważ blok c_i jest potrzebny do zaszyfrowania bloku $i+1$ wiadomości m. Deszyfrowanie może odbywać się równolegle.

2.2 OFB

Najpierw wybierany jest losowy wektor $IV \leftarrow 0,1^n$, następnie niezależnie od szyfru jawnego strumień zdefiniowany następująco: $r_0 = IV, r_i = F_k(r_{i-1})$. Bloki szyfrogramu powstają według reguły: $c_i = m_i \oplus r_i$. IV zawiera się w postaci wynikowej, analogicznie do trybu CBC.

Szyfrowanie i deszyfrowanie musi odbywać się sekwencyjnie. Jednakże przygotowanie strumienia wyrazów r_i odbywa się niezależnie i może zostać przeprowadzone jako pre-processing, po którym sam akt szyfrowania jest już szybki.

2.3 CTR

Najpierw wybierany jest losowy wektor $IV \leftarrow 0,1^n$, często oznaczany jako *ctr*. Następnie generowany jest strumień $r_i = F_k(ctr + i)$, gdzie dodawanie odbywa się modulo 2^n. i-ty blok szyfrogramu jest wyznaczany jako $c_i = r_i \oplus m_i$.

Szyfrowanie i deszyfrowanie może zostać zrównoleglone, strumień r_i może zostać przygotowany jako pre-processing.

3 Zadanie 1

Program wspiera tryby szyfrowania przedstawione w sekcji 2. Pozwala utworzyć *keystore* lub wczytać istniejący(wbudowana biblioteka *security* w jezyku Java). Ważne, by utworzyć *keystore* w formacie *.jceks, pozwalającym przechowywać klucze symetryczne. Do *keystore'a* można dodać klucz lub użyć istniejący. Klucz składa się z aliasu, klucza właściwego i hasła zabezpieczającego sam klucz. Ostatecznie, wybiera się jeden z trzech trybów:

- encryption oracle - pozwalający wczytać q wiadomości, których szyfrogramy zostaną zapisane do pliku *encryption.txt*
- challenge - zwracany jest losowo wybrany szyfrogram spośród dwóch wczytanych wiadomości
- decryption - weryfikacja odwracalności procesu szyfrowania, wczytanie szyfrów z pliku *encryption.txt* i zapisanie zdekodowanych teksów jawnych w pliku *decryption.txt*

Dla wygody pliki *encryption.txt* i *decryption.txt* znajdują się wewnątrz folderu projektu.

Na rysunku 1 pokazano pojedynczy przebieg programu. Na rysunku 2 widać stworzony *keystore* w formacie **.jceks*. Na rysunku 3 pokazano uzyskane szyfrogramy, zakodowane dodatkowo za pomocą Base64.

4 Zadanie 2

Celem jest pokazanie, że AES w trybie CBC, w którym kolejne IV generowane są w przewidywalny sposób, nie jest odporny na atak typu CPA. Jak opisano

```
Modes of encryption: OFB, CTR, CBC
Choose mode: CBC
Keystore options:
[1] - create
[2] - load
Option: 1
Keystore name: test.jceks
Keystore password: test
Key management options:
[1] - create key and use it
[2] - get key
Option: 1
Key alias: alias
Key: testtesttesttest
Key password: alias
Program modes:
[1] - encryption oracle
[2] - challenge
[3] - decryption
Choose mode: 1
Number of messages to encrypt: 3
1: ala
2: ma
3: kota
IV: 1234567890123456

Process finished with exit code 0
```

Rysunek 1: Program kolejno prosi o podanie niezbędnych parametrów. Przedstawiono proces tworzenia repozytorium kluczy, dodania nowego klucza pod aliasem *alias*(należy samodzielnie zadbać o odpowiednią długość), wprowadzenia 3 wiadomości do zaszyfrowania i wektora początkowego o odpowiednim rozmiarze.

Rysunek 2: Utworzony *keystore*.

uWNuyvLHLi7KxqEZFfMsbw==
SI7v8RwNw+LzgCkixIReUw==
lIifmKQWL1bfMa7nrsIGSw==

Rysunek 3: Szyfrogramy

w sekcji 2, IV jest częścią otrzymanego szyfrogramu. Mając dany stan IV i kontrolując liczbę generowanych szyfrogramów, adwersarz zna kolejne wartości IV używane do szyfrowania następnych wiadomości. Przygotowanie ataku polega na skorzystaniu z jawnej informacji jak działa tryb CBC, a także użycie "przyszłych" wartości IV, uzyskanych z przewidywalnego generatora(można, jak to zasugerowano, założyć że kolejne wartości zwiększają się o stałą wartość, np. o 1).

Załóżmy, że mamy dwóch aktorów, Ewę i Alicję. Celem Ewy jest dowiedzenie się czegoś o Alicji. Dla uproszczenia załóżmy, że jest to informacja binarna typu *true/false*. Ewa przygotowuje tekst jawny postaci

$$P_{Ewa} = IV_{Ewa} \oplus IV_{Alicja} \oplus "false"$$

Powstały szyfrogram przybierze w rozpatrywanym modelu postać

$$C_{Ewa} = Enc(IV_{Ewa} \oplus P_{Ewa}) = Enc(IV_{Ewa} \oplus IV_{Ewa} \oplus IV_{Alicja} \oplus "false")$$

Redukując, zostanie $C_{Ewa} = Enc(IV_{Alicja} \oplus "false")$. Teraz Ewa może porównać C_{Ewa} z C_{Alicja}. Jeśli są różne, informacja Alicji ma wartość *true*, jeśli takie same - *false*.

Na rysunku 4 pokazano fragment kodu wykonujący przedstawioną wyżej operację. W linijce 18 i 19 znajdują się przygotowane wartości IV, ręcznie wstawiane do klasy szyfrującej. Linijka 23 i 24 to przygotowanie tekstu jawnego wysyłanego przez Ewę. Pętla w 31 linijce wypisuje uzyskane bajty. Następnie symulowane jest zapytanie Alicji. Linijka 38 służy do wyrównania długości tekstów jawnych. Pętla w 42 linijce wypisuje uzyskane bajty w przypadku Alicji. Rezultat znajduje się na rysunku 5.

```
try
{
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");

    byte[] firstIV = new byte[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6};
    byte[] secondIV = new byte[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7};

    byte[] b = "false".getBytes();

    byte[] plaintext = xor(firstIV, secondIV);
    plaintext = xor(plaintext, b);

    IvParameterSpec iv = new IvParameterSpec(firstIV);
    cipher.init(Cipher.ENCRYPT_MODE, key, iv);

    byte[] cipherEve = cipher.doFinal(plaintext);

    for(int i = 0; i < cipherEve.length; i++)
    {
        System.out.print(cipherEve[i] + " ");
    }

    IvParameterSpec iv2 = new IvParameterSpec(secondIV);
    cipher.init(Cipher.ENCRYPT_MODE, key, iv2);
    byte[] plaintextAlice = xor(new byte[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "false".getBytes());
    byte[] cipherAlice = cipher.doFinal(plaintextAlice);

    System.out.println();
    for(int i = 0; i < cipherAlice.length; i++)
    {
        System.out.print(cipherAlice[i] + " ");
    }
}
```

Rysunek 4: Fragment kodu implementujący przedstawiony atak teoretyczny.

```
Keystore options:
[1] - create
[2] - load
Option: 2
KeyStore file name: test.jceks
Password: test
Key management options:
[1] - create key and use it
[2] - get key
Option: 2
Key alias: alias
Key password: alias
81 -5 39 101 -59 -62 125 -54 3 74 -84 97 -127 -61 89 -41 50 7 -61 64 -111 16 -17 60 -37 -105 27 13 -110 -126 -33 -94
81 -5 39 101 -59 -62 125 -54 3 74 -84 97 -127 -61 89 -41 50 7 -61 64 -111 16 -17 60 -37 -105 27 13 -110 -126 -33 -94
Process finished with exit code 0
```

Rysunek 5: Rezultat ataku.