Podstawowe zagadnienia kryptograficzne
Tryby pracy szyfrów blokowych

Anna Dziuba, 146393

1. Przeanalizuj dostępne tryby pracy szyfrów blokowych w wybranym środowisku programowania i zmierz czasy szyfrowania i deszyfrowania dla 3 różnej wielkości plików we wszystkich 5 podstawowych trybach ECB, CBC, OFB, CFB, i CTR. Zinterpretuj otrzymane wyniki.

Rozmiar pliku	Tryb	Czas szyfrowania [s]	Czas deszyfrowania [s]
10 KB	ECB	0,0010	0,0023
	CBC	0,0021	0,0035
	CFB	0,0260	0,0290
	OFB	0,0030	0,0040
	CTR	0,0010	0,0020
1 MB	ECB	0,0120	0,0115
	CBC	0,0260	0,0280
	CFB	0,2446	0,2282
	OFB	0,0280	0,0310
	CTR	0,0136	0,0310
100 MB	ECB	0,1177	0,1172
	CBC	0,2525	0,2712
	CFB	2,256	2,1822
	OFB	0,2617	0,2710
	CTR	0,1283	0,1355

ECB i CTR są trybami najszybszymi dla wszystkich analizowanych plików, zarówno dla szyfrowania i deszyfrowania. CFB jest zdecydowanie najwolniejszy we wszystkich przypadkach. Dla małego pliku różnice czasów są niewielkie, jednak mimo to CFB jest około 10 razy wolniejszy od ECB/CTR. Wraz ze wzrostem rozmiaru pliku czasy szyfrowania i deszyfrowania rosną, ale także różnice pomiędzy poszczególnymi trybami stają się większe. Dla największego analizowanego pliku, najwolniejszy CFB jest około 22 razy wolniejszy od najszybszych ECB/CTR.

2. Przeanalizuj propagację błędów w wyżej wymienionych trybach pracy. Czy błąd w szyfrogramie będzie skutkował niemożnością odczytania po deszyfrowaniu całej wiadomości, fragmentu, ..? Zinterpretuj wyniki obserwacji.

Zaimplementowano program, który testuje propagację błędów w różnych trybach szyfrowania AES. Dla każdego trybu szyfruje dane, wprowadza błąd w pierwszym bajcie szyfrogramu, a następnie porównuje deszyfrowany tekst oryginalny z deszyfrowanym tekstem po modyfikacji.

```
Tryb: ECB
Oryginał:  To jest zdanie testowe do analizy propagacji bledow.
Po błędzie: &9␀PS␀U-0~estowe do analizy propagacji bledow.

Tryb: CBC
Oryginał:  To jest zdanie testowe do analizy propagacji bledow.
Po błędzie: /c␀H1␀Qv␀dstowe do analizy propagacji bledow.

Tryb: CFB
Oryginał:  To jest zdanie testowe do analizy propagacji bledow.
Po błędzie: Uo jest zdanie t{qgN7g␀P0ay propagacji bledow.

Tryb: OFB
Oryginał:  To jest zdanie testowe do analizy propagacji bledow.
Po błędzie: Uo jest zdanie testowe do analizy propagacji bledow.

Tryb: CTR
Oryginał:  To jest zdanie testowe do analizy propagacji bledow.
Po błędzie: Uo jest zdanie testowe do analizy propagacji bledow.
```

- W trybie ECB pierwszy blok danych jest uszkodzony. Zmiana pierwszego bajtu w szyfrogramie powoduje zmianę całego bloku, ale kolejne bloki pozostają nienaruszone.
- W trybie CBC uszkodzony jest pierwszy blok danych, a także część drugiego bloku. Dzieje się tak, ponieważ CBC wprowadza zależności pomiędzy blokami, a więc błąd w pierwszym bloku propaguje się również do następnego bloku.
- W trybie CFB błąd propaguje się do kolejnych bloków, ale po czasie szyfr się synchronizuje.
- W trybach OFB i CTR błąd w pierwszym bajcie szyfrogramu powoduje zmianę tylko pierwszego bajtu w deszyfrowanym tekście. Reszta wiadomości pozostaje nienaruszona.

3. Zaimplementuj tryb CBC (korzystając z dostępnego w wybranym środowisku programowania trybu ECB).

```python
def encrypt(text, key, iv):
    text = pad(text.encode('utf-8'), AES.block_size)
    cipher_ecb = AES.new(key, AES.MODE_ECB)
    blocks = [text[i:i + AES.block_size] for i in range(0, len(text), AES.block_size)]

    ciphertext = b''
    previous_block = iv
    for block in blocks:
        xored_block = bytes([x ^ y for x, y in zip(block, previous_block)])
        encrypted_block = cipher_ecb.encrypt(xored_block)
        ciphertext += encrypted_block
        previous_block = encrypted_block

    return base64.b64encode(ciphertext).decode('utf-8')
```

Pierwszym krokiem do szyfrowania jest przygotowanie danych – kodowanie do postaci bajtów i dopełnienie do wielokrotności rozmiaru bloku AES (16 bajtów). Tworzony jest obiekt szyfrujący ECB, a następnie tekst jest dzielony na bloki po 16 bajtów. Proces szyfrowania jest przeprowadzany w pętli dla kolejnych bloków:

- pierwszy blok jest xor-owany z losowym wektorem inicjującym,
- wynik xor jest szyfrowany za pomocą ECB,
- kolejne bloki są xor-owane z poprzednim zaszyfrowanym blokiem, a następnie szyfrowane za pomocą ECB.

```python
def decrypt(ciphertext, key, iv):
    ciphertext = base64.b64decode(ciphertext)
    cipher_ecb = AES.new(key, AES.MODE_ECB)
    blocks = [ciphertext[i:i + AES.block_size] for i in range(0, len(ciphertext), AES.block_size)]

    plaintext = b''
    previous_block = iv
    for block in blocks:
        decrypted_block = cipher_ecb.decrypt(block)
        xored_block = bytes([x ^ y for x, y in zip(decrypted_block, previous_block)])
        plaintext += xored_block
        previous_block = block

    return unpad(plaintext, AES.block_size).decode('utf-8')
```

Pierwszym krokiem deszyfrowania jest zdekodowanie szyfrogramu do postaci binarnej. Podobnie jak w przypadku szyfrowania, tworzony jest obiekt szyfrujący, a szyfrogram jest dzielony na bloki. Proces deszyfrowania jest przeprowadzany w pętli dla kolejnych bloków:

- pierwszy blok szyfrogramu jest deszyfrowany za pomocą ECB,
- wynik deszyfrowania jest xor-owany z wektorem inicjującym, co daje nam pierwszy blok tekstu jawnego,
- kolejne bloki są deszyfrowane ECB, a następnie xor-owane z poprzednim blokiem szyfrogramu.

```
Oryginalna wiadomość: To jest przykładowy tekst
Zaszyfrowana wiadomość: xAHS4+0M0dnxvWMZQfzcr0Gl7GtEdhYM+zaSzANT5qU=
Odszyfrowana wiadomość: To jest przykładowy tekst
```