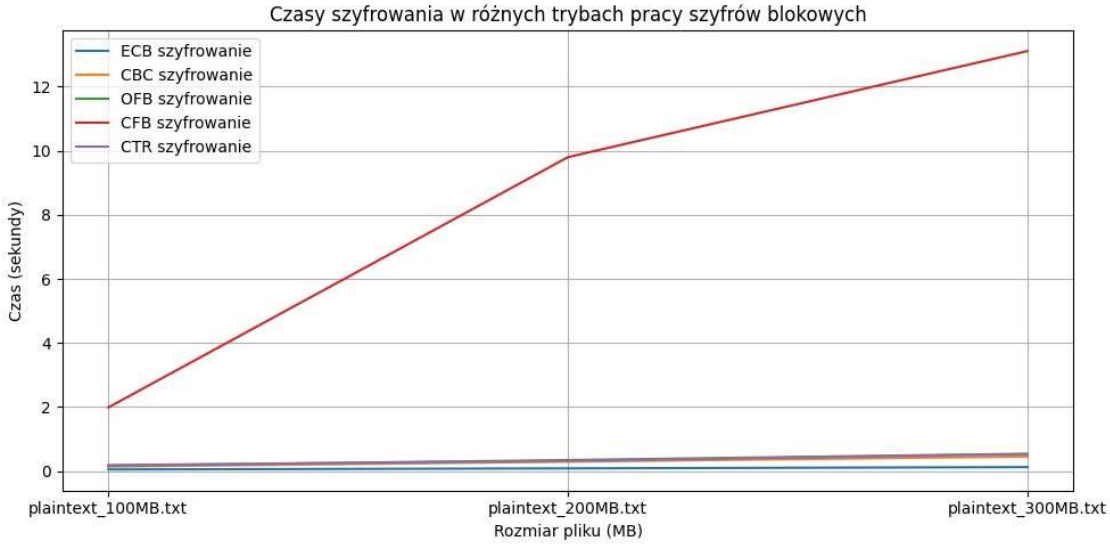
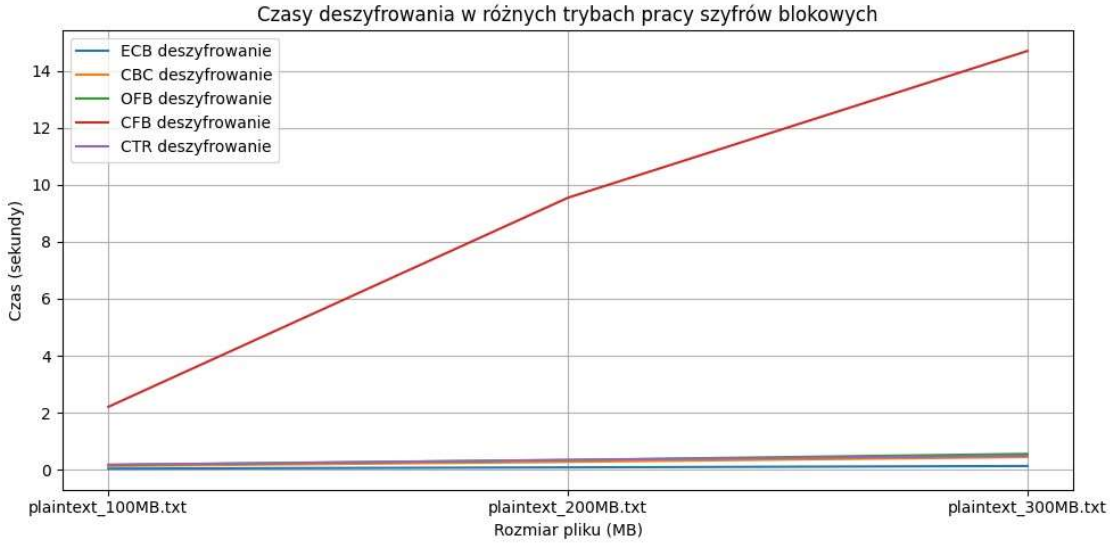
Sprawozdanie – Szyfry blokowe

Mateusz Kreczmer 151736

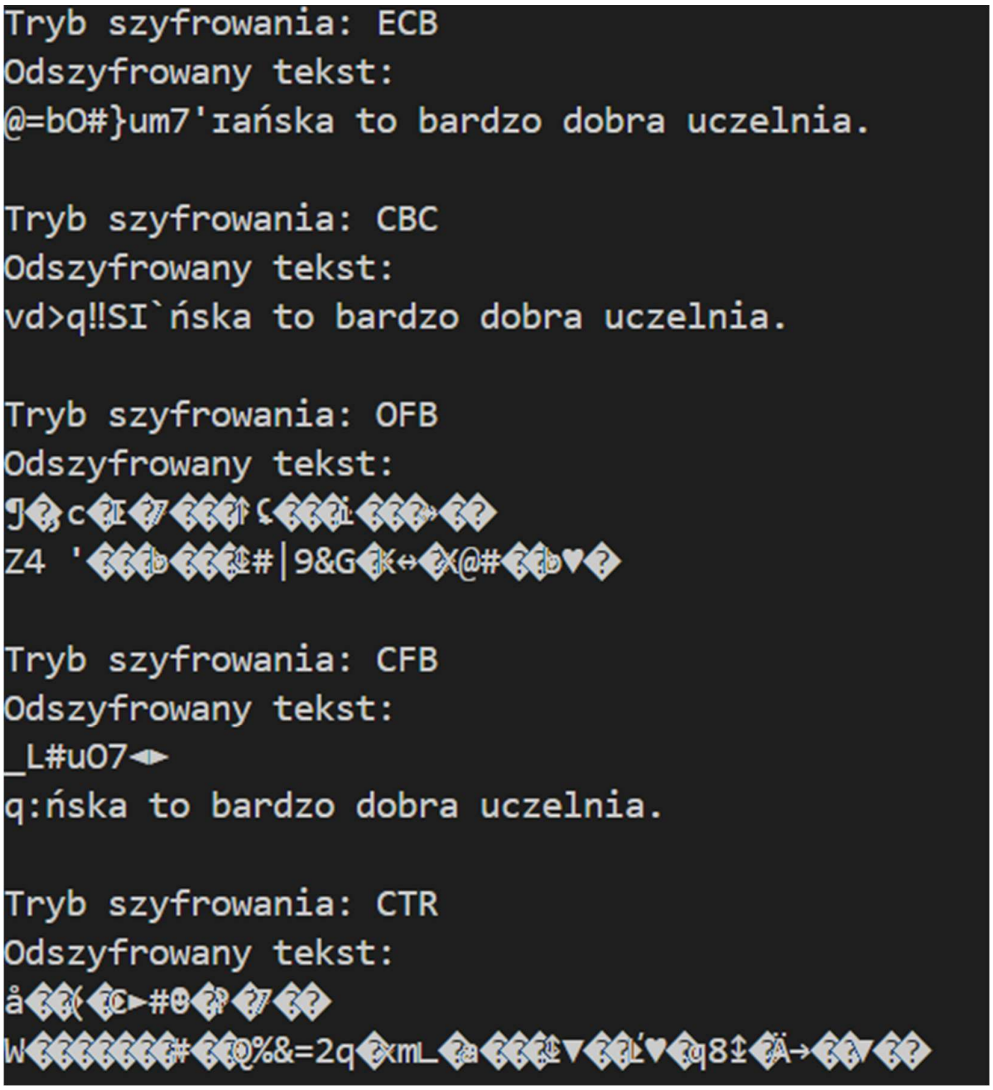
1. Przeanalizuj dostępne tryby pracy szyfrów blokowych w wybranym środowisku programowania i zmierz czasy szyfrowania i deszyfrowania dla 3 różnej wielkości plików we wszystkich 5 podstawowych trybach ECB, CBC, OFB, CFB, i CTR. Zinterpretuj otrzymane wyniki.





Szyfrowanie oraz deszyfrowanie w trybie CFB wykazuje znacznie dłuższy czas przetwarzania w porównaniu do pozostałych trybów pracy szyfrów blokowych (ECB, CBC, OFB, CTR) dla plików o rozmiarach 100 MB, 200 MB i 300 MB. Podczas gdy pozostałe tryby utrzymują czasy przetwarzania poniżej jednej sekundy dla każdej próbki danych, czas pracy trybu CFB rośnie do kilku sekund, osiągając wartości od 2 do nawet 14 sekund w zależności od rozmiaru pliku.

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.



Po wprowadzeniu błędu poprzez zmianę pierwszego bitu w zaszyfrowanym tekście można zauważyć, że w przypadku szyfrowań: ECB, CBC, jak i CFB jesteśmy w stanie odszyfrować większą część tekstu. Natomiast w przypadku szyfrowań OFB i CTR cały tekst jest niezdatny do odszyfrowania.

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

W poniższej implementacji tryb CBC działa poprzez wykonywanie operacji XOR pomiędzy każdym blokiem tekstu jawnego a poprzednim zaszyfrowanym blokiem przed zaszyfrowaniem. Ten schemat zapewnia lepsze właściwości bezpieczeństwa niż sam tryb ECB, ponieważ eliminuje powtarzalność dla tych samych bloków tekstu jawnego.

```
4. from Crypto.Cipher import AES
5. from Crypto.Util.Padding import pad, unpad
6. from Crypto.Random import get_random_bytes
7.
8. def cbc_encrypt(plaintext, key, iv):
9.     cipher = AES.new(key, AES.MODE_ECB)
10.    ciphertext = b''
11.    previous_block = iv
12.
13.    for i in range(0, len(plaintext), 16):
14.        block = plaintext[i:i+16]
15.
16.        block = bytes(x ^ y for x, y in zip(block, previous_block))
17.
18.        encrypted_block = cipher.encrypt(block)
19.
20.        ciphertext += encrypted_block
21.
22.        previous_block = encrypted_block
23.
24.    return ciphertext
25.
26.def cbc_decrypt(ciphertext, key, iv):
27.    cipher = AES.new(key, AES.MODE_ECB)
28.    plaintext = b''
29.    previous_block = iv
30.
31.    for i in range(0, len(ciphertext), 16):
32.        block = ciphertext[i:i+16]
33.
34.        decrypted_block = cipher.decrypt(block)
35.
36.        decrypted_block = bytes(x ^ y for x, y in zip(decrypted_block,
   previous_block))
37.
38.        plaintext += decrypted_block
39.
40.        previous_block = block
41.
42.    return plaintext
43.
44.plaintext = b"Politchnika Poznanska"
45.key = get_random_bytes(16)
```

```
iv = get_random_bytes(16)

ciphertext = cbc_encrypt(pad(plaintext, AES.block_size), key, iv)
print("CBC Encrypted:", ciphertext)

decrypted_text = unpad(cbc_decrypt(ciphertext, key, iv), AES.block_size)
print("CBC Decrypted:", decrypted_text.decode())

```

```
CBC Encrypted: b'Öxbf PÖxb7Öx08Öxb7=Öx10*UÖx8eÖxf6Öx92m@Öx8aÖx98Öx03Öx08fÖxabÖxb6Öxb2Öx08Öx87ÖrÖxe6ÖxecÖxd8ÖxfdÖx9dÖxf8Öx90'
CBC Decrypted: Politchnika Poznanska
```