Sprawozdanie

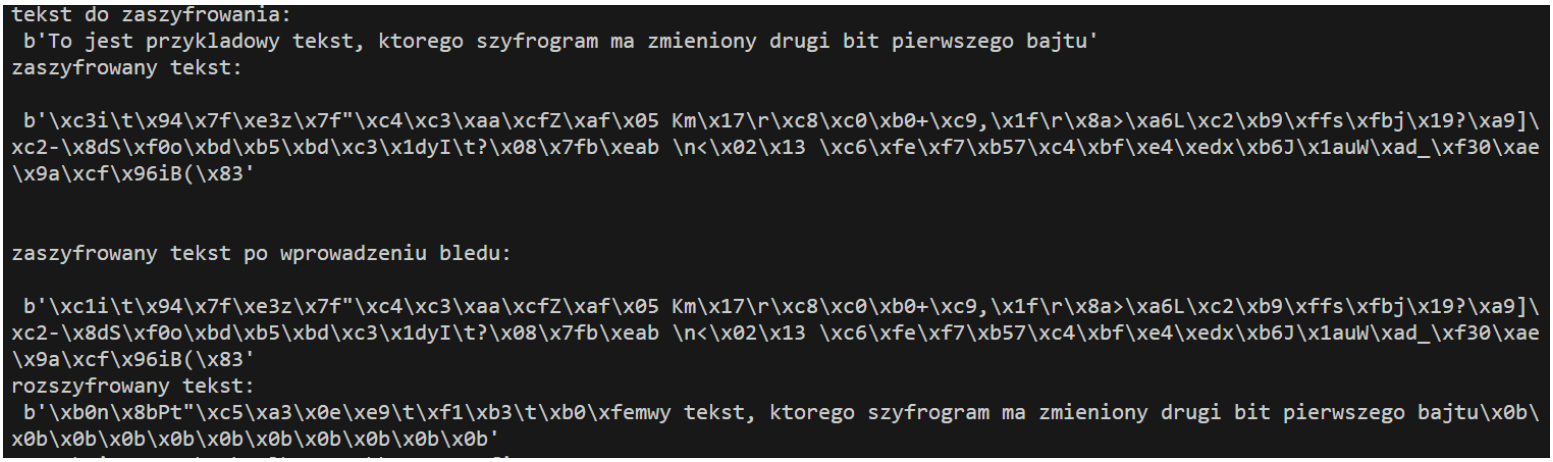
Szyfry Blokowe

Mikołaj Pluta 151827

1. Pomiar czasów:

Na podstawie wyników pomiarów czasów szyfrowania i deszyfrowania dla różnych trybów szyfrowania AES (ECB, CBC, CTR) dla danych o rozmiarach 10 MB, 100 MB i 1000 MB, można zauważyć, że tryb CBC (Cipher Block Chaining) zazwyczaj wymaga więcej czasu na szyfrowanie, szczególnie dla większych danych, niż pozostałe tryby.

2. Propagacja błędów.
   Badany tryb ECB, rozmiar bloku: 16 bajtów.



Jak widać na powyższym obrazie, pojawiające się błędy całkowicie uniemożliwiają odczytania bloku, w którym błąd się pojawił, jednakże błędy nie propagują się dalej. Nieprawidłowość w pierwszym bajcie, nie wpływa na wynik deszyfrowania reszty wiadomości.

3. Implementajca CBC.

Implementacja algorytmu CBC przy pomocy ECB wymaga przechowywania dodatkowego rejestru o długości bloku z którego korzysta algorytm ECB. Na jawnej informacji oraz tym rejestrze wykonywana jest operacja XOR, dopiero wynik tego działania szyfrowany jest dokładnie tak jak w ECB. Rejestr ten początkowo wypełniony jest wektorem inicjalizacyjnym IV, a następnie zastępuje go każdy kolejny wynik szyfrowania kolejnego bloku. Prosta implementacja w języku Python załączona na następnej stronie.

```python
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import os

def generate_random_bytes(size):
    return os.urandom(size)

def xor_bytes(a, b):
    return bytes(x ^ y for x, y in zip(a, b))

def encrypt_cbc(key, iv, plaintext):
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = b""
    previous_block = iv
    for i in range(0, len(plaintext), 16):
        block = plaintext[i:i+16]
        block_xor = xor_bytes(block, previous_block)
        encrypted_block = encryptor.update(block_xor)
        ciphertext += encrypted_block
        previous_block = encrypted_block
    return ciphertext + encryptor.finalize()

def decrypt_cbc(key, iv, ciphertext):
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
    decryptor = cipher.decryptor()
    plaintext = b""
    previous_block = iv
    for i in range(0, len(ciphertext), 16):
        block = ciphertext[i:i+16]
        decrypted_block = decryptor.update(block)
        decrypted_block_xor = xor_bytes(decrypted_block, previous_block)
        plaintext += decrypted_block_xor
        previous_block = block
    return plaintext


key = generate_random_bytes(32)
iv = generate_random_bytes(16)
plaintext = b"Sample plaintext to be encrypted using CBC mode."


ciphertext = encrypt_cbc(key, iv, plaintext)
print("Ciphertext (CBC mode):", ciphertext)

decrypted_plaintext = decrypt_cbc(key, iv, ciphertext)
print("Decrypted plaintext (CBC mode):", decrypted_plaintext)
```