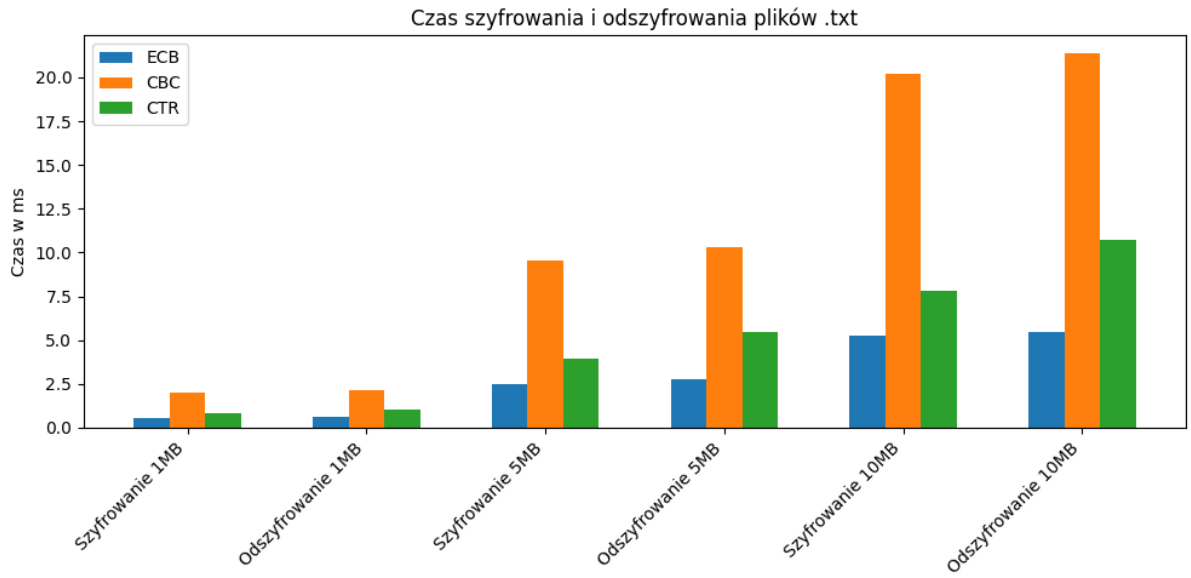
Sprawozdanie z Laboratorium 4

Mateusz Graja 155901

Wyniki pomiarów

Tryb	1 MB (enc / dec) [ms]	5 MB (enc / dec) [ms]	10 MB (enc / dec) [ms]
ECB	0,56 / 0,60	2,52 / 2,75	5,26 / 5,44
CBC	2,03 / 2,14	9,54 / 10,29	20,23 / 21,36
CTR	0,82 / 1,06	3,96 / 5,49	7,79 / 10,69



Interpretacja wyników:

- **ECB** – najprostszy tryb, brak dodatkowych operacji łańcuchowania, stosunkowo szybki i liniowy wzrost czasu wraz z rozmiarem danych.
- **CBC** – dodatkowe operacje XOR i pad/unpad spowalniają zarówno szyfrowanie, jak i odszyfrowanie, szczególnie przy większych plikach.
- **CTR** – tryb strumieniowy generuje strumień klucza niezależnie od bloków, przez co szyfrowanie jest równie szybkie jak w ECB, ale odszyfrowanie wymaga unpadowania i ponownego wyliczenia licznika, co nieco wydłuża czas.

Propagacja błędów

Do testu użyto 128-bajtowego ciągu

Błąd w szyfrogramie polegał na przestawieniu jednego bitu w 6. bajcie szyfrogramu. Następnie odszyfrowywano i zliczano różnice w bajtach wyjściowego tekstu jawnego.

Tryb	Liczba zmienionych bajtów po błędzie w szyfrogramie
ECB	15
CBC	17
OFB	1
CFB	17
CTR	1

Implementacja trybu CBC:

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

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

def encrypt_cbc(plaintext: bytes, key: bytes, iv: bytes) -> bytes:
    block_size = AES.block_size
    cipher_ecb = AES.new(key, AES.MODE_ECB)
    pt = pad(plaintext, block_size)
    ciphertext = b""
    prev = iv
    for i in range(0, len(pt), block_size):
        block = pt[i:i+block_size]
        x = xor_bytes(block, prev)
        enc = cipher_ecb.encrypt(x)
        ciphertext += enc
        prev = enc
    return ciphertext

def decrypt_cbc(ciphertext: bytes, key: bytes, iv: bytes) -> bytes:
    block_size = AES.block_size
    cipher_ecb = AES.new(key, AES.MODE_ECB)
    plaintext = b""
    prev = iv
    for i in range(0, len(ciphertext), block_size):
        block = ciphertext[i:i+block_size]
        dec = cipher_ecb.decrypt(block)
        pt_block = xor_bytes(dec, prev)
        plaintext += pt_block
        prev = block
    return unpad(plaintext, block_size)
```