Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Równoległa implementacja szyfrowania AES

Adam Krawczyk
Eryk Wawrzyn
Marcin Skrzypkowski
Numery albumów: 317678,291018, 283419

WARSZAWA 2021

Spis treści

tabularray

1. Szyfr AES

1.1. Historia

Na początku lat 90. stało się jasne, że potrzebny jest nowy standard w dziedzinie kryptografii. Wynikało to z faktu, że zarówno długość bloku (64 bity) jak i długość klucza (56 bitów) podstawowego algorytmu DES (wynalezionego w latach 70-tych) były zbyt małe dla przyszłych zastosowań (obecnie możliwe jest odzyskanie 56-bitowego klucza DES przy użyciu sieci komputerów lub specjalistycznego sprzętu). W odpowiedzi na ten problem amerykański Narodowy Instytut Norm i Technologii (NIST) zainicjował konkurs na nowy szyfr blokowy, który miał nosić nazwę Advanced Encryption Standard lub AES. W przeciwieństwie do procesu projektowania DES, który był utrzymywany w ścisłej tajemnicy, projekt AES został przeprowadzony publicznie. Wiele grup z całego świata przedstawiło projekty szyfru AES. Ostatecznie wybrano pięć algorytmów, znanych jako finaliści AES, które poddano szczegółowym badaniom.

Były to:

- MARS opracowany przez grupę z IBM,
- RC6 od grupy z RSA Security,
- Twofish od grupy z Counterpane, UC Berkeley i innych,
- Serpent od grupy trzech naukowców z Izraela, Norwegii i Wielkiej Brytanii,
- Rijndael od pary młodych belgijskich kryptografów (Vincent Rijmen i Joan Daemen).

Wreszcie jesienią 2000 roku NIST ogłosił, że ogólnym zwycięzcą AES został wybrany Rijndael.

1.2. Wstęp do zasad działania

DES i wszyscy finaliści AES są przykładami iterowanych szyfrów blokowych. Szyfry blokowe uzyskują swoje bezpieczeństwo poprzez wielokrotne użycie prostej funkcji zaokrąglania. Funkcja zaokrąglająca przyjmuje n-bitowy blok i zwraca n-bitowy blok, gdzie n jest rozmiarem bloku całego szyfru. Liczba rund r może być zmienna lub stała. Ogólną zasadą jest, że zwiększenie liczby rund zwiększa poziom bezpieczeństwa szyfru blokowego.

Aby umożliwić opis, każda runda musi być odwracalna. AES jest algorytmem z kluczem symetrycznym, co oznacza, że ten sam klucz jest używany zarówno do szyfrowania, jak i odszyfrowywania danych. AES ma stały rozmiar bloku 128 bitów i rozmiar klucza 128, 192 lub 256 bitów.

1.3. Definicje

1. AES - Advanced Encryption Standard.
2. Bit - Cyfra binarna mająca wartość 0 lub 1.

3. Block - Sekwencja bitów binarnych obejmująca wejście, wyjście, stan i klucz rundy. Długość sekwencji jest liczbą bitów, które zawiera.

4. Key Expansion - Metoda używana do generowania ciągu kluczy rund.

5. Round key - Wartość uzyskana z klucza szyfru za pomocą procedury rozwijania klucza. są stosowane do stanu w szyfrowaniu i deszyfrowaniu.

6. Cipher Key - Sekretny klucz użyty do generacji kluczy rund. Posiada cztery wiersze i Nk kolumn.

7. Key Expansion - Metoda generacji kluczy rund na podstawie klucza szyfru.

8. State - Pośredni efekt szyfrowania posiadający cztery wiersze i Nb kolumn.

9. S-box - Nie liniowa tablica podstawień

10. Word - Grupa 32 bitów.

11. Nb - Numer kolumn (32 bitowych słów). W tej implementacji Nb = 4.

12. Nk - Liczba słów składających się na klucz szyfru. Nk = 4, 6, 8.

13. Nr - Liczba rund szyfrujących. Im więcej rund tym lepsze zabezpieczenie używa się określonej liczby rund do określonej długości klucza. Nr = 10, 12, 14.

1.4. Opis funkcji

1. AddRoundKey() - Transformacja podczas szyfrowania i deszyfrowania. Klucz jest dodany poprzez operacje XOR.

2. MixColumns() - Transformacja podczas szyfrowania polegająca na pobraniu danych z kolumn i zamienieniu ich w ustalony sposób z innymi.

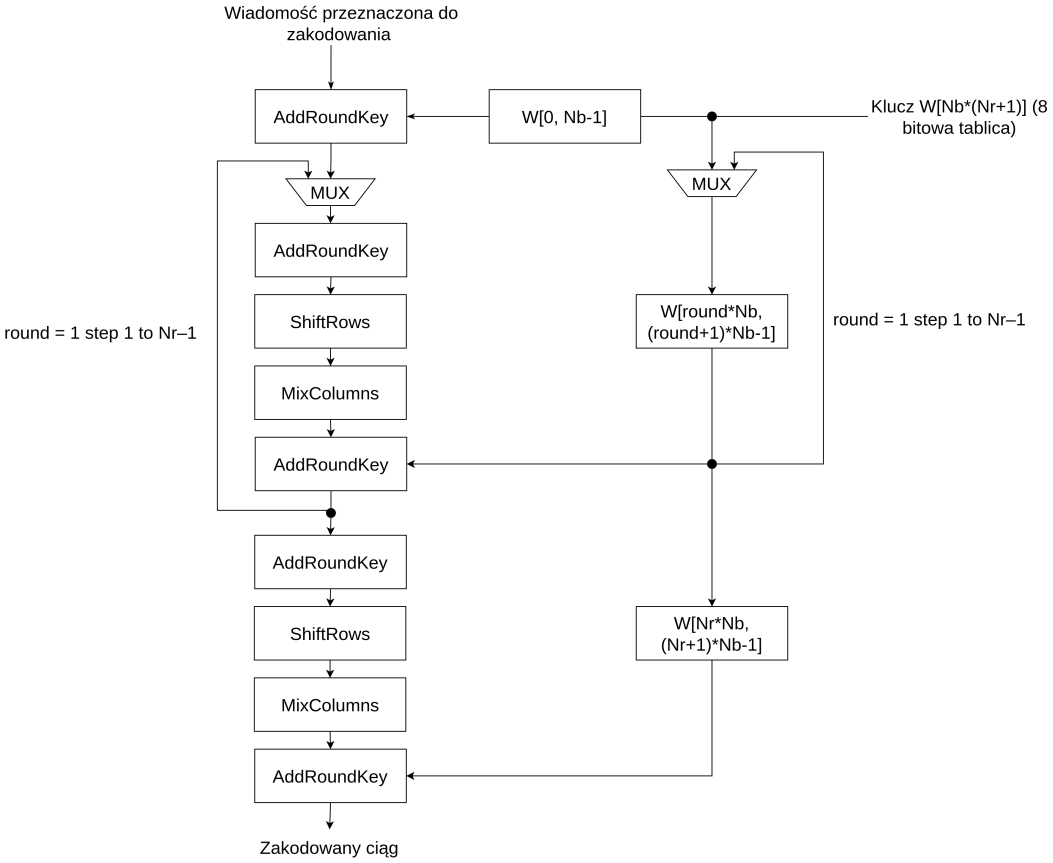
3. RotWord() - Funkcja używana podczas rozwijania klucza. Wykonuje permutacje na 4 bitowym słowie.

4. ShiftRows() - Cykliczna zamiana ostatnich trzech wierszy stanu.

1.5. Pseudocode

Listing 1. *Pseudokod szyfrowania*

```
1  Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
2  begin
3      byte state[4,Nb]
4      state = in
5
6      AddRoundKey(state, w[0, Nb-1])
7
8      for round = 1 step 1 to  N r 1
9          SubBytes(state)
10         ShiftRows(state)
11         MixColumns(state)
12         AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
13     end for
14
15     SubBytes(state)
16     ShiftRows(state)
17     AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
18     out = state
19
20 end
```

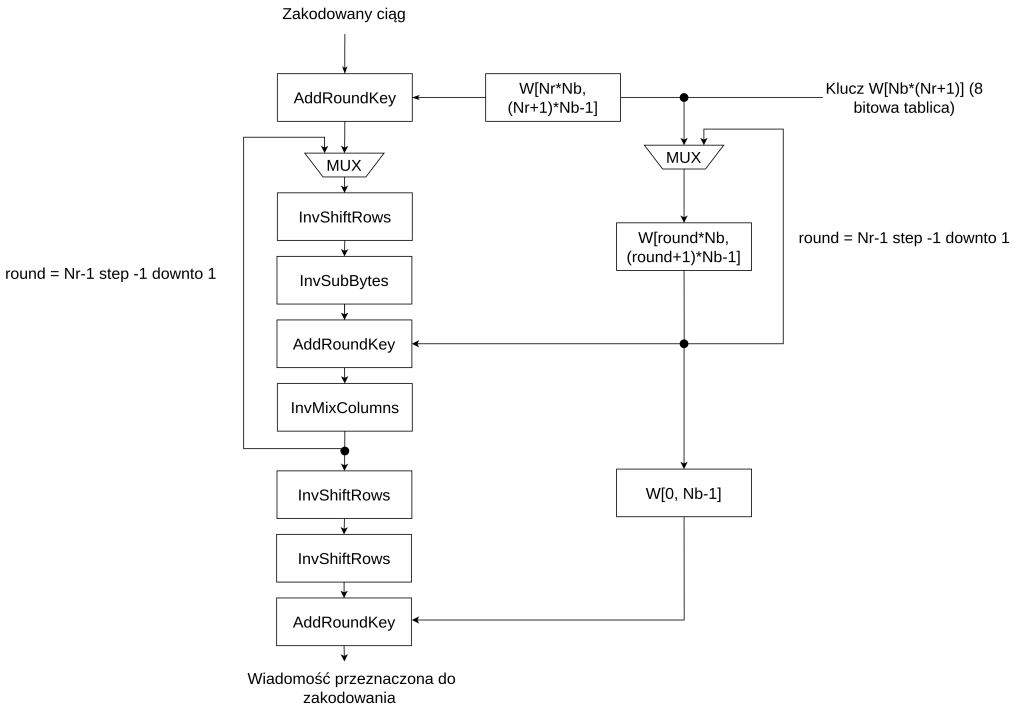


Rysunek 1.1. Schemat blokowy szyfrowania

Listing 2. *Pseudokod deszyfrowania*

```
1  InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
2  begin
3      byte state[4,Nb]
4      state = in
5      AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
6      for round = Nr-1 step -1 downto 1
7          InvShiftRows(state)
8          InvSubBytes(state)
9          AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
10         InvMixColumns(state)
11     end for
12
13 InvShiftRows(state)
14 InvSubBytes(state)
15 AddRoundKey(state, w[0, Nb-1])
16 out = state
17
```

18 end



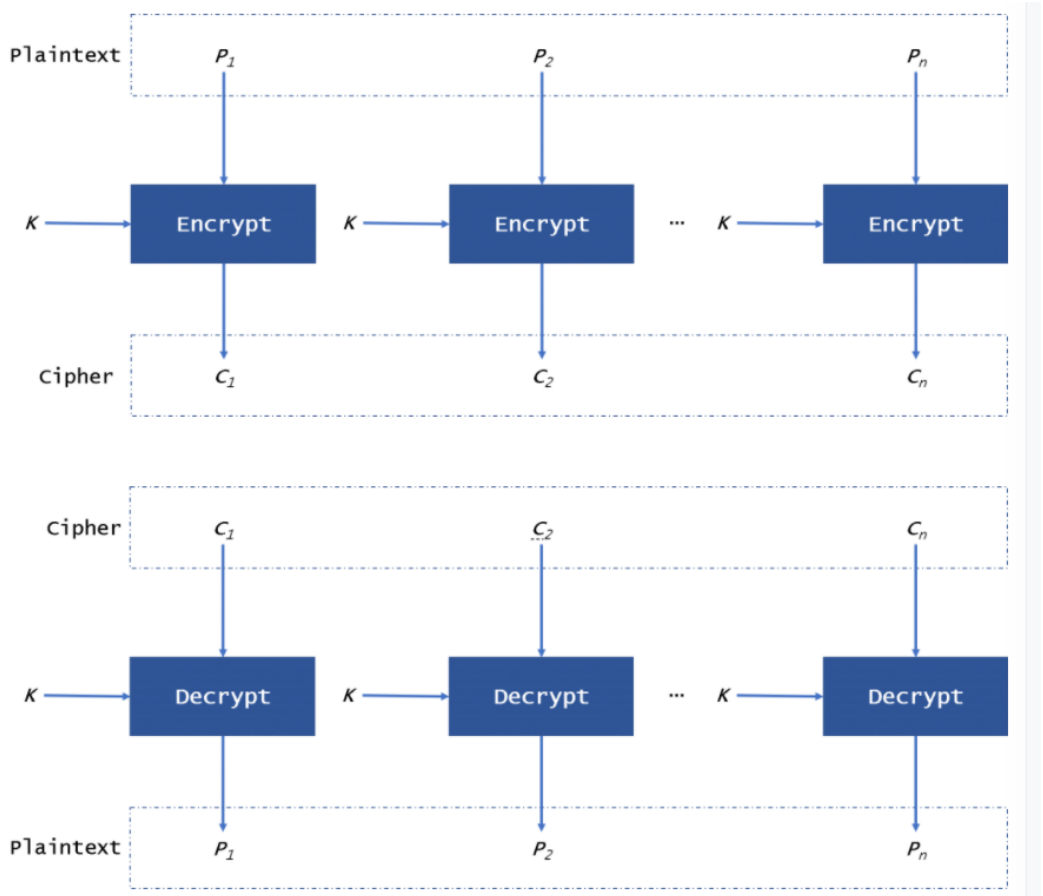
Rysunek 1.2. Schemat blokowy deszyfrowania

Listing 3. *Pseudokod rozwijania klucza*

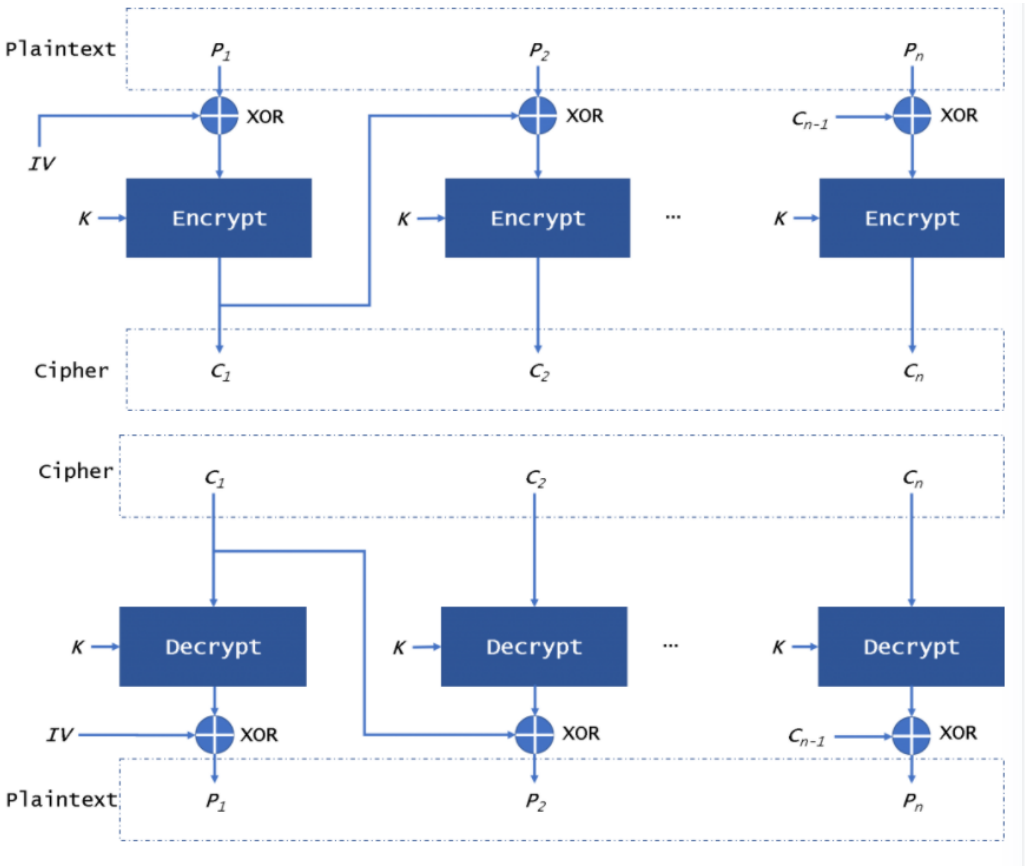
```
1  KeyExpansion(byte key[4 * Nk], word w[Nb * (Nr + 1)], Nk)
2  begin
3
4  word temp
5  i = 0
6
7  while (i < Nk)
8      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
9      i = i+1
10
11 end while
12
13 i = Nk
14
15 while (i < Nb * (Nr+1)]
16
17     temp = w[i-1]
18     if (i mod Nk = 0)
19         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
20     else if (Nk > 6 and i mod Nk = 4)
21         temp = SubWord(temp)
22     end if
23
24     w[i] = w[i-Nk] xor temp
25     i = i + 1
26 end while
27
28 end
```

2. Struktura kodu

W danym projekcie przyjęto, że zostaną zaimplementowane dwa "tryby" (ECB i CBC) algorytmu kryptograficznego AES. ECB jest podstawową wersją implementowanego algorytmu, nie wykorzystuje dodatkowych operacji logicznych, przez co posiada najmniejszą efektywność (zakodowany ciąg znaków jest podatny na rozszyfrowanie za pomocą ataku "brutalnego"). Na rys. 2.1 przedstawiono schemat blokowy trybu ECB (schemat bloku Encrypt i Decrypt przedstawiono odpowiednio na rys. 1.1 i 1.2). Każdy blok wiadomości jest szyfrowany i deszyfrowany tym samym kluczem i algorytmem, dzięki czemu każdy blok może być przetwarzany równolegle. A jeśli blok tekstu jawnego lub szyfrogramu jest uszkodzony, to nie będzie miało wpływu na inne bloki. Na rys. 2.2 przedstawiono schemat blokowy trybu CBC (schemat bloku Encrypt i Decrypt przedstawiono odpowiednio na rys. 1.1 i 1.2). Na każdym bloku wiadomości wykonywana jest dodatkowa operacja XOR. Dana operacja zwiększa bezpieczeństwo zakodowanej wiadomości, ale jednocześnie uzależnia następne bloki od poprzednich. Przetwarzanie różnych wiadomości w danym trybie nie może przebiegać równolegle, a błąd w jednej z wiadomości powoduje przekłamania w innych blokach.

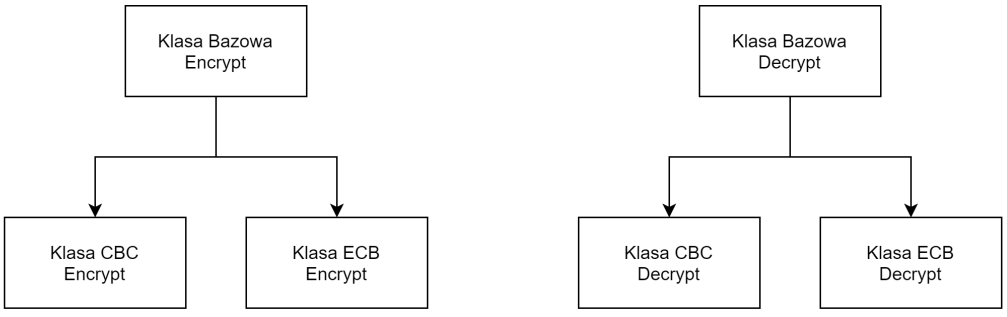


Rysunek 2.1. Schemat blokowy trybu ECB [1]



Rysunek 2.2. Schemat blokowy trybu CBC [1]

Na rys. 2.3 przedstawiono schemat blokowy, planowanej struktury kodu. Klasy bazowe będą posiadały podstawowe metody algorytmu AES. Natomiast klasy CBC i ECB będą dziedziczyły po klasie bazowej i dodatkowo będą posiadały metody umożliwiające wykonywanie algorytmu w danym trybie.

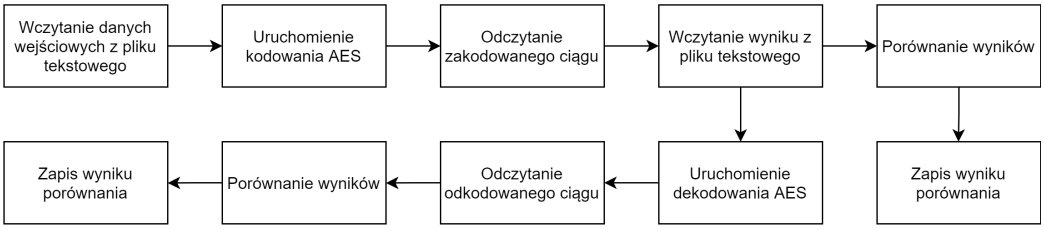


Rysunek 2.3. Schemat blokowy planowanych klas w algorytmie

3. Struktura testów

Na rys. 3.1 przedstawiono schemat blokowy planowanych testów. Planowane testy podzielono na poszczególne etapy:

1. Wczytanie danych przeznaczonych do zakodowania z pliku tekstowego
2. Uruchomienie kodowania AES
3. Odczytanie zakodowanego ciągu
4. Wczytanie prawidłowego wyniku z pliku tekstowego
5. Porównanie wyników
6. Zapis wyniku porównania
7. Uruchomienie dekodowania AES
8. Odczytanie odkodowanego ciągu
9. Porównanie wyników
10. Zapis wyniku porównania



Rysunek 3.1. Schemat blokowy testów zaimplementowanego algorytmu

Do walidacji uzyskiwanych wyników na wyjściu algorytmu zostanie wykorzystana biblioteka języka Python Crypto.Cipher. Klucz i wiadomość przeznaczona do zakodowania zostaną zapisane w pliku tekstowym, skrypt (listing 4) wczyta dane informacje a następnie obliczy ciąg wyjściowy, który zostanie zapisany do pliku i wykorzystany w testach z rys. 3.1.

Listing 4. *Kod do weryfikacji obliczeń*

```
1  from Crypto.Cipher import AES
2  from Crypto.Util.Padding import pad
3  from base64 import b64encode
4
5  keyFile = open("key.txt", "r")
6  key = bytes(keyFile.read(), 'utf-8')
7
8  dataFile = open("key.txt", "r")
9  data = bytes(dataFile.read(), 'utf-8')
10
11 cipher = AES.new(key[0:-1], AES.MODE_ECB)
```

```
12  cipherText = cipher.encrypt(pad(data[0:-1], AES.block_size))
13  cipherText = b64encode(cipherText).decode('utf-8')
14
15  cipherTextFileECB= open("cipherTextECB.txt", "a")
16  cipherTextFileECB.write(cipherText)
17
18  cipher = AES.new(key[0:-1], AES.MODE_CBC)
19  cipherText = cipher.encrypt(pad(data[0:-1], AES.block_size))
20  cipherText = b64encode(cipherText).decode('utf-8')
21
22  cipherTextFileCBC= open("cipherTextCBC.txt", "a")
23  cipherTextFileCBC.write(cipherText)
```

W celu ustalenia potencjalnych miejsc do przyśpieszenia w algorytmie wykonującym obliczenia sekwencyjnie zostanie wykorzystany program VTune firmy intel. Program umożliwia wskazanie czasu wykonywania poszczególnych fragmentów kodu.

4. Optymalizacja

Po zaimplementowaniu algorytmu w pierwotnej wersji i sprawdzeniu poprawności działania zostaną przeprowadzone próby optymalizacji algorytmu pod względem czasu szyfrowania. W wersji przed modyfikacjami każdy wątek będzie odpowiedzialny za przetworzenie pojedynczego bloku o wielkości $4x4$ bajty, które zachodzi w pętli. Wybrano trzy sposoby optymalizacji programu.

Pierwszy polega na programowym rozwinięciu wspomnianej pętli na pojedyncze instrukcje wykonywane jedna po drugiej za pomocą dyrektywy *#pragma unroll*.

Następnym krokiem będzie rozdzielenie przetwarzania pojedynczego bloku danych $4x4$ na oddzielne wątki, gdyż są to operacje niezależne. Każdy wątek będzie odpowiedzialny za przetwarzanie 4 bajtów zamiast 16. Krok ten zajmuje najwięcej czasu w pojedynczej pętli algorytmu, więc zrównoleglenie go powinno istotnie skrócić czas działania programu.

Trzecim krokiem będzie przeniesienie szyfrowanych danych z pamięci współdzielonej do rejestrów. W pierwotnej wersji algorytmu szyfrowany blok $4x4$ znajduje się w pamięci współdzielonej, co powoduje większe opóźnienie w dostępnie do danych. Wykorzystanie rejestrów powinno przynieść mniejsze opóźnienie i zapewnić brak konfliktów w dostępnie do danych.

Bibliografia

[1] H. GO, “The difference in five modes in the AES encryption algorithm”, https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/?fbclid=IwAR3P7fznaPbcAIz6l3cziYA-oQGfMMnsmVClmHFrwjbJ0Gym 2019.

Wykaz symboli i skrótów

EiTI – Wydział Elektroniki i Technik Informacyjnych
PW – Politechnika Warszawska
WEIRD – ang. *Western, Educated, Industrialized, Rich and Democratic*

Spis rysunków

Spis tabel

Spis załączników