Instytut Informatyki Politechniki Śląskiej

Zespół Mikroinformatyki i Teorii Automatów Cyfrowych

Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/SMiW):	Grupa	Sekcja
2017/2018	**SSI**	**Języki Asemblerowe**	**4**	**7**

Imię:	**Patryk**	**Prowadzący**: OA/JP/KT/GD/BSz/GB	**JP**
Nazwisko:	**Gregorczuk**		

Raport końcowy

Temat projektu:

Szyfrowanie i deszyfrowanie plików algorytmem Vigenère'a.

Data oddania: dd/mm/rrrr	**16.02.2018**

1. Temat projektu i opis założeń.

Tematem projektu jest program umożliwiający szyfrowanie i deszyfrowanie jednego wskazanego pliku tekstowego za pomocą podanego przez użytkownika klucza zgodnie z metodą Vigenère'a z uwzględnieniem polskich znaków diakrytycznych. Funkcja szyfrująca oraz deszyfrująca zostały zaimplementowane w języku C++ oraz Asemblerze i dołączone do programu głównego w postaci bibliotek DLL. Program główny wraz z interfejsem graficznym został zaimplementowany w języku C# z wykorzystaniem WPF (Windows Presentation Foundation). GUI aplikacji umożliwia użytkownikowi wybór pliku wejściowego, folderu do którego zostanie zapisany plik wynikowy, klucza na podstawie którego zostanie zaszyfrowany plik, trybu działania (szyfrowanie/deszyfrowanie), biblioteki (C/ASM), ilości wątków jaką program ma wykorzystać oraz rozmiaru bufora wejściowego.

Program wczytuje fragmentami przekazany plik do bufora wejściowego o wybranym rozmiarze, a następnie dokonuje podziału tego bufora na mniejsze segmenty przekazywane do funkcji bibliotecznych gdzie są przetwarzane przez wątki. Po przetworzeniu segmenty są na powrót scalane do jednego bufora wyjściowego, a następnie zapisywane do pliku wynikowego.

2. Analiza zadania.

Szyfrowanie tą metodą polega na przesunięciu szyfrowanej litery o tyle znaków względem jej pozycji w tablicy szyfrującej o ile przesunięta jest aktualna litera klucza. Domyślnie tablica szyfrująca składa się tylko z dużych liter alfabetu łacińskiego. Aby móc uwzględniać polskie znaki diakrytyczne należało rozszerzyć tablicę o brakujące litery. Dodatkowo dodane zostało uwzględnienie dużych i małych liter, aby po zaszyfrowaniu możliwe było odszyfrowanie tekstu do postaci początkowej.

Wymagana obsługa polskich znaków sprawia, że konieczne jest wykorzystywanie własnej, rozszerzonej tablicy szyfrującej. W takim przypadku wystarczyły by operacje na znakach ASCII. Wówczas algorytm byłby szybszy, ponieważ nie byłoby konieczności iterowania po tablicy szyfrującej w poszukiwaniu indeksów właściwych liter.

Podczas szyfrowania indeks litery, która ma zastąpić szyfrowaną literę obliczany jest według wzoru:

$$indeksWynikowy = indeksLiteryKlucza + indeksSzyfrowanejLitery$$

Jeżeli *indeksWynikowy* jest przekracza rozmiar tablicy szyfrującej pomniejszony o 1 (maksymalny indeks tablicy zaczynając od 0) należy wykonać korektę odejmując od niego rozmiar tablicy szyfrującej.

W przypadku deszyfrowania ma miejsce dokładnie odwrotna operacja:

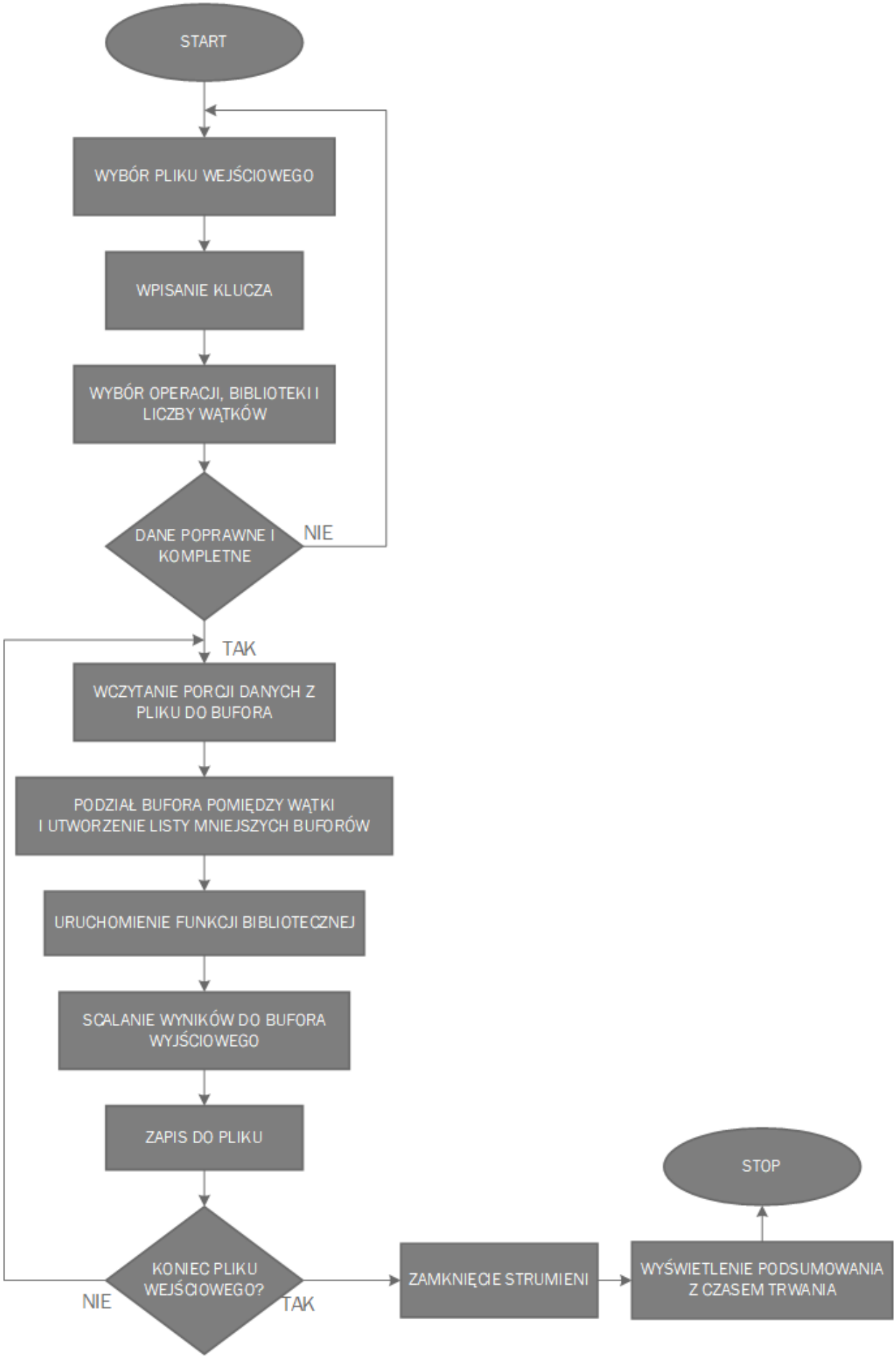
$$indeksWynikowy = indeksDeszyfrowanejLitery - indeksLiteryKlucza$$

Jeżeli *indeksWynikowy* jest mniejszy od 0 należy wykonać korektę dodając do niego rozmiar tablicy szyfrującej.

W obu przypadkach wszelkie indeksy we wzorach dotyczą indeksu odpowiadającej im litery w tablicy szyfrującej.

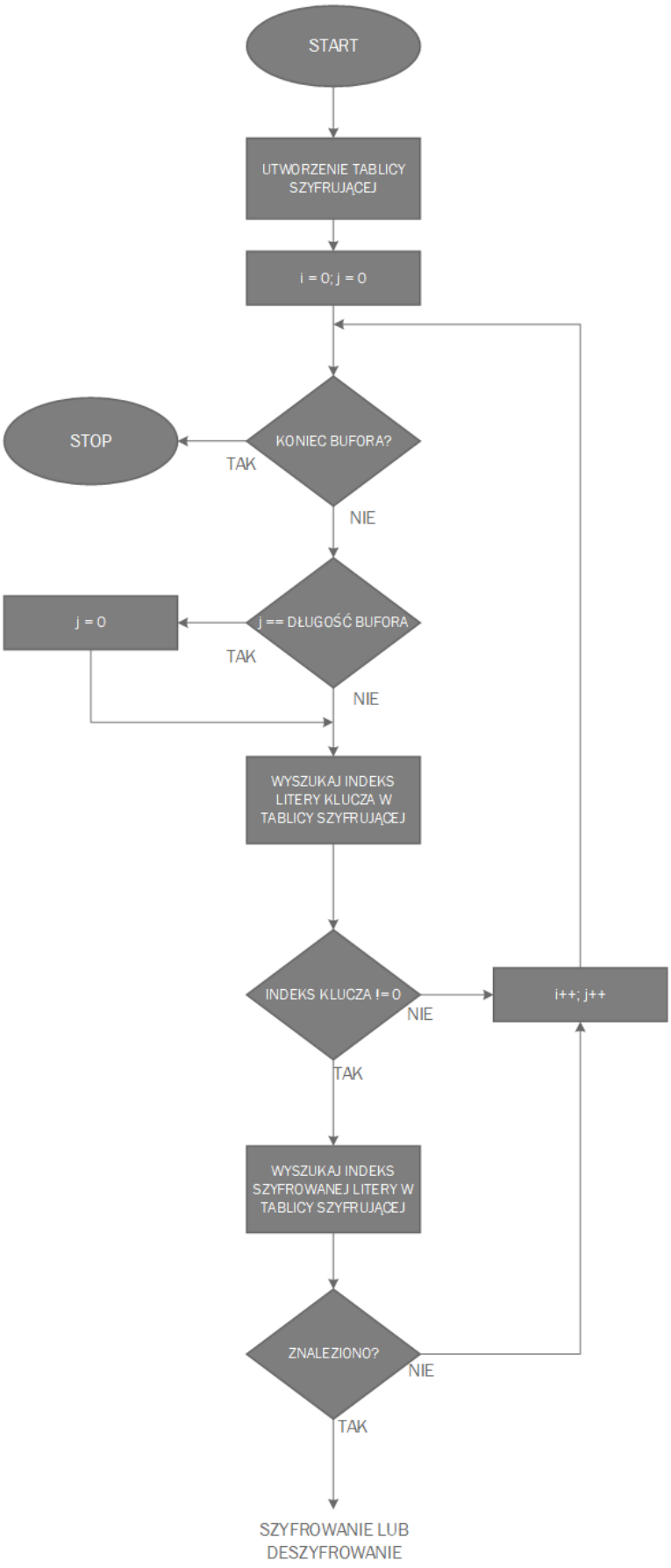
3. Schematy blokowe.

a) Program główny

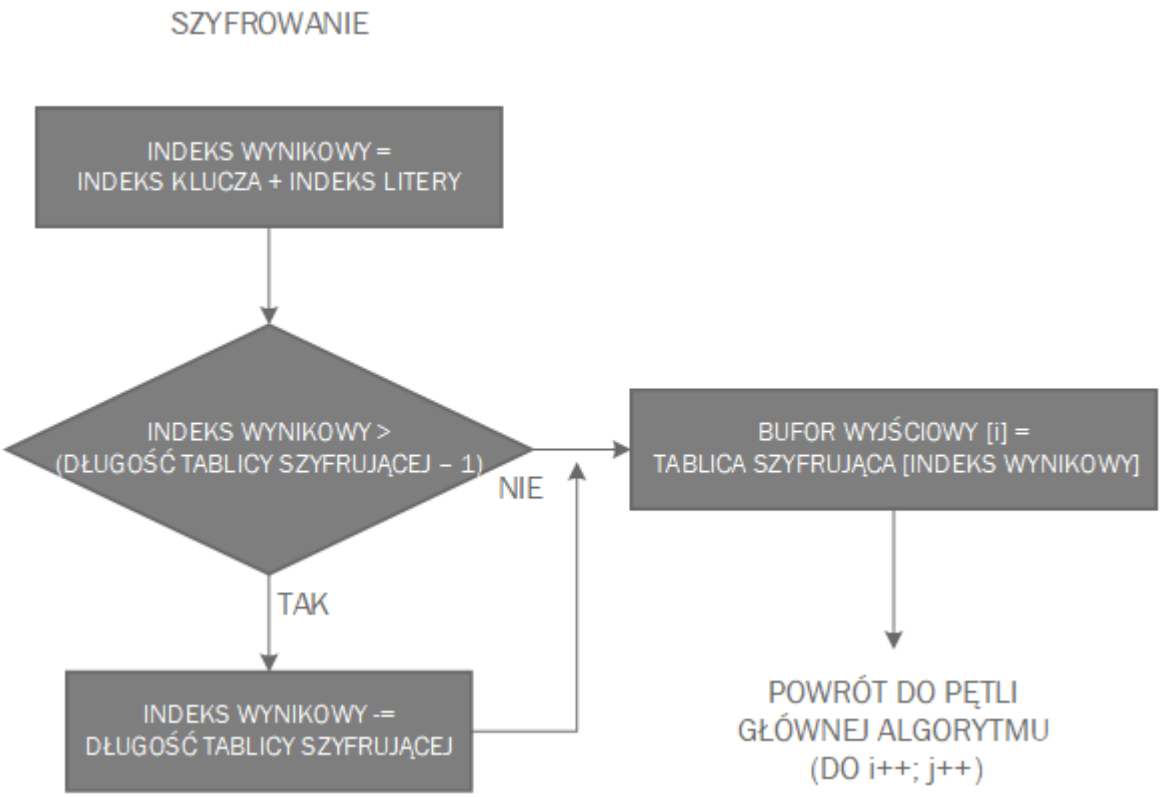


Rys. 1 - Schemat blokowy algorytmu programu.

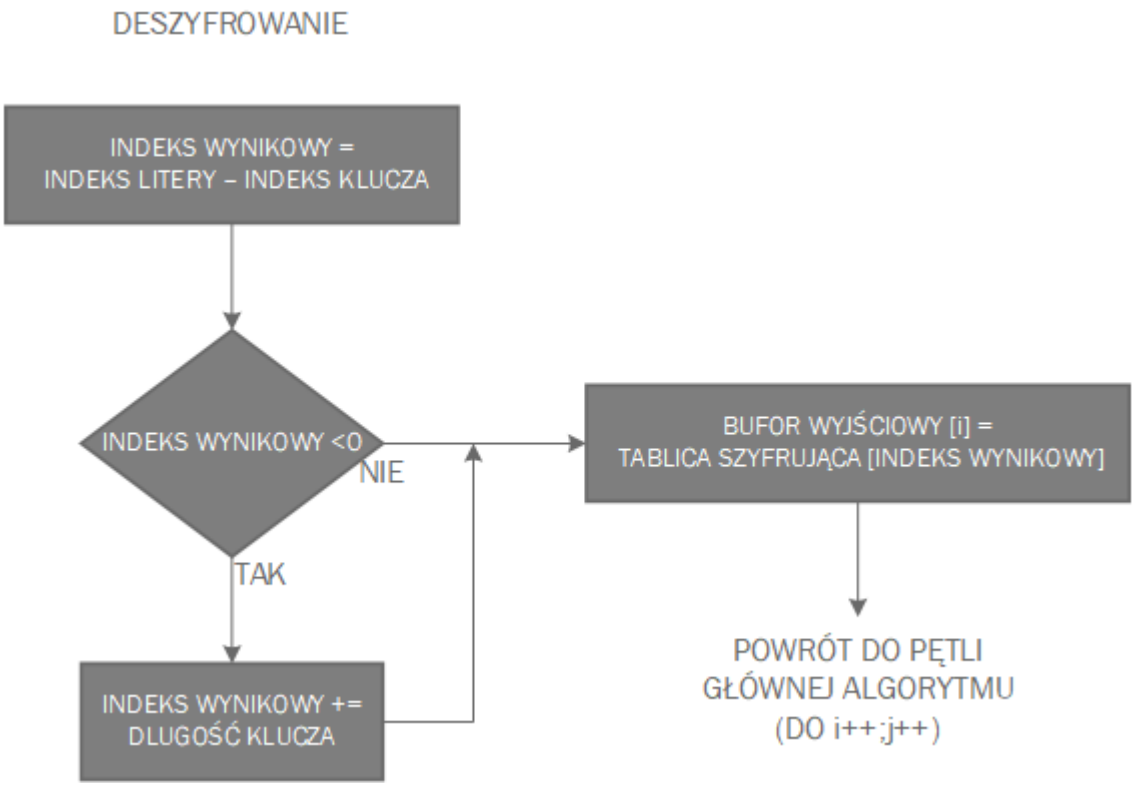
b) Biblioteki



Rys. 2 - Schemat głównej części algorytmu bibliotek.



Rys. 3 - Schemat algorytmu szyfrowania w bibliotekach.



Rys. 4 - Schemat algorytmu deszyfrowania w bibliotekach.

4. Opis programu głównego.

Nazwa i opis klasy:

```
public partial class MainWindow : Window
```
Klasa odpowiedzialna za obsługę okna programu.

Lista i opis zmiennych:

```
private int selectedLibrary = 0;
private int selectedMode = 0;
```
Zmienne przyjmują wartości na podstawie zaznaczonych RadioButtonów w oknie programu. Dla selectedLibrary: biblioteka C - selectedLibrary = 1, biblioteka ASM - selectedLibrary = 2. Dla selectedMode: szyfrowanie - selectedMode = 5, deszyfrowanie - selectedMode = 10.

```
private int threadsCount = 1;
```
Zmienna służy do przechowywania ilości wątków wybranej za pomocą suwaka.

```
private string inputFileName;
```
Zmienna przechowuje ścieżkę do wybranego pliku wejściowego.

```
private string outputFilePath;
private string outputFileName;
```
Zmienne przechowują folder wyjściowy oraz nazwę pliku wynikowego.

```
private VigenereConverter vigenereConverter;
```
Zmienna przechowuje instancję klasy przeprowadzającej operacje na danych. Instancja tworzona jest po naciśnięciu przycisku rozpoczynającego szyfrowanie/deszyfrowanie.

```
private Stopwatch mainStopwatch;
private Stopwatch libraryStopwatch;
```
Zmienne przechowują instancje klas typu Stopwatch służące do przeprowadzania pomiarów czasu wykonania całego procesu szyfrowania/deszyfrowania oraz samego czasu wykonywania funkcji bibliotecznych.

Lista i opis metod:

```
private void calculateButton_Click(object sender, RoutedEventArgs e)
```
Metoda obsługująca naciśnięcie przycisku rozpoczynającego przetwarzanie pliku. Wewnątrz niej sprawdzana jest poprawność danych podanych przez użytkownika i wyświetlane ewentualne komunikaty o błędnych bądź niekompletnych danych, tworzony jest również obiekt klasy VigenereConverter do którego przekazywana jest dalsza obsługa pliku oraz uruchamiany jest pomiar czasu wykonywania programu. Po zakończeniu operacji na pliku wyświetlany jest komunikat z podsumowaniem w postaci czasu działania.

```
private void inputFileButton_Click(object sender, RoutedEventArgs e)
```
Metoda obsługująca naciśnięcie przycisku odpowiedzialnego za wybór pliku wejściowego. Po naciśnięciu przycisku wyświetlany jest systemowy FileDialog, w którym wybierany jest plik wejściowy o rozszerzeniu .txt.

```
private void outputFileButton_Click(object sender, RoutedEventArgs e)
```
Metoda obsługująca naciśnięcie przycisku odpowiedzialnego za wybór folderu wyjściowego. Po naciśnięciu przycisku wyświetlany jest systemowy FolderBrowserDialog zaimportowany z bibliotek Windows Forms.

```
private void CLibraryRadioButton_Checked(object sender, RoutedEventArgs e)
private void AsmLibraryRadioButton_Checked(object sender, RoutedEventArgs e)
private void codingModeRadioButton_Checked(object sender, RoutedEventArgs e)
private void decodingModeRadioButton_Checked(object sender, RoutedEventArgs e)
```
Zbiór metod odpowiedzialnych za ustawianie wartości odpowiednich zmiennych po wybraniu przez użytkownika danej opcji.

```
private void threadsCountSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
```
Metoda obsługująca ustawienie wartości zmiennej przechowującej ilość wykorzystywanych wątków po wykryciu przesunięcie suwaka.

```
private bool checkCipherKeyCorectness(string cipherKey)
```
Metoda sprawdzająca poprawność podanego przez użytkownika klucza szyfrującego. Dopuszcza ona wyłącznie duże i małe litery alfabetu łacińskiego z dołączeniem polskich znaków diakrytycznych.

Nazwa i opis klasy:

```
public class VigenereConverter
```
Klasa odpowiedzialna za wszelkie operacje związane z właściwym przeznaczeniem programu - wczytywanie pliku do bufora wejściowego, podział danych na mniejsze bufory do obsługi przez wątki, przetworzenie danych, przygotowanie do zapisu poprzez scalanie do bufora wyjściowego i wreszcie zapis do pliku.

Lista i opis zmiennych:

```
private List<StringBuilder> buffers;
```
Zmienna przechowująca listę buforów. Trafiają do niej podzielone na segmenty dane z bufora wejściowego.

```
private string fileName;
```
Zmienna przechowuje nazwę pliku wejściowego, który jest otwierany w konstruktorze klasy.

```
private string outputFilePath;
```
Zmienna przechowuje ścieżkę do folderu wyjściowego. Domyślnie jest ona taka sama dla pliku wyjściowego jak dla pliku wejściowego.

```
private string outputFileName;
```
Zmienna przechowuje nazwę pliku wyjściowego, który jest otwierany w konstruktorze klasy.

```
private string resultFile;
```
Zmienna przechowuje nazwę pliku wyjściowego razem ze ścieżką dostępu.

```
private int threadsCount;
```
Zmienna przechowuje liczbę wykorzystywanych wątków.

```
int segmentSize;
```
Zmienna przechowuje rozmiar pojedynczych segmentów przechowywanych w liście buforów.

```
int cipherKeyLength;
```
Zmienna przechowuje długość klucza szyfrującego.

```
private int programMode;
```
Zmienna przechowuje tryb działania programu. Jest ona sumą zmiennych selectedLibrary oraz selectedMode z klasy MainWindow.

```
private int bufferSize;
```
Zmienna przechowuje przekazany przez użytkownika rozmiar bufora.

```
private char[] cipherKeyCharArray;
```
Zmienna przechowuje klucz szyfrujący w postaci tablicy typu char.

```
private StringBuilder cipherKey;
```
Zmienna przechowuje referencję do obiektu tworzącego string z kluczem szyfrującym.

```
private Stopwatch libraryStopwatch;
```
Zmienna przechowuje referencję do obiektu służącego do pomiaru czasu działania funkcji bibliotecznych.

```
MainWindow mainWindow;
```
Zmienna przechowuje referencję do głównego okna programu w celu umożliwienia odświeżania paska postępu.

```
Encoding encoding;
Encoding encodingOutput;
```
Zmienne przechowują sposób kodowania plików wejściowych oraz wyjściowych.

```
int correction;
```
Zmienna przechowuje informację o wielkości korekcji bajtów w zależności od kodowania.

```
StreamReader input;
StreamWriter output;
```
Zmienne przechowują strumienie wejściowe i wyjściowe do obsługi plików.

Lista i opis metod:

```
public VigenereConverter(string fileName, string outputFilePath, string outputFileName,
int threadsCount, int programMode, int bufferSize, char[] cipherKey, Stopwatch
libraryStopwatch, MainWindow mainWindow)
```
Konstruktor klasy VigenereConverter inicjalizuje pola klasy przy pomocy przekazywanych argumentów.

```
private delegate void Updater(int UI);
```
Deklaracja metody delegata. Odpowiada za wywoływanie aktualizacji głównego widoku programu polegającą na przesunięciu paska postępu.

```
public void UpdateData(int percentProgressValue)
```
Metoda aktualizująca stan paska postępu.

```
public void ProcessData()
```
Główna metoda klasy wywoływana z poziomu widoku. Odpowiada za wywoływanie pozostałych metod prywatnych zawartych wewnątrz klasy oraz funkcji bibliotecznych.

```
private int ReadFile(ref char[] data, ref long offset, ref bool isFinished)
```
Metoda wczytuje dane z pliku do bufora data. Parametr offset informuje od którego miejsca pliku metoda ma wczytywać dane. Zmienna typu bool ustawiana jest na true po przeczytaniu całego pliku, oznacza to że główna pętla ma się zakończyć po wykonaniu bieżącej iteracji.

```
private List<StringBuilder> SplitData(char[] data, ref int
segmentSize, ref int divisionOutlet)
```
Metoda odpowiedzialna za podział bufora wejściowego na mniejsze bufory. Zwraca listę segmentów powstałych w wyniku podziału. Zmienna segmentSize jest modyfikowana wewnątrz i oznacza długość segmentów. Przydział odbywa się tak, aby pierwszy znak w każdym segmencie był kodowany przez pierwszą literę klucza. Zmienna divisionOutlet wskazuje ile znaków z bufora wejściowego pozostała nieprzydzielona - o tyle należy cofnąć offset wczytujący dane z pliku.

```
private StringBuilder mergeBuffers(List<StringBuilder> buffers, int length,
int segmentLength)
```
Metoda odpowiedzialna za scalanie listy segmentów w jeden bufor wyjściowy. Argument length oznacza długość tworzonego bufora wyjściowego. Argument segmentLength jest wykorzystywany do odpowiedniego przesunięcia pozycji w buforze wyjściowym, do której należy skopiować dane z kolejnych segmentów. Zwraca bufor wyjściowy zaszyty wewnątrz obiektu klasy StringBuilder.

```
private void WriteFile(char[] data)
```
Metoda zapisuje do pliku bufor wyjściowy przekazany w postaci tablicy typu char.

5. Biblioteka DLL w języku C++.

W bibliotece tej znajdują się dwie procedury:

```
void _stdcall CodingC(wchar_t* wcharArray, int bufferSize, wchar_t* key, int keyLength)
```
Procedura szyfruje przekazany segment w postaci argumentu wskaźnika na wcharArray. Parametr bufferSize mówi jak duży jest przetwarzany segment. Analogicznie w przypadku dwóch pozostałych parametrów: key to wskaźnik na klucz szyfrujący, a keyLength oznacza długość klucza. Znaki zostały rozszerzone do dwóch bajtów, aby ujednolicić ich obsługę, stąd wykorzystanie typu wchar_t.

```
void _stdcall DecodingC(wchar_t* wcharArray, int bufferSize, wchar_t* key, int keyLength)
```
Analogicznie do procedury szyfrującej, procedura deszyfrująca przyjmuje dokładnie takie same parametry. Natomiast jej właściwe działanie jest dokładnie odwrotne względem procedury szyfrującej.

Dodatkowo posiada ona dwie globalne, nieudostępniane poza obręb biblioteki zmienne:

```
wchar_t baseCryptogram[] =
{
	L'A', L'Ą', L'B', L'C', L'Ć', L'D', L'E', L'Ę', L'F', L'G',
	L'H', L'I', L'J', L'K', L'L', L'Ł', L'M', L'N', L'Ń', L'O',
	L'Ó', L'P', L'Q', L'R', L'S', L'Ś', L'T', L'U', L'V', L'W',
	L'X', L'Y', L'Z', L'Ź', L'Ż', L'a', L'ą', L'b', L'c', L'ć',
	L'd', L'e', L'ę', L'f', L'g', L'h', L'i', L'j', L'k', L'l',
	L'ł', L'm', L'n', L'ń', L'o', L'ó', L'p', L'q', L'r', L's',
	L'ś', L't', L'u', L'v', L'w', L'x', L'y', L'z', L'ź', L'ż'
};

int cryptogramLength = 70;
```

Pierwsza z nich reprezentuje rozszerzoną do dwóch bajtów na każdy znak tablicę szyfrującą uwzględniającą małe i duże litery oraz polskie znaki. Druga zaś określa długość tejże tablicy.

6. Biblioteka DLL w Asemblerze.

W bibliotece tej również znajdują się dwie procedury:

```
CodingAsm PROC export data:QWORD, dataSize:DWORD, key:QWORD, keyLength:DWORD
```
Jest to procedura szyfrująca, która jako argumenty przyjmuje kolejno: segment w postaci tablicy, długość segmentu, klucz w postaci tablicy oraz długość klucza.

```
DecodingAsm PROC export dataSize:DWORD, data:QWORD, key:QWORD, keyLength:DWORD
```
Procedura deszyfrująca, która przyjmuje takie same argumenty jak procedura szyfrująca.

Wewnątrz procedury zostały zdefiniowane również zmienne:

```
cipherTable:
dw 0041h, 0104h, 0042h, 0043h, 0106h, 0044h, 0045h, 0118h, 0046h, 0047h
dw 0048h, 0049h, 004Ah, 004Bh, 004Ch, 0141h, 004Dh, 004Eh, 0143h, 004Fh
dw 00D3h, 0050h, 0051h, 0052h, 0053h, 015Ah, 0054h, 0055h, 0056h, 0057h
dw 0058h, 0059h, 005Ah, 0179h, 017Bh, 0061h, 0105h, 0062h, 0063h, 0107h
dw 0064h, 0065h, 0119h, 0066h, 0067h, 0068h, 0069h, 006Ah, 006Bh, 006Ch
dw 0142h, 006Dh, 006Eh, 0144h, 006Fh, 00F3h, 0070h, 0071h, 0072h, 0073h
dw 015Bh, 0074h, 0075h, 0076h, 0077h, 0078h, 0079h, 007Ah, 017Ah, 017Ch

cipherTableLength db 70
```

Podobnie jak w przypadku biblioteki w C++ cipherTable to tablica szyfrująca rozszerzona na dwa bajty z uwzględnieniem polskich znaków, dużych i małych liter oraz cipherTableLength określająca długość tablicy szyfrującej.

```
Opis wykorzystania rejestrów w procedurach(nazwy zmiennych analogiczne do tych z C++):
RDI - rejestr przechowujący indeks tablicy wejściowej 'i'
RBX - rejestr przechowuje adres pierwszego elementu tablicy wejściowej
RSI - rejestr przechowuje adres pierwszego elementu tablicy cipherTable
R10/MM0 - rejestr przechowuje indeks litery z tablicy cipherKey, którą po wykonaniu
szyfrowania/deszyfrowania należy wpisać w miejsce szyfrowanej/deszyfrowanej litery
R13 - rejestr przechowuje zmienna iterującą po kluczu 'j'
R14/MM1 - rejestr przechowuje indeks odpowiadającej litery klucza z tablicy cipherTable
R15/MM2 - rejestr przechowuje indeks odpowiadającej szyfrowanej/deszyfrowanej litery z
tablicy cipherTable
```

Na rejestrach MM0, MM1, MM2 - wykonywane są operacje dodawania oraz odejmowania indeksów. Wcześniej indeksy te zostały znalezione i zapisane w rejestrach R14 oraz R15 po czym trafiły do rejestrów MM, gdzie wykonywane są wyżej wspomniane operacje.

7. Opis struktury danych wejściowych.

Dane wejściowe stanowi pojedynczy plik tekstowy o rozszerzeniu .txt. Plik poddawany szyfrowaniu powinien być kodowany w Windows-1250 lub opcjonalnie UTF-8, jednak jest to mniej zalecana opcja, ponieważ UTF-8 charakteryzuje się zmienną długością znaków w wyniku czego niektóre znaki kodowane są jako jeden bajt, a inne (np. polskie) jako dwa. Zaszyfrowany plik o kodowaniu UTF-8 w rezultacie będzie posiadał polskie znaki rozbite na dwie osobne litery, których nie ma w tablicy szyfrującej przez co nie zostaną one zaszyfrowane. Do pliku wynikowego przepisane zostaną te dwie litery. Jednak po wykonaniu deszyfrowania znaki te znów będą odpowiednią literą.

Deszyfrowane powinny być pliki będące efektem działania funkcji szyfrujących tego samego programu ze względu na rozszerzenie o polskie znaki diakrytyczne oraz uwzględnianą wielkość liter, a także kodowanie zaszyfrowanych plików wyjściowych, które zostało ustawione na Unicode.

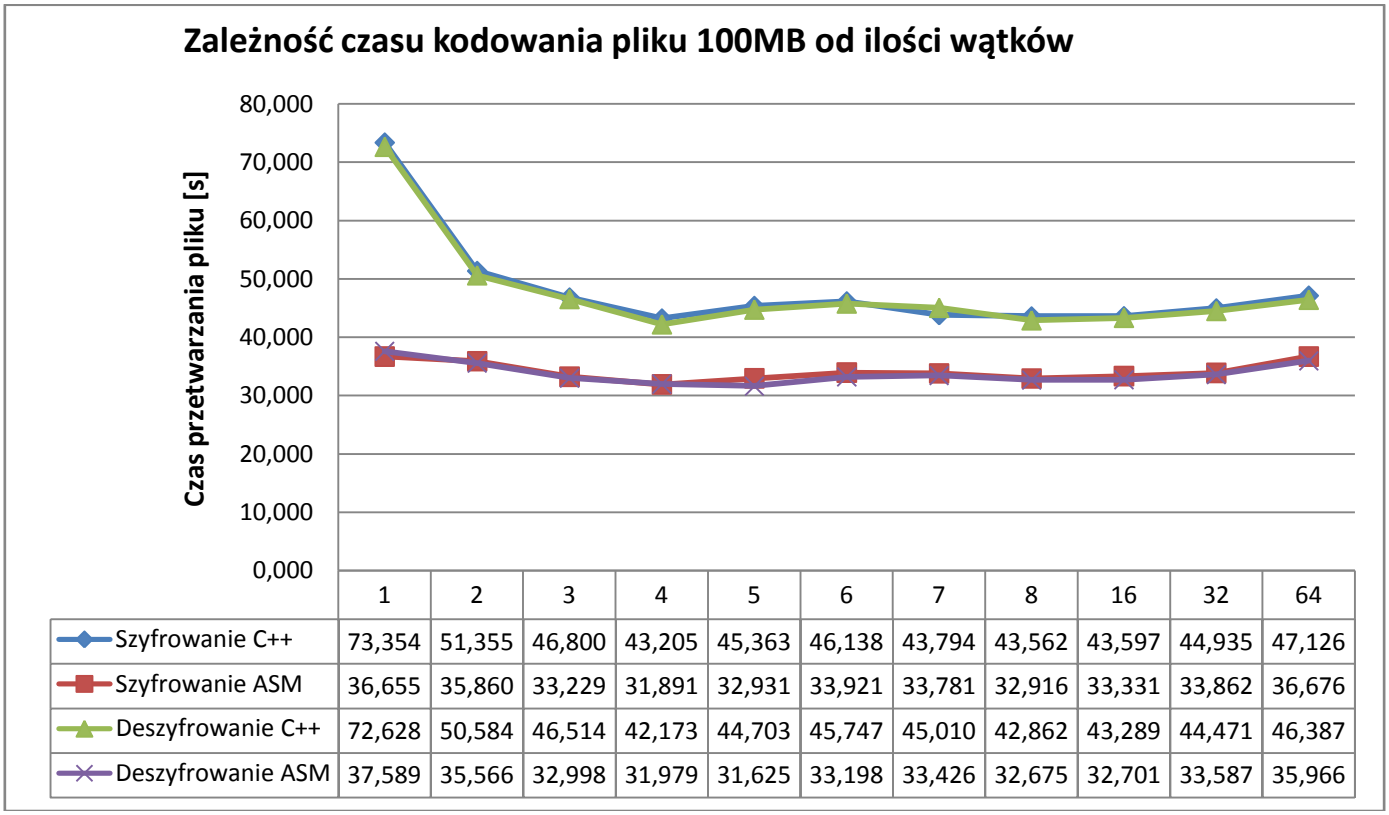
8. Uruchamianie i testowanie.

Program został przetestowany w wygenerowanej wersji Release. W głównym oknie programu przetestowana została każda możliwość podania niekompletnych lub błędnych danych. Przez błędne dane należy rozumieć pusty, za długi lub zawierający niedozwolone znaki klucz szyfrujący. W takiej sytuacji każda próba uruchomienia obliczeń powoduje wyświetlenie stosownego komunikatu. Podobnie w przypadku nie wybrania pliku, trybu bądź biblioteki.

Aplikację testowano zarówno na małych plikach zawierających kilka liter, jak i większych rzędu +500MB dla różnych ilości wątków. Przetestowano obsługę polskich znaków. Poprawność plików po zaszyfrowaniu stwierdzano ręcznie na podstawie analizy kluczowych dla algorytmu miejsc, jak na przykład koniec segmentu, początek nowego segmentu, przejście pomiędzy ostatnim, a pierwszym indeksem tablicy szyfrującej i na odwrót. Ostatecznie całość została zweryfikowana poprzez deszyfrowanie, a następnie porównanie oryginalnego pliku z odszyfrowanym za pomocą narzędzia porównującego dostępnego w programie Word.

9. Wyniki pomiarów.

Pomiary dla szyfrowania i deszyfrowania wykonano korzystając ze stałego klucza szyfrującego "AĄBCĆD", bufora o rozmiarze 4kB oraz pliku tekstowego o rozmiarze ~100MB.

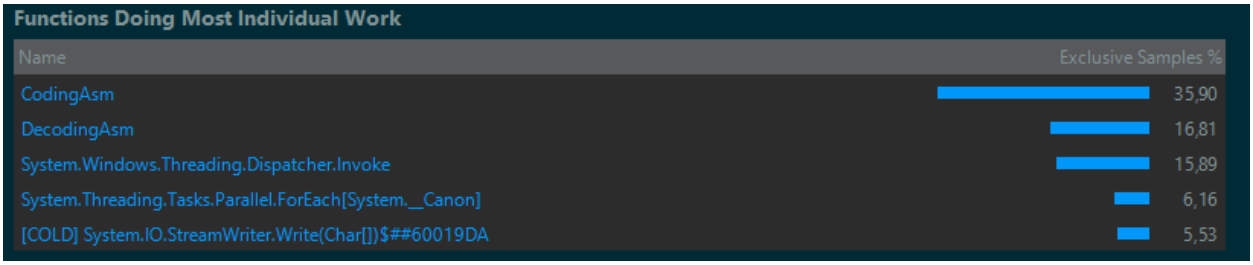


	1	2	3	4	5	6	7	8	16	32	64
Szyfrowanie C++	73,354	51,355	46,800	43,205	45,363	46,138	43,794	43,562	43,597	44,935	47,126
Szyfrowanie ASM	36,655	35,860	33,229	31,891	32,931	33,921	33,781	32,916	33,331	33,862	36,676
Deszyfrowanie C++	72,628	50,584	46,514	42,173	44,703	45,747	45,010	42,862	43,289	44,471	46,387
Deszyfrowanie ASM	37,589	35,566	32,998	31,979	31,625	33,198	33,426	32,675	32,701	33,587	35,966

Pomiary wykonywano na procesorze dwurdzeniowym, czterowątkowym - Intel Core i7-4510U. Zgodnie z oczekiwaniami dla procesora czterowątkowego, najlepsze rezultaty otrzymano przy użyciu czterech wątków. Czasy wykonywania szyfrowania i deszyfrowania w tych samych bibliotekach niemal się pokrywają, co stanowi potwierdzenie symetryczności funkcji szyfrującej i deszyfrującej. Znacznie lepsze czasy udało się uzyskać działając na bibliotece napisanej w Asemblerze, co jest dość nieoczekiwanym wynikiem. Najprawdopodobniej stan ten spowodowany jest nieoptymalnym kodem w bibliotece języka wysokiego poziomu oraz wyłączeniem wszelkich optymalizacji biblioteki na etapie kompilacji.

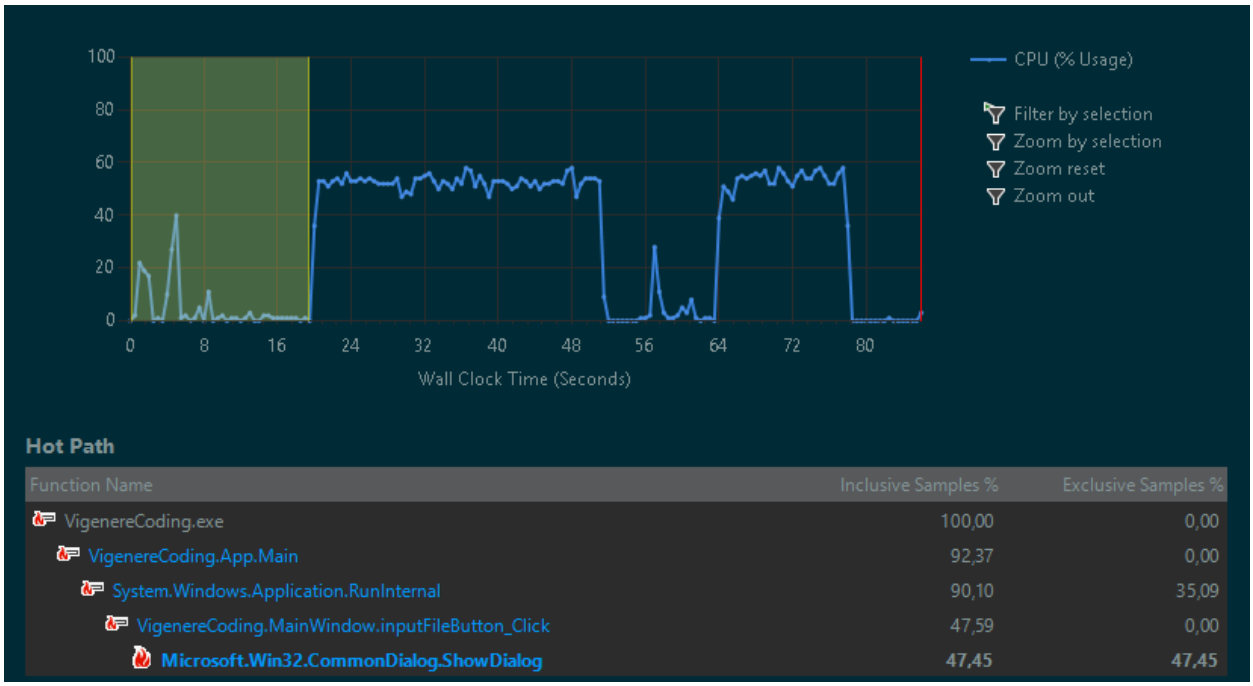
10. Analiza działania programu z wykorzystaniem modułu profilera VS2015.

Podczas analizy działania programu zaszyfrowano przykładowy plik, a następnie na powrót odszyfrowano. Okazuje się, że bardzo obciążające dla procesora na przestrzeni całego procesu testowego jest wywoływanie metody Dispatchera - Invoke. Metoda ta w programie odpowiedzialna jest za umożliwienie odświeżania paska postępu w widoku GUI podczas pracy z wątkami.

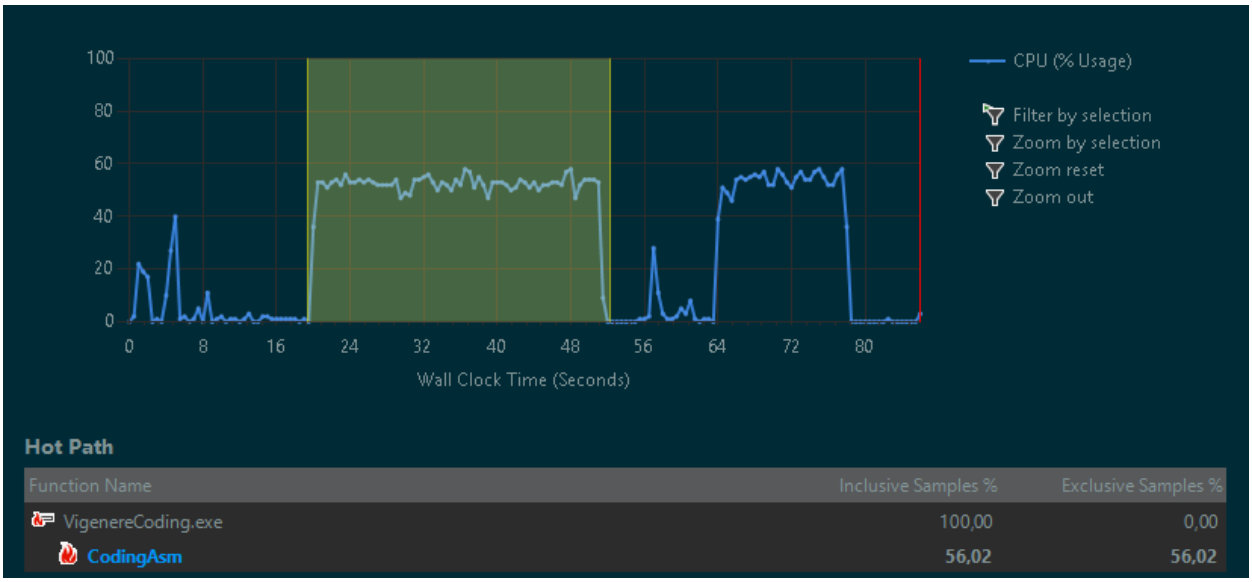


Podczas procesu testowego procedura szyfrująca oraz deszyfrująca wykonały największą pracę. Za nimi znalazł się wspomniany już Dispatcher, a dopiero później metoda ForEach wywołująca wątki czy metoda Write zapisująca dane do pliku.

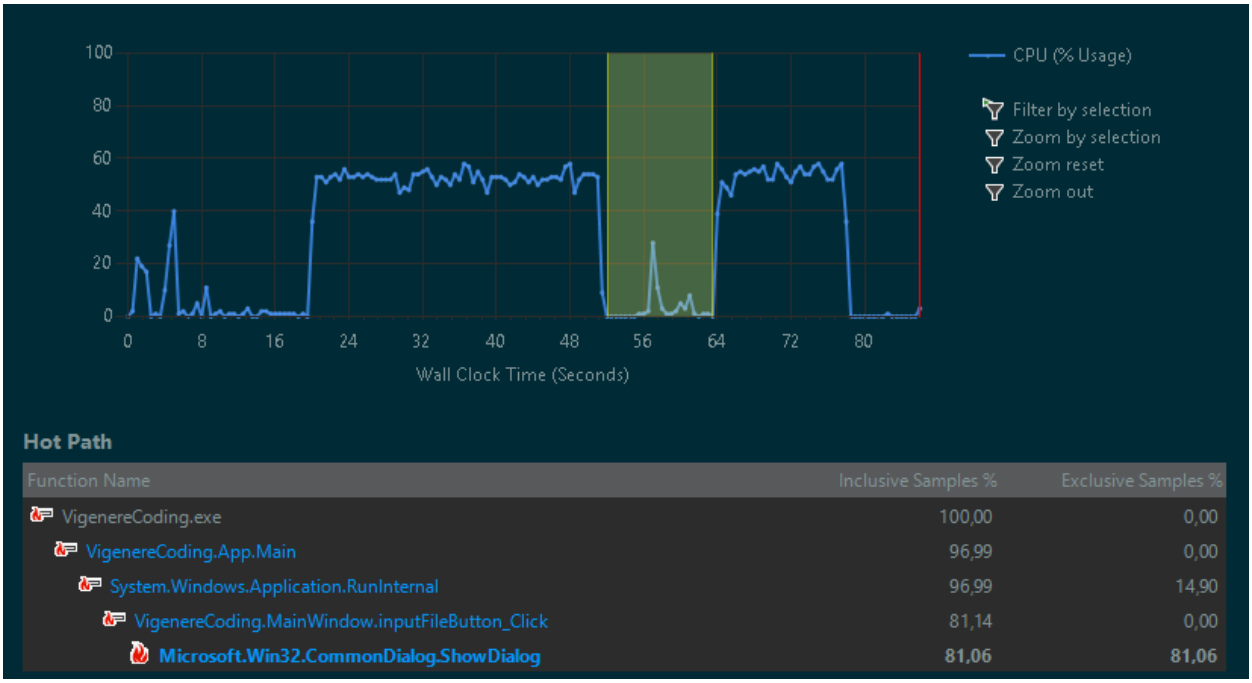
Analizie poddano również poszczególne etapy pracy:



Na początku po załadowaniu aplikacji należy wybrać plik wejściowy. Po kliknięciu na odpowiedni przycisk wyświetlane jest plikowe okno dialogowe. Na tym etapie działania programu wyświetlenie tego okna jest najbardziej obciążające dla procesora.



Po rozpoczęciu szyfrowania najbardziej obciążająca jest sama procedura szyfrująca, co jest pożądanym efektem.



Hot Path

Function Name	Inclusive Samples %	Exclusive Samples %
VigenereCoding.exe	100,00	0,00
VigenereCoding.App.Main	96,99	0,00
System.Windows.Application.RunInternal	96,99	14,90
VigenereCoding.MainWindow.inputFileButton_Click	81,14	0,00
Microsoft.Win32.CommonDialog.ShowDialog	**81,06**	**81,06**

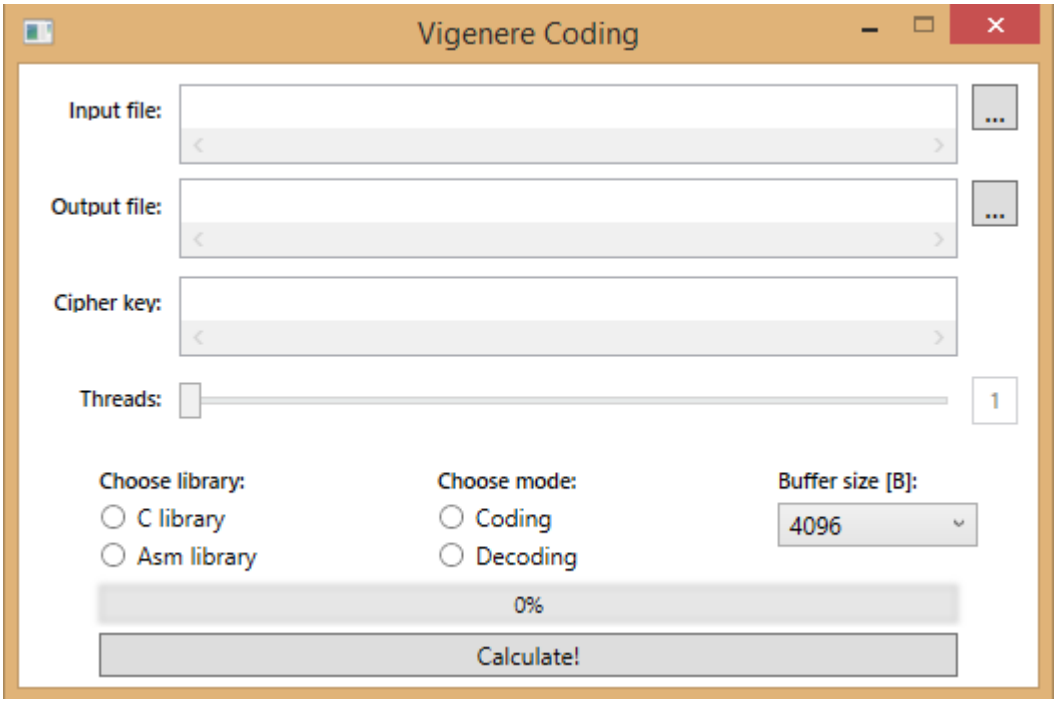
Po zakończeniu szyfrowania musiała nastąpić zmiana pliku wejściowego celem odszyfrowania. W efekcie po raz kolejny wyświetlone zostało mocno obciążające procesor plikowe okno dialogowe.

Hot Path

Function Name	Inclusive Samples %	Exclusive Samples %
VigenereCoding.exe	100,00	0,00
DecodingAsm	**55,90**	**55,90**

Ostatni etap pracy podczas testów to deszyfrowanie. Zgodnie z oczekiwaniami było one procedurą najbardziej obciążającą procesor. Można zauważyć również ciekawe zachowanie aplikacji czy raczej procesora, który był w stanie samemu przyspieszyć proces deszyfrowania. Przypuszczam, że ma to związek z odwołaniami do pamięci cache procesora, dzięki czemu operacje i odwołania do pamięci przebiegają znacznie szybciej. W normalnych warunkach czas trwania tego procesu byłby porównywalny z czasem szyfrowania.

11. Instrukcja obsługi programu.



Po uruchomieniu programu należy kliknąć na przycisk [...] w wierszu Input file. Wyświetlone zostanie systemowe okno dialogowe, w którym należy wskazać plik o rozszerzeniu .txt, który użytkownik chce zaszyfrować. Opcjonalnie w drugim wierszu analogicznie ma możliwość wyboru folderu docelowego. Wiersz oznaczony jako Cipher key, to miejsce na podanie klucza szyfrującego wskazany plik. Nie może on być pusty, nie może być dłuższy niż 16 znaków oraz może zawierać wyłącznie duże i małe litery łacińskie oraz polskie. Poniżej znajduje się suwak za pomocą którego można wybrać ilość wątków z jaką ma pracować program. W dolnej części nad paskiem postępu widnieją dwie sekcje przycisków typu radio - służą one do wyboru biblioteki z jakiej użytkownik chce skorzystać oraz wyboru trybu działania - szyfrowanie lub deszyfrowanie. Dodatkowo użytkownik ma możliwość zmiany rozmiaru bufora wejściowego. Operacje na pliku zaczynają się po wciśnięciu guzika Calculate! widniejącego w dolnej części okna aplikacji. Jeżeli wszystkie pola zostały wypełnione poprawnie rozpocznie się przetwarzanie pliku, co zostanie zasygnalizowane przesuwającym się paskiem postępu tuż nad przyciskiem rozpoczynającym proces. W innym przypadku zostanie wyświetlony komunikat z powodem dla którego plik nie jest jeszcze przetwarzany. Gdy proces się zakończy wyświetlony zostanie komunikat prezentujący czas wykonania programu, po czym użytkownik może szyfrować/deszyfrować kolejne pliki lub wyłączyć aplikację.

12. Wnioski.

Realizacja tego projektu umożliwiła dokładniejsze zapoznanie się z wykorzystaniem instrukcji dla rejestrów wektorowych MMX oraz dała okazję do napisania własnych bibliotek DLL. Na podstawie porównania czasów wykonania funkcji poszczególnych bibliotek można stwierdzić, że dobrze napisana funkcja w Asemblerze może być znacząco szybsza od tych napisanych w języku wysokiego poziomu. Najbardziej problematyczna w projekcie była idea podziału bufora wejściowego na mniejsze obsługiwane przez wątki, a tak że sama konfiguracja projektów i czasami pojawiające się z niewiadomej przyczyny błędy kompilacji typu Unresolved External Symbol wobec kodu, który nie został zmieniony, a jeszcze przed chwilą się kompilował. Dotyczy to głównie biblioteki Asemblerowej.

13. Bibliografia.

- http://softpixel.com/~cwright/programming/simd/mmx.php
- https://pl.wikipedia.org/wiki/Szyfr_Vigen%C3%A8re%E2%80%99a
- https://msdn.microsoft.com/en-us