Politechnika Białostocka

Sprawozdanie z pracowni specjalistycznej

Bezpieczeństwo Sieci Komputerowych

Ćwiczenie numer: 5-6

Temat: *GENERATORY LICZB PSEUDOLOSOWYCH. SZYFRY STRUMIENIOWE*

Wykonujący ćwiczenie: **Wioletta Rushnitskaya, Marcin Paulouski, Igor Balyka**

Studia dzienne

Kierunek: Informatyka

Semestr: VI

Grupa zajęciowa: Grupa PS 9

Prowadzący ćwiczenie: mgr inż. Katarzyna Borowska

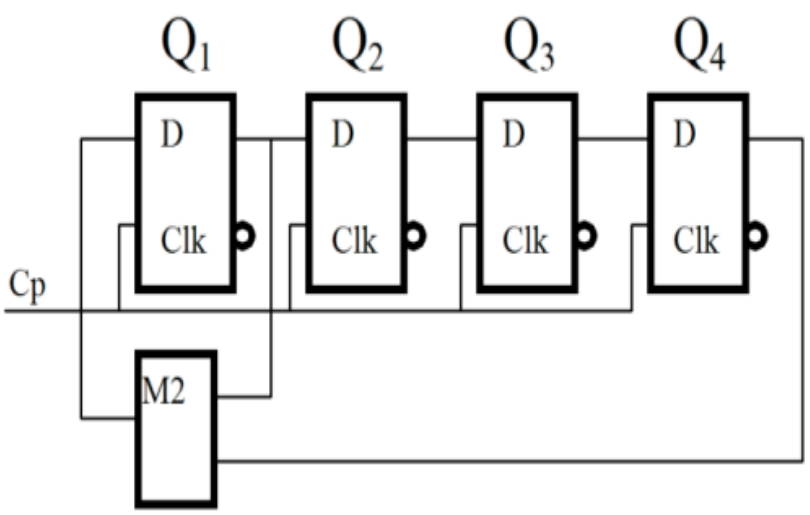
Data wykonania ćwiczenia:
7.04.2022

Treść zadania

Celem zajęć jest kontynuacja zapoznawania się z podstawowymi metodami utajniania informacji. Zaimplementuj i zaprezentuj działanie poniższych algorytmów. Program powinien obsługiwać zarówno opcję szyfrowania, jak i deszyfrowania

Krótkie wprowadzenie teoretyczne

LFSR - rejestr przesuwający z liniowym sprzężeniem zwrotnym



Sekwencja binarna generowana przez LFSR jest okresowa

Okres zależy od stanu początkowego rejestru i struktury sprzężenia zwrotnego

Jest to generator sekwencji (strumienia), w którym blok kombinacyjny to sieć bramek XOR

Kombinacyjny układ sprzężenia zwrotnego jest reprezentowany przez funkcję wielomianową:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + a_1 x^1 + a_0$$

<u>Szyfry strumieniowe:</u>
Stanowią klasę szyfrów z kluczem symetrycznym.

Opierają się na generowaniu strumienia bitów klucza, którego następnie używają do szyfrowania danych wejściowych bit po bicie.

Każdy znak tekstu jawnego szyfrowany jest osobno, używając określonej transformacji szyfrującej, która może być zależna od czasu oraz różna dla każdego ze znaków tekstu jawnego

Do tworzenia klucza wykorzystywane są generatory liczb pseudolosowych (np. LFSR)

Szyfry takie są odpowiednie do zastosowań, w których porcje danych muszą być od razu przetwarzane

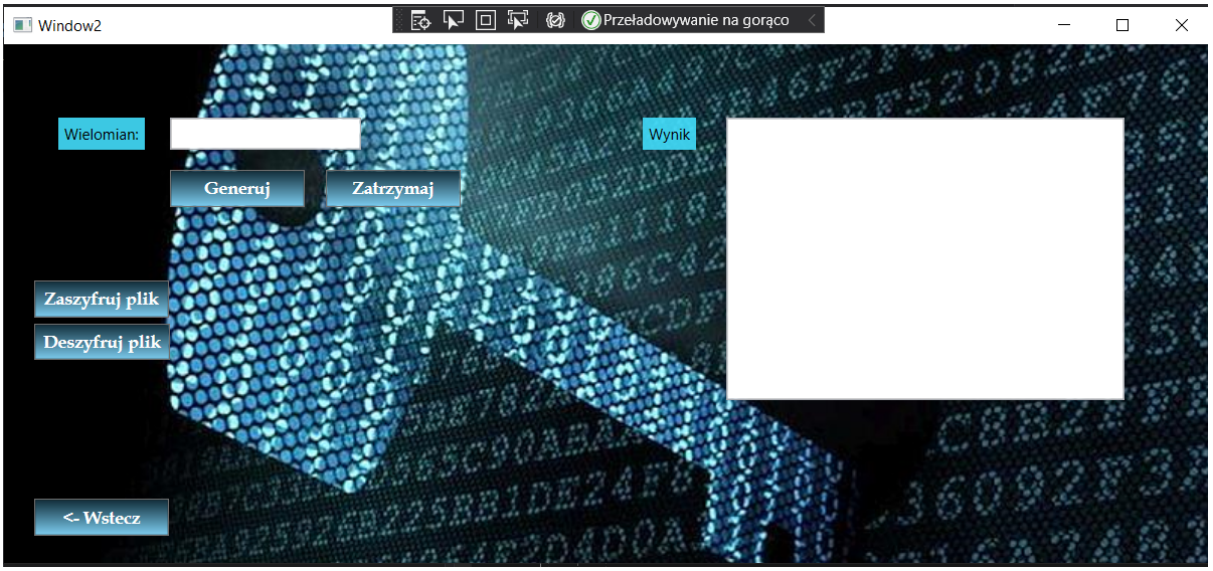
Podział pracy

Link do repozytorium na githubie: https://github.com/VioRush/ModulyKryptograficzne (nowe commity)

Realizacja zadania

Wykorzystane oprogramowanie: Visual Studio Community 2019
Technologia: WPF (.NET Framework)
Język: C#

Nowe dodatkowe okno interfejsu użytkownika:

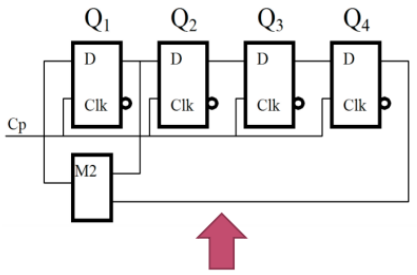


GENERATOR LICZB PSEUDOLOSOWYCH

Zaimplementuj generator liczb pseudolosowych bazujący na metodzie LFSR. Wielomian powinien być podawany przez użytkownika jako parametr. Program powinien generować bity „w nieskończoność” aż do momentu zatrzymania algorytmu przez użytkownika.

Przykład działania algorytmu:

LFSR – przykład



Dane są cztery bloki $Q_1 - Q_4$, gdyż największa potęga wielomianu wynosi 4. Blok $M2$ reprezentuje sprzężenie zwrotne.

$$\varphi(x) = 1 + x + x^4$$ ← Funkcja wielomianowa

$$Q_1(k+1) = Q_1(k) \oplus Q_4(k)$$

$$Q_2(k+1) = Q_1(k)$$

$$Q_3(k+1) = Q_2(k)$$

$$Q_4(k+1) = Q_3(k)$$

← Metoda obliczania wartości rejestrów w kolejnych iteracjach *k*

- **Realizacja**

Kod:

```
1 reference
private void Generuj_button(object sender, RoutedEventArgs e)
{
    Random rnd = new Random();
    wielomian = WielomianPodany.Text;
    if (wielomian_prev != wielomian)
    {
        wielomian_prev = wielomian;
        ciąg_bitow = null;
        TekstWynikowy.Text = "";
    }
    for (int i = 0; i < wielomian.Length; i++)
    {
        ciąg_bitow += rnd.Next(2);
    }
    Console.WriteLine(ciąg_bitow);
    TekstWynikowy.Text += "Losowy ciąg bitów: " + ciąg_bitow + "\nWyniki: ";

    LFSR();
}
```

```
public void LFSR()
{
    var wynik = 0;
    var ostatni_bit = ciąg_bitow.ElementAt(ciąg_bitow.Length - 1);

    for (int i = 0; i < wielomian.Length; i++)
    {
        if (wielomian[i] == '1')
        {
            wynik = xor(ciąg_bitow[i], ostatni_bit);
        }
    }

    string tmp = "";
    tmp += wynik;

    for (int i = 1; i < ciąg_bitow.Length; i++)
    {
        tmp += ciąg_bitow[i - 1];
    }

    ciąg_bitow = tmp;
    TekstWynikowy.Text += tmp + "\n";
}
```

Na początku wczytano podane przez użytkownika dane, czyli wielomian (wielomian) w postaci kolejnych potęg x w funkcji. Następnie za pomocą funkcji random wygenerowano losowy ciąg bitowy (ciąg_bitow).

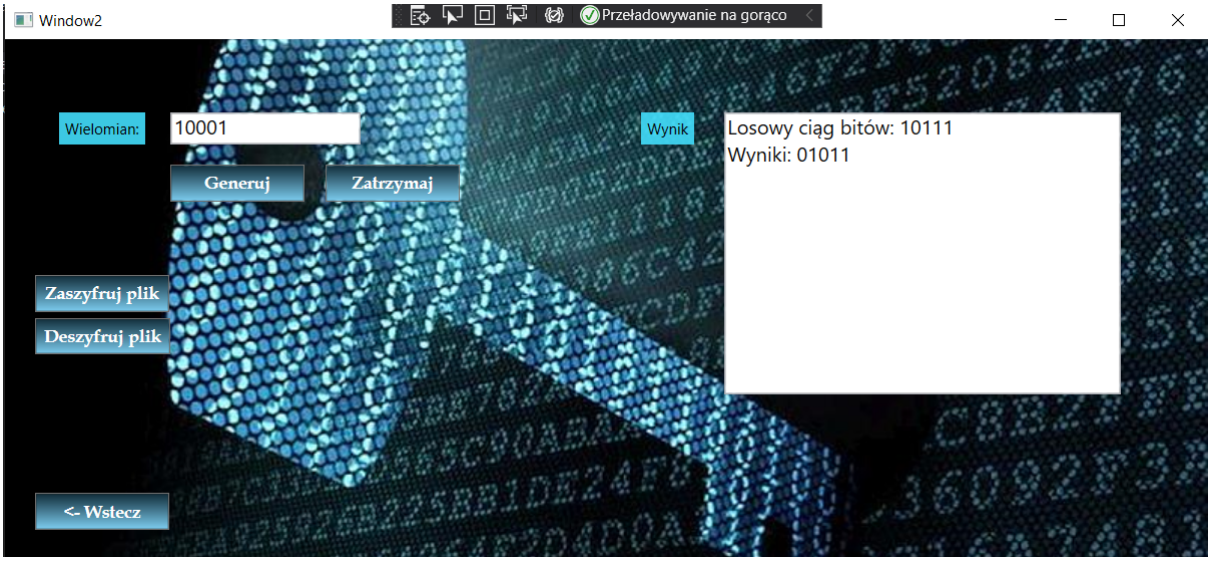
Kolejnym krokiem wywołano funkcję LFSR.

LFSR:

Pierwszym krokiem zadeklarowano niezbędne zmienne do przechowywania wyniku operacji xor i ostatniego bitu. Następnie obliczano wartości rejestrów w kolejnych iteracjach według schematu, czyli wynik operacji xor trafia na pozycję najstarszego bitu, czyli najpierw do zmiennej tmp, a później na pozycję najstarszego bitu zmiennej ciąg_bitow. Pozostałe bity są przesuwane o jedną pozycję w prawo.

Wprowadzono przykładowe dane i sprawdzono działanie algorytmu.

Przykłady:

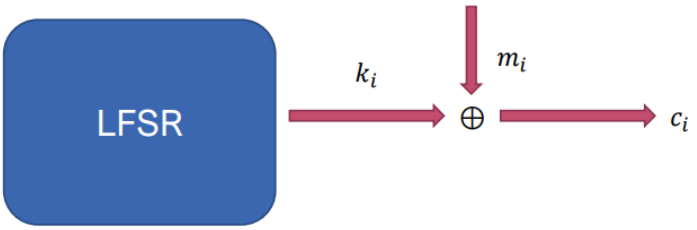


SZYFR STRUMIENIOWY

Zaimplementuj kryptosystem bazujący na schemacie Synchronous Stream Cipher dla podanego wielomianu. Do generowania klucza wykorzystaj metodę LFSR. Na potrzeby działania algorytmu informację jawną przekształć do postaci binarnej. Program powinien mieć możliwość obsługi plików różnego typu (szyfrowanie zarówno plików tekstowych, jak i co najmniej jednego innego formatu plików).

Przykład działania algorytmu:

Synchroniczny szyfr strumieniowy - przykład



gdzie:
k_i - bit klucza wygenerowanego metodą LFSR
m_i - bit tekstu jawnego do zakodowania
c_i - zaszyfrowana postać bitu tekstu (kryptogram)

- **Szyfrowanie**

```
public void Cipher_file(object sender, RoutedEventArgs e)
{
    var ofd = new Microsoft.Win32.OpenFileDialog() {};
    if (ofd.ShowDialog() == false) return;
    string Cipher_me = File.ReadAllText(ofd.FileName);
    string Ciphered = "";
    next = 0;
    for (int i = 0; i < Cipher_me.Length; i++)
    {
        Ciphered += (Char)((Char)Cipher_me[i] + NextByte());
    }
    using (StreamWriter sw = File.CreateText(ofd.FileName))
    {
        sw.Write(Ciphered);
    }
}
```

- **Deszyfrowanie**

```
public void Decipher_file(object sender, RoutedEventArgs e)
{
    var ofd = new Microsoft.Win32.OpenFileDialog() { };
    if (ofd.ShowDialog() == false) return;
    string Cipher_me = File.ReadAllText(ofd.FileName);
    string Ciphered = "";
    next = 0;
    for (int i = 0; i < Cipher_me.Length; i++)
    {
        Ciphered += (Char)((Char)Cipher_me[i] - NextByte());
    }
    using (StreamWriter sw = File.CreateText(ofd.FileName))
    {
        sw.Write(Ciphered);
    }
}
```

Wnioski

- Wszystkie algorytmu zostały wykonane zgodnie z instrukcją i otrzymano wynik którego oczekiwaliśmy.
- Liczby pseudolosowe nie są losowe, ale wyglądają jako losowe.
- Dla większości zastosowań liczby pseudolosowe są lepsze niż losowe.