Politechnika Białostocka

Sprawozdanie z pracowni specjalistycznej

Bezpieczeństwo Sieci Komputerowych

Ćwiczenie numer: 4

Temat: **Implementacja podstawowych modułów kryptograficznych**

Wykonujący ćwiczenie: **Michał Kuczyński**, **Łukasz Litwiński** i **Daniel Pietrzeniuk**

Studia dzienne

Kierunek: Informatyka

Semestr: VI

Grupa zajęciowa: 5

Prowadzący ćwiczenie: mgr inż. Dariusz Jankowski

Data wykonania ćwiczenia:
27 maja 2022r.

Wstęp:

Algorytmy zaimplementowaliśmy w języku C# 9.0, do którego uruchomienia wymagany jest framework .NET 6 (z racji, że jest to język pół-interpretowany).

Aplikacja udostępnia interfejs TUI za pomocą którego można przetestować algorytm, podać seed i wielomian użyty do inicjalizacji generatora oraz następnie tryb pracy, enkrypcja plików w którym podajemy ścieżki do plików jakie chcemy zaszyfrować (lub odszyfrować, algorytm działa dwustronnie), lub tryb ciągłego generowania nowych pseudo-losowych bitów.

Proces szyfrowania wykorzystuje technikę SSC (Synchronous Stream Cipher).

Omówienie kodu:

Kod algorytmu składa się z 3 klas, kolejno: LFSR, LFSR_Generator i LFSR_Encryptor.

Klasa LFSR:

Zaimplementowana w pliku: **src/LFSR.cs**

Dla łatwości w użytkowaniu implementuje interfejs IEnumerable<bool> i służy głównie jako wrapper na LFSR_Generator w którym zaimplementowana jest właściwa część algorytmu.

LFSR przechowuje *Seed* algorytmu, wielomian w surowej postaci (*Polynomial*), wykładniki wielomianu w zmiennej *Taps* i referencję na obiekt klasy LFSR_Generator w zmiennej *generator*. Do pozyskania wykładników wielomianu używana jest funkcja *ParsePolynomial()*.

Poniżej widoczny jest kod tej funkcji, funkcja ta przyjmuje wykładniki wielomianu oddzielone od siebie spacją i przetwarza je na zbiór wykładników.

```
public class LFSR : IEnumerable<bool>
{
    public readonly BigInteger Seed;
    public readonly string Polynomial;
    public readonly HashSet<int> Taps;

    private LFSR_Generator generator;

    public LFSR(BigInteger seed, string polynomial)
    {
        Polynomial = polynomial;
        Taps = ParsePolynomial(polynomial);
        Seed = seed;
        generator = new LFSR_Generator(Seed, Taps);
    }
```

```
// Polynomial format: 32 2 13 => x^32 + x^13 + x^2 + 1
//  (last constant one is assumed by default)
//  (order of exponents is arbitral)
//  (only primitive polynomials are accepted)
private static HashSet<int> ParsePolynomial(string rawPolynomial)
{
    const string ExponentsSeparator = " ";
    var result = new HashSet<int>();
    foreach (string rawExponent in rawPolynomial.Split(ExponentsSeparator))
        result.Add(int.Parse(rawExponent));
    result.Add(0); // Add constant one

    if(result.Count < 2)
        throw new ArgumentException($"Couldn't parse polynomial: {rawPolynomial}");

    return result;
}
```

Kolejną metodą klasy LFSR jest *GetByte()* która pobiera 8 bitów z ciągu lfsr i układa je w bajt który później zwraca. Korzysta przy tym z metody rozszerzeń interfejsu IEnumerator *PopMoveNext(),* której kod widoczny jest z prawej strony.

```
public byte GetByte()
{
    byte result = 0;
    for(int i = 7; i >= 0; i--)
        result |= (byte)((generator.PopMoveNext() ? 1 : 0) << i);
    return result;
}
```

```
/// <summary>
/// Returns Enumerator.Current and moves Enumerator
/// </summary>
public static T PopMoveNext<T>(this IEnumerator<T> self)
{
    T temp = self.Current;
    self.MoveNext();
    return temp;
}
```

Ostatnimi metodami tej klasy do omówienia są implementacje interfejsu IEnumerable odpowiedzialne za iterację.

```
IEnumerator<bool> IEnumerable<bool>.GetEnumerator()
    => GetEnumerator();
IEnumerator IEnumerable.GetEnumerator()
    => GetEnumerator();

public LFSR_Generator GetEnumerator()
    => generator;

public void Reset()
    => generator.Reset();
```

Klasa LFSR_Generator:

Zaimplementowana w pliku: **src/LFSR_Generator.cs**

Dla łatwości w użytkowaniu implementuje interfejs IEnumerator<bool> i implementuje główną część algorytmu LFSR.

Cały jej kod widoczny jest po prawej stronie. Klasa ta przechowuje Seed funkcji LFSR, wykładniki wielomianu, wraz z zcache'owanym największym wykładnikiem, oraz obecny stan iteratora w zmiennej *State*. Klasa ta udostępnia możliwość pobrania obecnego bitu w ciągu przy pomocy właściwości *Current*. Przy pomocy metody Reset() ustawia się stan generatora do stanu początkowego.

Główna część algorytmu (to znaczy generowanie kolejnych bitów ciągu) odbywa się w metodzie *MoveNext()* najpierw *State* jest zamieniany na swoją reprezentację bitową, a następnie dochodzi do wyznaczenia bitu poprzez potraktowanie go funkcją xor z bitami zmiennej *State*. Na koniec zwracany jest true, by oznajmić kodowi korzystającemu z iteratora, że iteracja dalej trwa.

```
public class LFSR_Generator : IEnumerator<bool>
{
    public readonly BigInteger Seed;
    private int[] taps;
    private int taps_Max;
    public BigInteger State { get; private set; }

    public LFSR_Generator(BigInteger seed, HashSet<int> taps)
    {
        Seed = seed;
        this.taps = taps.ToArray();
        taps_Max = taps.Max();
        Reset();
    }

    object IEnumerator.Current => Current;
    public bool Current => Convert.ToBoolean((int)State & 1);

    public void Dispose() { }

    public bool MoveNext()
    {
        string State_string = State.ToBase(2).PadLeft(taps_Max + 1, '0');
        int bit = 0;
        foreach (int tap in taps)
            bit ^= State_string[taps_Max - tap];
        State >>= 1;
        State |= bit << taps_Max;
        return true;
    }

    public void Reset()
        => State = Seed & ((1 << (taps_Max + 1)) - 1);
}
```

Klasa LFSR_Encryptor:

Zaimplementowana w pliku: **src/LFSR_Encryptor.cs**

Odpowiada za funkcjonalność czytania, enkrypcji i zapisywania plików.

```
public class LFSR_Encryptor
{
    public readonly BigInteger Seed;
    public readonly string Polynomial;

    private LFSR _lsfr;

    public LFSR_Encryptor(BigInteger seed, string polynomial)
    {
        Polynomial = polynomial;
        Seed = seed;
        _lsfr = new LFSR(Seed, polynomial);
    }

    /// <summary>
    /// This function is both for encryption and decryption.
    /// </summary>
    public void ShiftFile(string inputFilePath, string? outputFilePath = null)
    {
        Reset();
        if(outputFilePath is null)
            outputFilePath = GetOutputFilePath(inputFilePath);

        using(var inputFile = new BinaryReader(new FileStream(inputFilePath, FileMode.Open, FileAccess.Read)))
        using(var outputFile = new BinaryWriter(new FileStream(outputFilePath, FileMode.Create, FileAccess.Write)))
            while (inputFile.BaseStream.Position != inputFile.BaseStream.Length)
                outputFile.Write(ShiftByte(inputFile.ReadByte()));
    }
    private byte ShiftByte(byte inputByte)
        => (byte)(inputByte ^ _lsfr.GetByte());

    public void Reset()
        => _lsfr.Reset();

    public static string GetOutputFilePath(string inputFilePath)
        => $"{Path.GetFileNameWithoutExtension(inputFilePath)}_out{Path.GetExtension(inputFilePath)}";
}
```

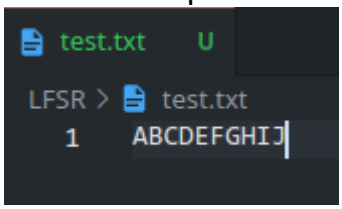
Klasa ta przechowuje seed i wielomian w surowej postaci i tworzy na ich podstawie obiekt klasy LFSR za pomocy którego później szyfruje pliki. Główna funkcjonalność klasy zamyka się w metodzie *ShiftFile()* która pobierając ścieżki do plików, czyta je bajt po bajcie, szyfruje je przy pomocy SSC, a następnie zapisuje te bajty do pliku wynikowego. Funkcja nazywa się ShiftFile() ze względu na funkcję algorytmu SSC który działa tak samo przy enkrypcji i dekrypcji. Z racji na charakterystykę działania algorytmu czytania pliku bajt po bajcie, obsługuje on wszystkie rodzaje plików i nie jest zbyt trudno przerobić go by obsługiwał dowolny strumień danych.

Testy działania:

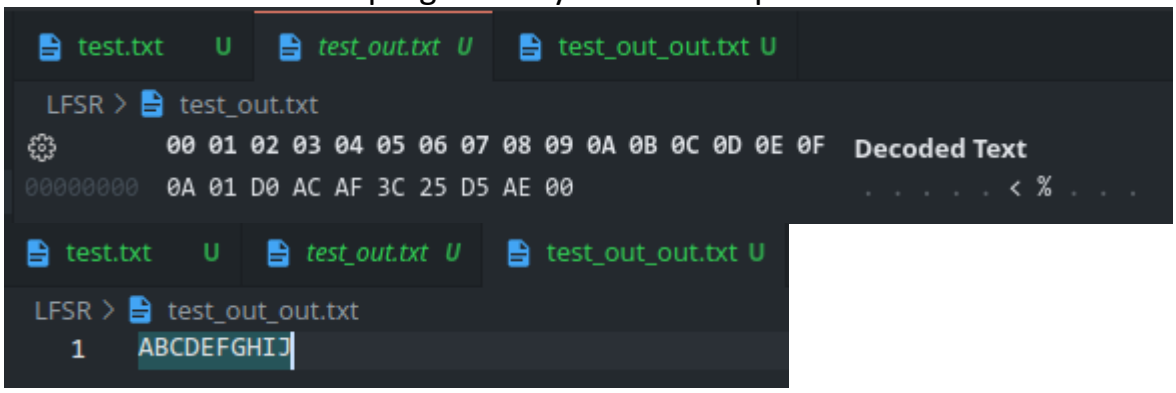
Testowanie algorytmu przeprowadziliśmy przy pomocy prostego kodu użytego bezpośrednio w funkcji *Main()*:

```
BigInteger DefaultSeed = 1234567890;
const string DefaultPolynomial = "32 44 12";

var lfsr = new LFSR_Encryptor(DefaultSeed, DefaultPolynomial);

lfsr.ShiftFile("test.txt");
lfsr.ShiftFile("test_out.txt");
```

Treść pliku testowego test.txt to:



Po uruchomieniu programu wytwarza on 2 pliki:



test_out.txt jest w formie bajtowej, jedynka test_out_out.txt jest dokładną kopią pliku test.txt co udowadnia prawidłowe dwukierunkowe działanie algorytmu.

Przetestowałem także przy pomocy napisanego TUI funkcjonalność generowania ciągu LFSR algorytmu. Algorytm działa poprawnie i w nieskończoność.

Wnioski:

Udało się pomyślnie zaimplementować algorytm LFSR, wraz z szyfrowaniem SSC. LFSR jest ciekawy przykładem jak z prostych instrukcji i konceptów można wytworzyć skomplikowane zachowanie. W używaniu LFSR do celów kryptograficznych kluczowe jest określenie odpowiednich warunków początkowych (wybranie odpowiedniego wielomianu i ziarna), ponieważ generator działa w cyklu, a odpowiednie dobranie tych parametrów pozwala na wydłużenie tego cyklu.

Kod algorytmu rozwijany był na repozytorium git:
https://github.com/ProgramistycznySwir/BSK-4--LFSR-