MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ a MOLDOVEI

FACULTATEA CALCULATOARE, INFORMATICĂ și MICROELECTRONICĂ

Raport

Lucrare de laborator 2

la disciplina Securitatea Informațională

Tema: Algoritmul RSA

A efectuat: st. gr. TI-151 F/R **Constantinescu Nadejda**

A verificat: lect. Superior **Poștaru Andrei**

Chișinău 2020

Sarcina

De studiat algoritmii simetrici de criptare. Algoritmul RSA. De implementat un program în care să fie implementate criptarea și decriptarea folosind algoritmul RSA.

Considerații teoretice

RSA este un algoritm criptografic cu chei publice, primul algoritm utilizat atât pentru criptare, cât și pentru semnătura electronică. RSA este un algoritm de criptare pe blocuri. Aceasta înseamnă că atât textul clar cât și cel cifrat sunt numere între 0 și n-1, cu un n ales. Un mesaj de dimensiune mai mare decât $\log_n$ este împărțit în segmente de lungime corespunzătoare, numite blocuri, care sunt cifrate rând pe rând.[2] De asemenea, ca algoritm criptografic cu chei publice, funcționează pe baza unei perechi de chei legate matematic între ele: o cheie publică, cunoscută de toată lumea, și una secretă, cunoscută doar de deținătorul acesteia. Pașii de bază sunt reprezentați în figura 1 și 2.

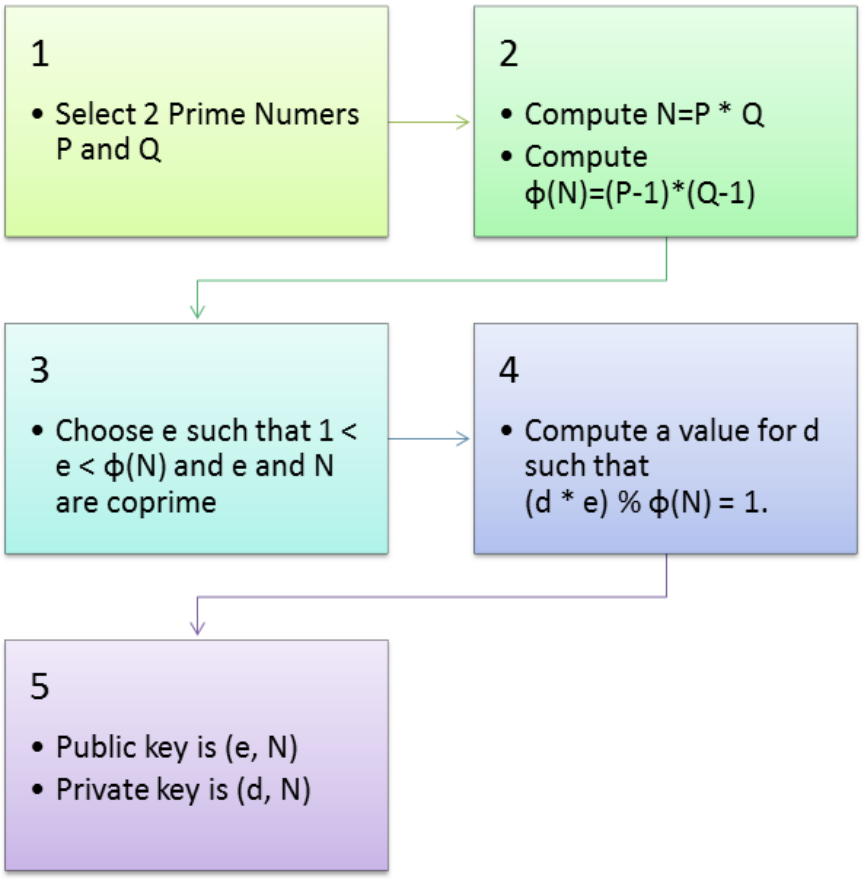

Figura 1- Generarea cheilor a algoritmului RSA

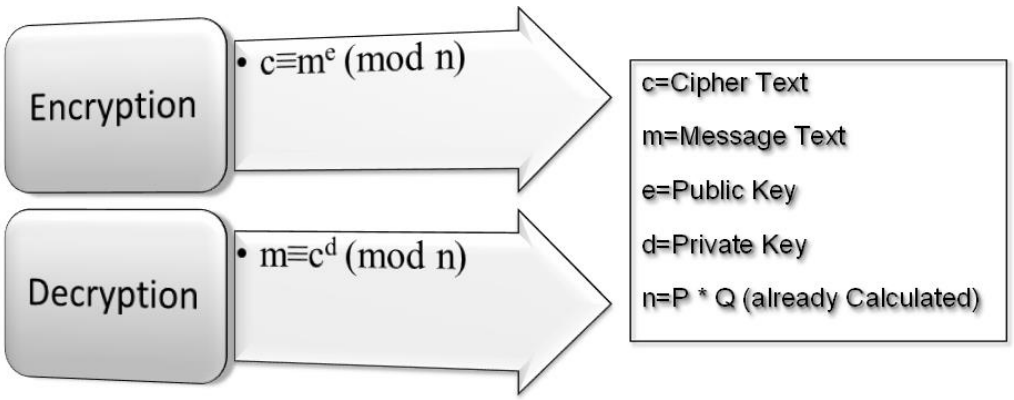

Figura 2-Criptarea și decriptarea a algoritmului RSA

Mersul lucrării

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Numerics;

namespace RSA_code
{
    class RSA
    {
        private byte p; //destroy
        private byte q; //destroy
        private ushort phi; //destroy
        private ushort n;
        private ushort e;
        private Int32 d;

        private struct ExtendedEuclideanResult
        {
            public int u1;
            public int u2;
            public int gcd;
        }

        public RSA() {}

        private void InitKeyData()
        {
            Random random = new Random();

            byte[] simple = GetNotDivideable();
            this.p = simple[random.Next(0, simple.Length)];
            this.q = simple[random.Next(0, simple.Length)];
            this.n = (ushort)(this.p * this.q);
            this.phi = (ushort)((p - 1) * (q - 1));
            List<ushort> possibleE = GetAllPossibleE(this.phi);

            do
            {
                this.e = possibleE[random.Next(0, possibleE.Count)];
                this.d = ExtendedEuclide(this.e % this.phi, this.phi).u1;
```

```
    } while (this.d < 0);
}

public Int32 GetNKey()
{
    return this.n;
}

public Int32 GetDKey()
{
    return this.d;
}

public string encode(string text)
{
    InitKeyData();

    string outStr = "";
    System.Text.UTF8Encoding enc = new System.Text.UTF8Encoding();
    byte[] strBytes = enc.GetBytes(text);
    foreach (byte value in strBytes)
    {
        int encryptedValue = ModuloPow(value, this.e, this.n);
        outStr += encryptedValue + "!";
    }

    return outStr;
}

public string decode(string text, string n_s, string d_s)
{
    string outStr = "";
    Int32 n = Int32.Parse(n_s);
    Int32 d = Int32.Parse(d_s);
    Int32[] arr = GetDecArrayFromText(text);
    byte[] bytes = new byte[arr.Length];
    System.Text.UTF8Encoding enc = new System.Text.UTF8Encoding();
    int j=0;
    foreach (int i in arr)
    {
        byte decryptedValue = (byte)ModuloPow(i, d, n);

        bytes[j] = decryptedValue;
        j++;

    }
    outStr += enc.GetString(bytes);
    return outStr;
}

private Int32[] GetDecArrayFromText(string text)
{
    int i = 0;
    foreach (char c in text)
    {
        if (c == '!')
        {
            i++;
        }
    }
```

```
    Int32[] result = new Int32[i];
    i = 0;

    string tmp = "";

    foreach (char c in text)
    {
        if (c != '!')
        {
            tmp += c;
        }
        else
        {
            result[i] = Int32.Parse(tmp);
            i++;
            tmp = "";
        }
    }

    return result;
}

static int ModuloPow(int value, int pow, int modulo)
{
    int result = value;
    for (int i = 0; i < pow - 1; i++)
    {
        result = (result * value) % modulo;
    }
    return result;
}

/// obtain all possible variants for e
static List<ushort> GetAllPossibleE(ushort phi)
{
    List<ushort> result = new List<ushort>();

    for (ushort i = 2; i < phi; i++)
    {
        if (ExtendedEuclide(i, phi).gcd == 1)
        {
            result.Add(i);
        }
    }

    return result;
}

/// <summary>
/// u1 * a + u2 * b = u3
/// </summary>
/// <param name="a">Number a</param>
/// <param name="b">Module of number</param>
private static ExtendedEuclideanResult ExtendedEuclide(int a, int b)
{
    int u1 = 1;
    int u3 = a;
    int v1 = 0;
    int v3 = b;
```

```
        while (v3 > 0)
        {
            int q0 = u3 / v3;
            int q1 = u3 % v3;

            int tmp = v1 * q0;
            int tn = u1 - tmp;
            u1 = v1;
            v1 = tn;

            u3 = v3;
            v3 = q1;
        }

        int tmp2 = u1 * (a);
        tmp2 = u3 - (tmp2);
        int res = tmp2 / (b);

        ExtendedEuclideanResult result = new ExtendedEuclideanResult()
        {
            u1 = u1,
            u2 = res,
            gcd = u3
        };

        return result;
    }

    static private byte[] GetNotDivideable()
    {
        List<byte> notDivideable = new List<byte>();

        for (int x = 2; x < 256; x++)
        {
            int n = 0;
            for (int y = 1; y <= x; y++)
            {
                if (x % y == 0)
                    n++;
            }

            if (n <= 2)
                notDivideable.Add((byte)x);
        }
        return notDivideable.ToArray();
    }

}
                                                    }
```

Generarea cheilor

Perechea de chei se generează conform următorilor pași:

1. Se generează două numere prime, de preferat mari, p și q

```
byte[] simple = GetNotDivideable();
this.p = simple[random.Next(0, simple.Length)];
this.q = simple[random.Next(0, simple.Length)];
```

2. Se calculează$N = P * Q$și$\phi = (P - 1)(Q - 1)$

```
this.n = (ushort)(this.p * this.q);
this.phi = (ushort)((p - 1) * (q - 1));
```

3. Se calculează $1 < \varepsilon < \phi$ și astfel încât$cmmdc(\varepsilon, \phi) = 1$

```
    static List<ushort>GetAllPossibleE(ushort phi)
{List<ushort> result = new List<ushort>();
            for (ushorti = 2; i< phi; i++) {
                if (ExtendedEuclide(i, phi).gcd == 1) {
result.Add(i);
                }
            }
            return result;
        }
```

4. Folosind algoritmul lui Euclid extins, se calculează întregul d, unicul cu proprietatea că$de \equiv 1 \; mod \; \phi(n, d)$ Ea constituie cheia secretă.

```
do{
       this.e = possibleE[random.Next(0, possibleE.Count)];
       this.d = ExtendedEuclide(this.e % this.phi, this.phi).u1;
} while (this.d< 0);
```

Decizia cu privire la care dintre *e* și *d* este cheia publică și care este cea secretă este, din punct de vedere matematic, arbitrară, oricare dintre cele două numere poate juca oricare dintre roluri.

Criptarea și decriptarea

Criptarea

Presupunând că mesajul clar este sub forma unui număr m, mai mic decât n, atunci mesajul cifrat, notat cu c este$c = \; m^{\varepsilon}(mod \; n)$unde ε este cheia publică a destinatarului mesajului:

```
public string encode(string text)
{
    string outStr = "";
    System.Text.UTF8Encoding enc = new System.Text.UTF8Encoding();
    byte[] strBytes = enc.GetBytes(text);
    foreach (byte value in strBytes)
    {
        int encryptedValue = ModuloPow(value, this.e, this.n);
```

```
        outStr += encryptedValue + "|";
    }
    return outStr;
}
```

Decriptarea

Pentru a decripta mesajul, destinatarul își folosește cheia sa secretă d, care are proprietatea foarte importantă că $de \equiv 1 \, mod \, \phi(n,d)$Astfel, mesajul clar este recuperat calculând:$m = c^d (mod\ n)$.

```
public string decode(string text, string n_s, string d_s)
{string outStr = "";
       Int32 n = Int32.Parse(n_s);
       Int32 d = Int32.Parse(d_s);
            Int32[] arr = GetDecArrayFromText(text);
            byte[] bytes = new byte[arr.Length];
            System.Text.UTF8Encoding enc = new System.Text.UTF8Encoding();
            int j=0;
            foreach (int i in arr)
            {
                byte decryptedValue = (byte)ModuloPow(i, d, n);
                bytes[j] = decryptedValue;
                j++;
            }
       outStr += enc.GetString(bytes);
       return outStr;}
```

Oricine poate cripta mesaje cu cheia publică a destinatarului, dar numai acesta din urmă poate decripta, deoarece trebuie să folosească cheia sa secretă.Algoritmul poate fi folosit și pentru semnătura electronică, folosind cheile invers. Dacă o entitate criptează un mesaj (sau mai degrabă un hash al acestuia) cu cheia sa secretă și atașează rezultatul mesajului său, atunci oricine poate verifica, decriptând cu cheia publică a semnatarului și comparând rezultatul cu mesajul clar (sau cu hash-ul acestuia), că într-adevăr acea entitate este autorul mesajului.

La execuția programului s-au obținut următoarele rezultate afișate în figura 3.

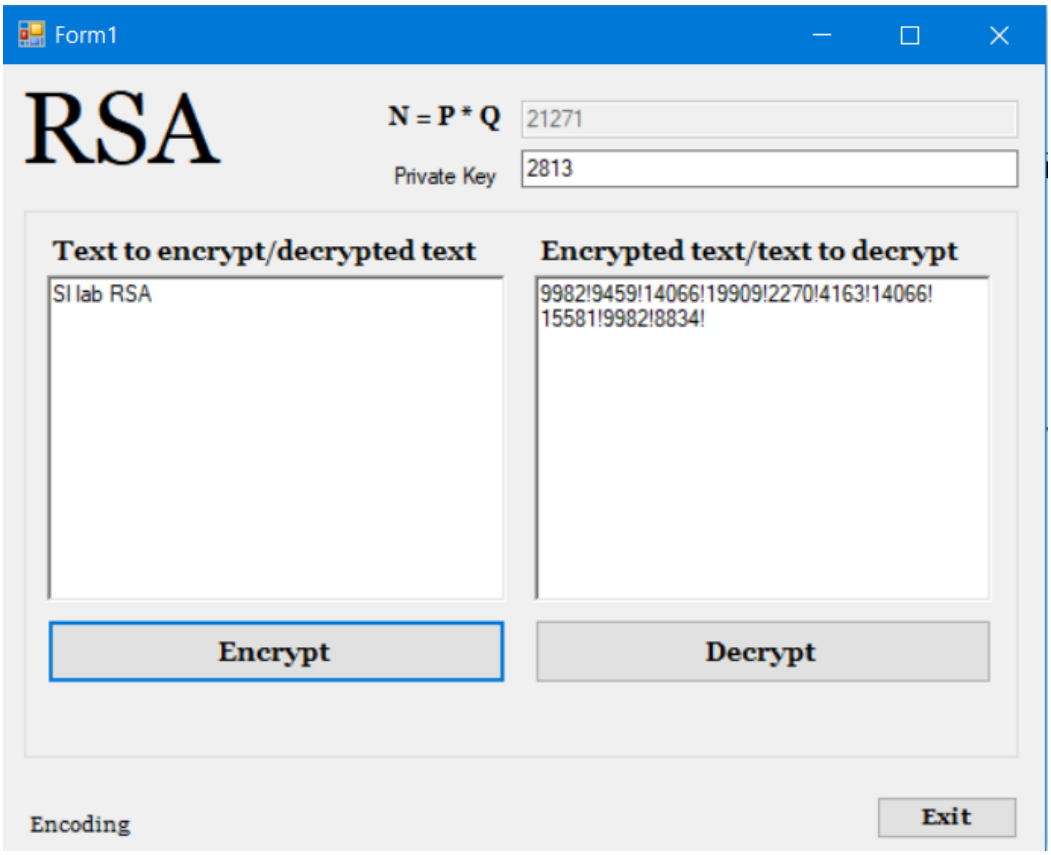

Figura 3- Rezultate obținute

Concluzie

In aceasta lucrare de laborator am studiat cum lucreaza algoritm RSA, care este relativ lent si din aceasta cauza este rar utilizat direct pentru criptarea datelor utilizatorului. Algoritmu RSAse utilizeaza pentru a crepta cheile criptografie simetrica. Implementarea algoritmului RSA este costisitor din punct de vedere a resurselor folosite si a timpuli de calcul care este lung.

O implementarea hardware de RSA este de o mie de ori mai lentă decât implementarea algoritmului DES, iar in software, RSA este de 100 de ori mai lent. Există posibilitatea de a spori performanța algoritmului precum alegerea unui exponent de criptare mai mic, care astfel reduce calcule necesare criptării rezolvând în același fel și unele probleme de securitate.

Bibliografie

1 Algoritmul RSA. Resursă electronică:
https://ro.wikipedia.org/wiki/RSA

2 Code ProjectImage Cryptography using RSA Algorithm in C#. Resursăelectronică:
https://www.codeproject.com/Articles/723175/Image-Cryptography-using-RSA-Algorithm-in-Csharp

3 Sourabh Somani, RSA Algorithm With C#. Resursăelectronică:
http://www.c-sharpcorner.com/uploadfile/75a48f/rsa-algorithm-with-c-sharp2/