

**Технически университет – София**

**Факултет по приложна математика и информатика**

**Курсов проект**

**на тема**

**Криптографски алгоритъм RSA**

**Изготвил: Денис Пламенов Цолов**

**Фак. номер: 471220038**

**Група: 78**

1. **Какво представлява криптографският алгоритъм RSA?**

RSA(1977) е асиметричен криптографски алгоритъм, което означава, че работи с два ключа – публичен и частен. Публичният се разпространява свободно и се използва за криптиране на съобщения. Частният е таен и го притежава само човекът, генерирал ключовете. С него се декриптират съобщенията.

Сигурността на алгоритъма се основава на това, че за много големите числа(получени от умножението на 2 прости числа) е трудно да се намерят обратно съставните им множители. Силата на криптиране е директно свързана с размера на ключа като удвояване на размера му оказва експоненциално нарастване в сигурността. Обикновено ключовете, използвани от RSA са с дължина 1024 или 2048 бита.

RSA е утвърден като един от най-използваните криптографския алгоритми – многобройни протоколи, софтуерни програми и браузъри използват доказаните възможности на RSA както за криптиране и декриптиране, така и за електронен подпис. Използването му за криптиране на голям обем данни обаче е недостатъчно бързо и отнемащо много изчислителен ресурс. Поради тази причина RSA често се използва в комбинация с друг(симетричен) алгоритъм като задачата на RSA в тази ситуация е да предаде успешно тайния ключ на другия алгоритъм.

1. **Програмно реализиране на RSA. Стъпки на алгоритъма.**

Алгоритъмът се състой от 3 основни функции:

* Генериране на ключовете – това е най-сложната част от реализацията. Потребителят отговарящ за тази стъпка генерира две големи прости положителни числа p и q. Те се секретни параметри, а произведението им е числото N, което е несекретно и присъства и в публичния, и в секретния ключ. Числото Ф(N) = (p-1)\*(q-1) е секретен параметър. Потребителят генерира и самия публичен ключ Pk, който удовлетворява условията да бъде взаимно прост с Ф(N) и да принадлежи на интервала [1< Pk< Ф(N)]. Частният ключ Sk се изчислява като мултипликативна инверсия на Pk  по модул Ф(N): **SK=PK(phi(Ф(N))-1)modФ(N),** където phi(Ф(N))=Ф(N)(1-1/m1)(1-1/m2)...(1-1/mn) - функция на Ойлер, m1, m2,...,mn - прости делители на Ф(N).
* Криптиране – Поредицата от символи(Plain text) се превръща в числови вид (чрез ASCII Table) и всеки от тях се криптира като се повдига на степен Pk, последвано от модулно деление на N (**XPKmodN=Y**).
* Декриптиране – Всяко отделно число от криптограмата се повдига на степен Sk, последвано от модулно деление на N (**YSKmodN=X).** Полученото число се превръща обратно в символ (отново чрез ASCII Table)

1. **Програмен код и снимков материал от създадената програма.**

(Програмата е тествана и работи без грешки)

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**using System.Windows.Forms;**

**using System.Numerics;**

**namespace RSA**

**{**

**public partial class Form1 : Form**

**{**

**public Form1()**

**{**

**InitializeComponent();**

**}**

**private int regime = 0;**

**public static bool IsPrime(int number)**

**{**

**if (number <= 1) return false;**

**if (number == 2) return true;**

**if (number % 2 == 0) return false;**

**var boundary = (int)Math.Floor(Math.Sqrt(number));**

**for (int i = 3; i <= boundary; i += 2)**

**if (number % i == 0)**

**return false;**

**return true;**

**}**

**public static int gcd(int a, int b)**

**{**

**if (a == 0 || b == 0)**

**return 0;**

**// base case**

**if (a == b)**

**return a;**

**// a is greater**

**if (a > b)**

**return gcd(a - b, b);**

**return gcd(a, b - a);**

**}**

**public static bool coprime(int a, int b)**

**{**

**if (gcd(a, b) == 1)**

**return true;**

**else**

**return false;**

**}**

**public static int phi(int n)**

**{**

**int result = 1;**

**for (int i = 2; i < n; i++)**

**if (gcd(i, n) == 1)**

**result++;**

**return result;**

**}**

**public static bool IsDigitsAndCommasOnly(string str)**

**{**

**for (int i = 0; i < str.Length; i++)**

**{**

**if(str[i] != ',')**

**if (str[i] < '0' || str[i] > '9')**

**return false;**

**if (i == 0 || i == str.Length - 1) //Checks for incorrect ',' placement.**

**if (str[i] == ',')**

**return false;**

**}**

**return true;**

**}**

**private BigInteger Sk = 0;**

**private int Pk = 0;**

**private int N = 0;**

**private int p = 0;**

**private int q = 0;**

**private void button1\_Click(object sender, EventArgs e)**

**{**

**if (label1.Text == "This is RSA cryptosystem." || label1.Text == "Cryptogram to decrypt:") // Leads to enter Encryption mode.**

**{**

**regime = 1;**

**}**

**if (regime == 0) // Encrypts**

**{**

**String plainText = "";**

**plainText = textBox4.Text;**

**int.TryParse(textBox1.Text, out N);**

**int.TryParse(textBox3.Text, out Pk);**

**if (N > 0 && Pk > 0 && plainText != "")**

**{**

**label7.Visible = true;**

**label7.Text = "Cryptogram:";**

**textBox7.Visible = true;**

**byte[] asciiBytes = Encoding.ASCII.GetBytes(plainText);**

**int[] resultArray = new int[asciiBytes.Length];**

**String cryptogram = "";**

**for (int i = 0; i < asciiBytes.Length; i++)**

**{**

**resultArray[i] = (int)BigInteger.ModPow(int.Parse(asciiBytes[i].ToString()), Pk, N);**

**cryptogram += resultArray[i].ToString() + ",";**

**}**

**textBox7.Text = cryptogram.Substring(0, cryptogram.Length - 1);**

**}**

**else**

**{**

**textBox7.Visible = false;**

**label7.Visible = true;**

**label7.Text = "Incorrect input.";**

**if(N<0)**

**textBox1.Text = "";**

**if(Pk < 0)**

**textBox3.Text = "";**

**}**

**}**

**else if (regime == 1) // Enters Encryption mode**

**{**

**label2.Visible = true;**

**label4.Visible = true;**

**textBox3.Visible = true;**

**textBox4.Visible = true;**

**textBox4.Text = "";**

**label7.Visible = false;**

**textBox7.Text = "";**

**textBox7.Visible = false;**

**regime = 0;**

**label1.Text = "Plain text to encrypt:";**

**label3.Visible = false;**

**textBox2.Visible = false;**

**textBox1.Visible = true;**

**label4.Text = "Public key:";**

**label2.Text = "N:";**

**if (Pk != 0)**

**textBox3.Text = Pk.ToString();**

**else textBox3.Text = "";**

**if (N != 0)**

**textBox1.Text = N.ToString();**

**label1.Padding = new Padding(236, 0, 0, 0);**

**}**

**}**

**private void button2\_Click(object sender, EventArgs e)**

**{**

**if(label1.Text == "This is RSA cryptosystem." || label1.Text == "Plain text to encrypt:") // Leads to enter decryption mode.**

**{**

**regime = 1;**

**}**

**if(regime == 0) // Decrypts**

**{**

**String cryptogram = "";**

**cryptogram = textBox4.Text;**

**int.TryParse(textBox1.Text, out N);**

**BigInteger.TryParse(textBox3.Text, out Sk);**

**if(N > 0 && Sk > 0 && cryptogram != "" && IsDigitsAndCommasOnly(cryptogram))**

**{**

**label7.Visible = true;**

**label7.Text = "Plain Text:";**

**textBox7.Visible = true;**

**List<String> cryptogramArray = new List<string>();**

**foreach (string asciiChar in cryptogram.Split(','))**

**{**

**cryptogramArray.Add(asciiChar);**

**}**

**int length = cryptogramArray.ToArray().Length;**

**String plainText = "";**

**for (int i = 0; i < length; i++)**

**{**

**cryptogramArray[i] = BigInteger.ModPow(BigInteger.Parse(cryptogramArray[i]), Sk, N).ToString();**

**plainText += (char)int.Parse(cryptogramArray[i]);**

**}**

**textBox7.Text = plainText;**

**if (textBox7.Text == "")**

**textBox7.Text = "You are trying to print unprintable charachters. :(";**

**}**

**else**

**{**

**textBox7.Visible = false;**

**label7.Visible = true;**

**label7.Text = "Incorrect input.";**

**if (N < 0)**

**textBox1.Text = "";**

**if (Sk < 0)**

**textBox3.Text = "";**

**}**

**}**

**else if(regime == 1) // Enters decryption mode**

**{**

**label7.Visible = false;**

**label2.Visible = true;**

**label4.Visible = true;**

**textBox3.Visible = true;**

**textBox4.Visible = true;**

**textBox4.Text = "";**

**textBox7.Text = "";**

**textBox7.Visible = false;**

**regime = 0;**

**label1.Text = "Cryptogram to decrypt:";**

**label3.Visible = false;**

**textBox2.Visible = false;**

**textBox1.Visible = true;**

**label4.Text = "Secret key:";**

**label2.Text = "N:";**

**if (Sk != 0)**

**textBox3.Text = Sk.ToString();**

**else textBox3.Text = "";**

**if (N != 0)**

**textBox1.Text = N.ToString();**

**label1.Padding = new Padding(236, 0, 0, 0);**

**}**

**}**

**private void button3\_Click(object sender, EventArgs e)**

**{**

**if (regime == 0) // Enters key generation**

**{**

**label7.Visible = false;**

**regime = 1;**

**label1.Padding = new Padding(220, 0, 0, 0);**

**label1.Text = "Please input your parameters:";**

**label2.Visible = true;**

**label2.Text = "p:";**

**label3.Visible = true;**

**label4.Visible = true;**

**textBox4.Visible = false;**

**label4.Text = "Public key:";**

**textBox1.Visible = true;**

**textBox2.Visible = true;**

**textBox3.Visible = true;**

**textBox7.Visible = false;**

**if (p != 0)**

**textBox1.Text = p.ToString();**

**else textBox1.Text = "";**

**if (q != 0)**

**textBox2.Text = q.ToString();**

**if (Pk != 0)**

**textBox3.Text = Pk.ToString();**

**}**

**else if (regime == 1) // Key generation**

**{**

**int fN = 0;**

**if (int.TryParse(textBox1.Text.ToString(), out p) && int.TryParse(textBox2.Text.ToString(), out q) && int.TryParse(textBox3.Text.ToString(), out Pk))**

**{**

**if (Pk < 0) // resets the saved data in case of bad keygen input**

**{**

**Pk = 0;**

**Sk = 0;**

**}**

**fN = (p - 1) \* (q - 1);**

**if(!IsPrime(p) || !IsPrime(q) || p<=10 || q<=10)**

**{**

**label1.Padding = new Padding(175, 0, 0, 0);**

**label1.Text = "Invalid input: P and q must be prime numbers > 10.";**

**textBox4.Visible = false;**

**textBox5.Visible = false;**

**label6.Visible = false;**

**label5.Visible = false;**

**textBox6.Visible = false;**

**}**

**else if (Pk <= 1 || Pk > fN)**

**{**

**label1.Padding = new Padding(215, 0, 0, 0);**

**label1.Text = "Invalid input: 1 < Pk < (p-1)\*(q-1).";**

**textBox4.Visible = false;**

**textBox5.Visible = false;**

**label6.Visible = false;**

**label5.Visible = false;**

**textBox6.Visible = false;**

**}**

**else if (!coprime(fN, Pk))**

**{**

**label1.Padding = new Padding(180, 0, 0, 0);**

**label1.Text = "Invalid input: Pk should be coprime with (p-1)\*(q-1).";**

**textBox4.Visible = false;**

**textBox5.Visible = false;**

**label6.Visible = false;**

**label5.Visible = false;**

**textBox6.Visible = false;**

**}**

**else**

**{**

**N = p \* q;**

**Sk = BigInteger.ModPow(Pk, phi(fN) - 1,fN);**

**textBox5.Visible = true;**

**label6.Visible = true;**

**label5.Text = "Your Public key:";**

**textBox6.Visible = true;**

**label5.Visible = true;**

**label6.Text = "Your Secret key:";**

**textBox6.Text = "Pk = " + Pk.ToString() + ", N = " + N;**

**textBox5.Text = "Sk = " + Sk.ToString() + ", N = " + N;**

**label1.Padding = new Padding(265, 0, 0, 0);**

**label1.Text = "Success!";**

**}**

**}**

**else**

**label1.Text = "Invalid input. Please use only numbers.";**

**}**

**}**

**}**

**}**

