
MINISTERUL EDUCAȚIEI AL REPUBLICII MOLDOVA
UNIVERSITATEA TEHNICĂ A MOLDOVEI
FACULTATEA CALCULATOARE, INFORMATICĂ ȘI MICROELECTRONICĂ
DEPARTAMENTUL INGINERIE SOFTWARE ȘI AUTOMATICĂ

LUCRARE DE LABORATOR NR.1

Disciplina : *Securitatea Informațională*

Tema : *Algoritmul de criptare RSA*

A elaborat : st.gr.TI-171 f/r , Florea Cristina

A verificat : lect.univ. Poștaru Andrei

2021

Scopul lucrării.

Studierea , analiza metodologiei și implementarea algoritmului RSA.

Algoritmul RSA

RSA , în criptografie, este un algoritm cu chei publice, primul algoritm utilizat atât pentru criptare cât și pentru semnătura digitală. Acest algoritm a fost dezvoltat în 1977 și publicat în 1978 de Ron Rivest, Adi Shamir și Leonard Adleman la MIT și își trage numele de inițialele numelor celor trei autori.

Puterea sa criptografică se bazează pe dificultatea problemei factorizării numerelor întregi, problema la care se reduce criptanaliza RSA și pentru care toți algoritmi de rezolvare cunoscuți au complexitate exponențială.

Există însă câteva metode de criptanaliză care ocolesc factorizarea efectivă , exploatând maniere eronate de implementare efectivă a schemei de criptare.

Funcționare

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.

Ca algoritm criptografic cu chei public, acesta funcționează pe baza unei perechi de chei legate matematic între ele : o cheie publică, cunoscută de toată lumea, și una secretă, necunoscută decât de deținătorul acesteia.

Generarea cheilor

Perechea de chei se generează după următorii pași :

1. Se generează 2 numere prime , de preferat mari , p și q
2. Se calculează $n = pq$ și $\Phi = (p-1)(q-1)$
3. Se alege un întreg aliat e , $1 < e < \Phi$ astfel încât $\text{cmmdc}(e, \Phi) = 1$. Perechea (n, e) este cheia publică.
4. Folosind algoritmul lui Euclid extins , se calculează întreg d , unicul cu proprietatea ca $ed = 1 \bmod \Phi$. (n, d) constituie cheia secretă

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 2 numere poate juca oricare dintre roluri.

În practică însă , pentru a mări viteza de criptare ,și întrucât dintre cele 2 numere e este cel ales arbitrar, e este cheia public iar valoarea sa este aleasă u număr mai mic, de exemplu 3, 17 sau 65537 ($2^{16}+1$). Acesta conduce la un număr minim de înmulțiri , deci la o performanță sporită, deoarece aceste numere au doar 2 cifre 1 în reprezentarea lor binară.

Criptarea și decriptarea

Presupunând că mesajul clar este sub forma unui număr m , mai mic decât n , atunci mesajul cifrat, notat cu c este :

$$c = m^e \pmod{n}$$

unde e este cheia publică a destinatarului mesajului.

Pentru a decripta mesajul, destinatarul își folosește cheia sa secretă d , care are proprietatea foarte importantă ca :

$$de = 1 \pmod{\Phi}$$

Astfel, mesajul clar este recuperat calculând :

$$m = c^d \pmod{n}$$

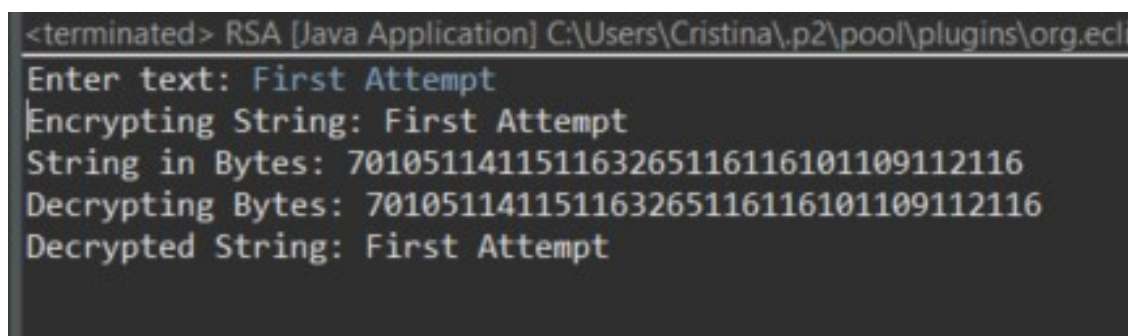
Oricine poate cripta mesaje cu cheia publică a destinatarului, dar numai acesta din urmă poate decripta, deoarece trebuie să folosească cheia sa secretă.

Algoritmul poate cripta mesaje cu cheia publică a destinatarului, dar numai acesta din urmă poate decripta, deoarece trebuie să folosească cheia sa secretă.

Dacă o entitate criptează un mesaj (un hash al acestuia) cu cheia sa secretă, și atașează rezultatul mesajului sau, atunci oricine poate verifica, decriptând cu cheia publică a semnatarului și comparând rezultatul cu mesajul clar (sau hash-ul acestuia), că într-adevăr acea entitate este autorul mesajului.

Implementarea algoritmului.

Câteva exemple practice a funcționalității algoritmului RSA.



```
<terminated> RSA [Java Application] C:\Users\Cristina\.p2\pool\plugins\org.ecj
Enter text: First Attempt
Encrypting String: First Attempt
String in Bytes: 701051141151163265116116101109112116
Decrypting Bytes: 701051141151163265116116101109112116
Decrypted String: First Attempt
```

```
Problems Javadoc Declaration Console X
<terminated> RSA [Java Application] C:\Users\Cristina\.p2\pool\plugins\org.eclipse.justj
Enter text: Cristina
Encrypting String: Cristina
String in Bytes: 6711410511511610511097
Decrypting Bytes: 6711410511511610511097
Decrypted String: Cristina
```

```
Problems Javadoc Declaration Console X
<terminated> RSA [Java Application] C:\Users\Cristina\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657
Enter text: Florea Cristina , grupa TI - 171 F/R
Encrypting String: Florea Cristina , grupa TI - 171 F/R
String in Bytes: 70108111114101973267114105115116105110973244321031141171129732847332453249554932704782
Decrypting Bytes: 70108111114101973267114105115116105110973244321031141171129732847332453249554932704782
Decrypted String: Florea Cristina , grupa TI - 171 F/R
```

Concluzii.

În procesul realizării acestui raport ,a fost studiat algoritmul de criptare RSA și a fost implementat și aplicat în practică acesta . Au fost introduce mai multe exemple pentru a vizualiza cum se criptează , cum se vede textul inițial, și cum se decriptează acesta.

Codul sursă.

```
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;

public class RSA {
    private BigInteger p;
    private BigInteger q;
    private BigInteger N;
    private BigInteger phi;
    private BigInteger e;
    private BigInteger d;
    private int bitlength = 1024;
    private Random r;

    public RSA()
    {
        r = new Random();
        p = BigInteger.probablePrime(bitlength, r);
        q = BigInteger.probablePrime(bitlength, r);
        N = p.multiply(q);
        phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(bitlength / 2, r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) <
0)
        {
            e.add(BigInteger.ONE);
        }
        d = e.modInverse(phi);
    }

    public RSA(BigInteger e, BigInteger d, BigInteger N)
    {
        this.e = e;
        this.d = d;
        this.N = N;
    }

    @SuppressWarnings("deprecation")
    public static void main(String[] args) throws IOException
    {
        RSA rsa = new RSA();
        DataInputStream in = new DataInputStream(System.in);
        String teststring;
        System.out.print("Enter the plain text: ");
        teststring = in.readLine();
        System.out.println("Encrypting String: " + teststring);
        System.out.println("String in Bytes: "
            + bytesToString(teststring.getBytes()));
        // encrypt
        byte[] encrypted = rsa.encrypt(teststring.getBytes());
        // decrypt
        byte[] decrypted = rsa.decrypt(encrypted);
        System.out.println("Decrypting Bytes: " + bytesToString(decrypted));
        System.out.println("Decrypted String: " + new String(decrypted));
    }

    private static String bytesToString(byte[] encrypted)
    {
        String test = "";
```

```

        for (byte b : encrypted)
        {
            test += Byte.toString(b);
        }
        return test;
    }

    // Encrypt message
    public byte[] encrypt(byte[] message)
    {
        return (new BigInteger(message)).modPow(e, N).toByteArray();
    }

    // Decrypt message
    public byte[] decrypt(byte[] message)
    {
        return (new BigInteger(message)).modPow(d, N).toByteArray();
    }
}

```