JAVA IMPLEMENTED ENCRYPTION ALGORITHM

Luminiţa SCRIPCARIU1, Andrei ALISTAR2, Mircea Daniel FRUNZĂ3
Technical University “Gh.ASACHI” of Iaşi, ROMANIA
Bd. Copou 11, RO – 700506 Iasi
1luminita.scripcariu@gmail.com

Abstract. Secure communications involves the use of different cryptography methods, with digital keys longer than the transmitted message [1]. Many applications dedicated to text transmission request special encryption algorithms. A secure encryption algorithm, with a great diversity of the key space, is proposed in this paper, based on polynomial invertible functions defined on Galois Fields [2].

Keywords: communication security, encryption, algorithm, Galois field.

Invertible Functions Defined on Galois Fields

A class of 3-parameters polynomial invertible functions defined on Galois Fields, with m-bits symbols, can be used for cryptography:

$$E_k(a) = \bar{c} = k_1 + k_2 \bar{a}^{-k_3}, k_2 \neq 0, k_3 \neq 0$$

$$k_3 \neq 2^n - 1, \bar{k} = [k_1, k_2, k_3]. \tag{1}$$

The data vector \( \bar{a} = a_1 a_2 ... a_m \) is an element of the definition field GF \((2^m)\) and the m bits \(a_1, a_2 \ldots a_m\) are obtained by decimal-to-binary conversion of this element.

The inverted functions, defined on the same GF, are:

$$E_k^{-1}(\bar{c}) = \left[k_1^{-1}(\bar{c} + k_3)\right]^q \tag{2}$$

The inverse key component \(q\) verifies:

$$\bar{a}^q = \bar{a} \tag{3}$$

$$(k_2 \cdot q) \mod (2^n - 1) = 1 \tag{4}$$

The maximum number of all 3-parameters polynomial invertible functions defined on a GF \((2^m)\) is given by:

$$M = 2^n \cdot (2^n - 1) \cdot (2^n - 2) \tag{5}$$

It can be approximated exponentially as:

$$M \approx ct \cdot 10^N, 1 \leq ct < 10 \tag{6}$$

The existence of \(q\) for all \(k_2\) values is guaranteed only if \(m\) and \(2^m - 1\) are simultaneously prime numbers, when all the elements of the GF have the same order equal to \(2^m - 1\) and the maximum number of simple encryption functions defined on a GF \((2^m)\) is obtained (Table 1).

For example, on GF \((8)\) there are two symmetric couples \((k_2, q), (1 - 1)\) and \((6 - 6)\), and two asymmetric couples: \((2 - 4)\) with \((4 - 2), (3 - 5)\) with \((5 - 3)\).

These functions are applied on each data block of m bits, with a special set of parameters, randomly generated with the initial state \((s)\) defined by the public key. All the arithmetic operations (addition, multiplication) are defined on the GF.

Table1. Optimum GF Dimensions

<table>
<thead>
<tr>
<th>m</th>
<th>(2^m-1)</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>8191</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>131 071</td>
<td>15</td>
</tr>
<tr>
<td>19</td>
<td>524 287</td>
<td>17</td>
</tr>
<tr>
<td>31</td>
<td>2 147 483 647</td>
<td>27</td>
</tr>
</tbody>
</table>

Chaotic Number Generators

The key-vector \(\bar{k}\) contains the encryption function parameters and it is used as the key sequence for encryption. This is given by a chaotic number generator based on a dynamic system described by the logistic function:

$$x_{k+1} = R x_k \cdot (1 - x_k), x_k \in (0, 1), R \in (0, 4). \tag{7}$$
The initial value is deduced based on the user password by character–to-number conversion. The chaotic generated number stream is sampled and quantized in order to obtain only those values included in the GF used for encryption. The non-zero values are excluded for two components of the encryption key and the maximum value of the GF is also forbidden for the exponent value. Special cases regarding the encryption key-vector should be analyzed and solved in the Java implemented application. The public key represents only the non-zero initial state of a random number generator on a GF, so it is not a very long sequence and the number of choices is increased by m. The secret key used for encryption is the generated random number sequence itself and it is much longer than the public key.

**Encryption Principle**

The serial data stream is partitioned into blocks of m bits which are converted binary-to-decimal to obtain an element of the GF (Figure 1). For each value \( a \), the encryption function uses another set of parameters. The same value could be different encoded, based on the memory of the pseudo-random number generator. The public key is also randomly modified to maximize the length of the encryption key. The public key could be changed for every new transmitted frame or on-demand. In this way, the encryption method avoids being a redundant code without any memory. The encryption key length is greater or comparable to the message length.

![Figure 1. Encryption Method Principle](image)

The transmitted data is the output of a discrete information source, in particular a character generator, encoded for example by the ASCII code (American Standard Code for Information Interchange), with or without parity check bit. For values greater than 2, the prime number m is an odd number so the encryption algorithm is applied on some interleaved bits which could originate in different characters. Therefore the same character will be encoded in a different way, depending on the partitioning process, its position into the input stream and the corresponding key symbols. Figure 2 illustrates the partitioning of a binary data stream for encryption on GF (8).

![Figure 2. Partitioning Process for Encryption on GF (8) of the 7-bit ASCII Code.](image)

The data encryption and the pseudo-random key generation are parallel processes. A great dimension of the GF could be used to increase the algorithm robustness. If \( m > 8 \), than the encryption function is applied on groups of bits loaded from more than one character. In this way, the proposed encryption algorithm modifies the probabilistic distribution of the initial character set. Near-uniform distributed characters in the encrypted frame result. The decryption is made similar, using the same chaotic number generator, the same password with the inverse polynomial function given by (2). The encryption system is symmetric because the same password is used to extract the plain text. The coding rate is 1:1, so the transmission data rate is not decreased by encryption.

**Encryption/Decryption Algorithms**

The encryption algorithm has the following steps:

**STEP 1** READ THE PASSWORD

**STEP 2** DEDUCE THE INITIAL STATE OF THE CHAOTIC NUMBER GENERATOR

**STEP 3** READ THE DATA BIT SEQUENCE OF ONE CHARACTER, SYMBOL OR
BLOCK OF THE INFORMATION SOURCE

STEP 4
GENERATE A SET OF THREE COMPONENTS OF THE ENCRYPTION KEY

STEP 5
ENCRYPT THE DATA BLOCK

STEP 6
RUN STEP 2 UNTIL THE END OF THE FILE IS REACHED.

The decryption algorithm is designed in the same way only the key components are changed and the GF function used to process the data. Encryption algorithm implementation in a programming language such as Java or C/C++ has some difficulties given by special unprintable characters resulted from the encoding process (Appendix A). The control character, with ASCII codes less than 0x.20 could give uncontrollable behavior of the receiver, for example the BREAK condition could be obtained so the communication process is stopped and the text is truncated with information loss.

Therefore we have to use a “character stuffing” method. Each time, a control character results from the encryption code, the ESCAPE code (0x.1B) is introduced first, in order to ignore the significance of that character.

In the same way we have to proceed with the code 0x.34 used for the quotation mark character (“”) which is used for string delimitation.

Conclusions

The proposed encryption algorithm modifies the probabilistic distribution of the initial character set. Near-uniform distributed characters in the encrypted frame result.

Two JAVA algorithms are implemented for encryption and decryption of text files, with different classes which execute GF arithmetic operations, character-to-ASCII conversion, the encoding and the decoding functions.

The performances of these algorithms are very good. The processing time is reduced by parallel processing, the robustness is ensured by a very long key, the encrypted file probabilistic distribution is near-uniform so the statistic attacks chances are minimized.

The proposed JAVA text encryption algorithm is more efficient than the well-known DES or IDEA algorithms because the encryption key length is theoretically infinite, longer than the message itself, without any periodicity or redundancy.

References


Appendix A

The JAVA encryption algorithm is given below. Different GF (8) functions are used. This is a selective encryption algorithm for text files.

Criptcharchaos.java

// Copyright (c) 2005-2010 Faculty of Electronics and Telecommunications - Iasi

import java.io.*;

public class Criptcharchaos {
    int joker[]=new int[8];
    int joker3[]=new int[3];
    int joker_e;
    int aaaa[];
int num_key[]=new int[1000];
int num_a[]=new int[1000];
double x[]=new double[1000];
int b[]=new int[1000];
int sir[]=new int[1000];
int bia[]=new int[1000];
int bia3[]=new int[3];
e;
v[]=new int[7];
v7[]=new int[7];
joker_v[]=new int[8];
joker_v7[]=new int[8];
c[]=new int[1000];
joker_c;
Byte aa[]=new Byte[1000];
Byte by[]=new Byte[1000];
Integer in[]=new Integer[1000];
static int lungime_key;
public static void main(String args[])
{
    Criptcharchaos ccc= new Criptcharchaos();
    String nume_fisier="";
    String key="";
    try
    {
        InputStream is= System.in;
        DataInputStream dis1= new DataInputStream(is);
        System.out.println("

  Copyright (c)
2005-2010 Faculty of Electronics and
Telecommunications - Iasi"
);
        System.out.println("*****
Introduceti
numele fisierului text (maxim 500
caractere) pe care doriti sa-l
criptati: ");
        nume_fisier = dis1.readLine();
        if (nume_fisier.substring(nume_fisier.length()-4,
nume_fisier.length()).equals(".txt")
        {
        }
        else
        {
            System.out.println("Fisierul nu e in
format .txt !")
        System.exit(1);
        }
        DataInputStream(is);
        System.out.println("\n Copyright (c)
2005-2010 Faculty of Electronics and
Telecommunications - Iasi\n")
        System.out.println("\t\t\t\t*****")
        System.out.println("\nIntroduceti
numele fisierului text (maxim 500
caractere) pe care doriti sa-l
criptati: ");
        nume_fisier = dis1.readLine();
        if (nume_fisier.substring(nume_fisier.length()-4,
nume_fisier.length()).equals(".txt")
        {
        }
        else
        {
            System.out.println("Fisierul nu e in
format .txt !")
        System.exit(1);
        }
        DataInputStream dis2= new DataInputStream(is);
        String key="";
    }
    catch(IOException e){}
    ccc.criptare(nume_fisier,key);
    System.out.println("Criptarea a avut
loc cu succes !")
    }
    private void criptare(String nume_fis,
String key)
    {
        aaaa=criptcharchaos(FileIO.fileIO(nume_fis),key);
        try
        {
            FileWriter fos = new
FileWriter(nume_fis.substring(0,nume_fis.length()-4)+"_criptat.txt");
            BufferedWriter dos = new
BufferedWriter(fos);
        int i=0;
        while (aaaa[i]!=0)
        {
            dos.write(aaaa[i]);
i++;
        }
            dos.close();
        }
        catch(IOException e)
        {
            System.out.println("Eroare scriere
fiser "+e);
        }
    }
    public int[] criptcharchaos(String a,
String key)
    {
        int k=0;
        int L=a.length();
        int m=3;
        int tp=300;
        double r=3.9;
        int n1min=0,n1max=7;
        int n2min=1,n2max=7;
        int n3min=1,n3max=6;
        byte=string2ascii(key);
        for (int i=0;i<key.length();i++)
        {
            key=Integer.valueOf(key.charAt(i));
        }
        for (int i=0;i<lungime_key;i++)
        {
            num_key[i]=by[i].intValue();
        }
        for (int i=0;i<lungime_key;i++)
        {
            in=binarInteger(num_key[i],7);
            if (key.length()>256)
            {
                b[z]=in[z].intValue();
            }
            else
            {
                b[z]=in[z].intValue();
            }
        }
    }
}
double s0=0;
for (int ii=1;ii<=7*lungime_key;ii++)
{s0=s0+(Math.pow(2,(-ii)))*sir[ii-1];
}
aa=string2ascii(a);
for (int j=0;j<a.length();j++)
num_a[j]=aa[j].intValue();
joker3[0]=joker[0];
joker_e=bin2zec(joker3);
joker_e=power3(joker_e, 1);
joker_v=binar(sgf3(1, pgf3(1, joker_e)), 3);
for(int s=3;s<8;s++)
{joker_v[s]=joker[s];
}
for(int s=0;s<8;s++)
{joker_v7[s]=joker_v[s];
}
joker_c=bin2zec(joker_v7);
for(int i=0;i<7;i++)
{x[0]=s0;
for (int q=1;q<=tp+1;q++)
{x[q]=r*x[q-1]*(1-x[q-1]);
}
int k1=(int)Math.round(x[100]*(n1max-n1min)+n1min);
int k2=(int)Math.round(x[200]*(n2max-n2min)+n2min);
int k3=(int)Math.round(x[300]*(n3max-n3min)+n3min);
bia=binar(num_a[i], 7);
for(int w=0;w<3;w++)
bia3[w]=bia[w];
e=bin2zec(bia3);
e=power3(e,k3);
v=binar(sgf3(k1, pgf3(k2, e)), 3);
for(int s=3;s<7;s++)
{v[s]=bia[s];
}
for(int s=0;s<7;s++)
{v7[s]=v[s];
}
c[i]=bin2zec(v7);
if (c[i]==0)
c[i]=joker_c;
return c;
}
if (x == 0)
    
k = -1;
return k;
}

public int pgf3(int a, int b) {
    int y = 0;
    int pow2to3[][] = {
        {0, 0, 0, 0, 0, 0, 0, 0}, {0, 1, 2, 3, 4, 5, 6, 7}, {0, 2, 4, 6, 3, 1, 7, 5}, {0, 3, 6, 5, 7, 4, 1, 2},
        {0, 4, 3, 7, 6, 2, 5, 1}, {0, 5, 1, 4, 2, 7, 3, 6}, {0, 6, 7, 1, 5, 3, 2, 4}, {0, 7, 5, 2, 1, 6, 4, 3}};
    a = a - 8 * (int) Math.floor(a / 8);
    b = b - 8 * (int) Math.floor(b / 8);
    y = pow2to3[a][b];
    return y;
}

public int power3(int a, int k) {
    int power[][] = {
        {0, 0, 0, 0, 0, 0, 0, 0}, {1, 1, 1, 1, 1, 1, 1, 1}, {1, 2, 4, 3, 6, 7, 5, 1}, {1, 3, 5, 4, 7, 2, 6, 1},
        {1, 4, 6, 5, 2, 3, 7, 1}, {1, 5, 7, 6, 3, 4, 2, 1}, {1, 6, 2, 7, 4, 5, 3, 1}, {1, 7, 3, 2, 5, 6, 4, 1}};
    int y = power[a][k];
    return y;
}

public int sgf3(int a, int b) {
    int sgf[][] = {
        {0, 1, 2, 3, 4, 5, 6, 7}, {1, 0, 3, 2, 5, 4, 7, 6}, {2, 3, 0, 1, 6, 7, 4, 5}, {3, 2, 1, 0, 7, 6, 5, 4},
        {4, 5, 6, 7, 0, 1, 2, 3}, {5, 4, 7, 6, 1, 0, 3, 2}, {6, 7, 4, 5, 2, 3, 0, 1}, {7, 6, 5, 4, 3, 2, 1, 0}};
    int s = sgf[a][b];
    return s;
}

public byte[] string2ascii(String str) {
    byte b[] = str.getBytes();
    byte bye[] = new Byte[1000];
    for (int i = 0; i < str.length(); i++)
        bye[i] = new Byte(b[i]);
    return bye;
}

public Integer[] zec2bin(int a[], int m) {
    int l = a.length;
    Integer vi[] = new Integer[l * m];
    Integer bi[] = new Integer[l * m];
    for (int i = 0; i < l * m; i++)
        vi[i] = binarInteger(a[i], m);
    for (int j = 0; j < m * l; j++)
        bi[(i) * m + j] = vi[j];
    return bi;
}

public String conversie(int a[]) {
    String str = "";
    char ch[] = new char[a.length];
    for (int i = 0; i < a.length; i++)
        ch[i] = (char) a[i];
    str = new String(ch);
    return str;
}