IMPLEMENTATION OF RABBIT ENCRYPTION OF STREAM CIPHER ALGORITHM TO SECURE FILE OF CONFIDENTIAL SECRET IN MANGKUBUMI POLICE TASIKMALAYA

Muhamad Septian[1], Angga Setiyadi, S.Kom., M.Kom[2]
[1,2] Teknik Informatika – Universitas Komputer Indonesia
Jl. Dipatiukur 112-114 Bandung
E-mail : me@mseptian.id[1], angga.setiyadi@email.unikom.ac.id[2]

ABSTRACT

Information leakage is one of the problems that occur within government agencies, especially in the police. One prevention is to encrypt the information. Encryption has been used by Spartan soldiers in the years 46 to 120 AD which aims to send hidden messages when the war takes place. Today, encryption has experienced rapid development, one of which is the cipher rabbit stream. Rabbit stream cipher was first introduced to the public in 2003. Rabbit Stream Cipher uses a secret key 128 bit and 64 bit Initialization Vector (IV) as input for use in an output block consisting of 128 pseudo-random random bits. In this paper will be discussed about rabbit stream cipher, implementation into the program and performance of the rabbit stream cipher itself.

Keywords: *encryption, decryption, information security, rabbit stream cipher, cryptography*.

1. INTRODUCTION

The Indonesian National Police is an Indonesian state security agency that has the task of maintaining public security and order, upholding the law, providing protection, protection, and service to the community. One body that has a high level of confidentiality, therefore the Indonesian National Police is very careful in securing all of its files and data. With the Polres and Polsek in every region of Indonesia, the Police is one of the most sensitive places for data storage.

Encryption method is the process of scrambling data or information so that it cannot be read by other parties or parties who are not entitled to receive, except for those who are entitled to receive and with the existence of this encryption method is expected to prevent interference from people who do not have the right to delete data or information that has been received. This information is divided into two, namely personal information and general information. General information is information that may be known by many people or general. The flow of the process of sending a data or information does not escape interference - interference from unauthorized parties. One technique to maintain data or information confidentiality is to use cryptographic algorithms. One way to overcome this problem is to use data encryption. Expected to secure confidential data so that no confidential information is disclosed to the public.

The discovery of the storage and dissemination of confidential information to the public is not encrypted is a problem experienced by Mangkubumi Tasikmalaya Police today were feared to be a crack leaking of confidential information to the public which can lead to performance Mangkubumi Tasikmalaya City Police. Besides that, the file which is not disseminated into the internal police is also a problem in the Tasikmalaya Sub-district police station.

2. Implementation of Decryption Encryption

2.1 Research methods

The research method used in this paper is literature study which collects references both from papers, articles, internet and books on stream cipher algorithms and programming to make the application.

In this stage the results of the analysis are made system modeling, using the program structure and algorithms, the interface design is made to facilitate the flow system that occurs when programming programs and program analysis. In this paper the author uses java programming language to make it easier to create applications on a desktop platform.

2.2 Analysis of Decryption Encryption Algorithm

According to [1] and [2], Encryption is the process of changing the original data (plaintext) into cipher text (data that cannot be understood) while Decryption is the opposite of encryption, namely the process of returning the form of cipher text back to plaintext so that it can be understood. Encryption and Decryption is done using the specified key. Rabbit Stream Cipher is one of the encryption algorithms with symmetrical keys.

Rabbit is one of the many cipher algorithms introduced in 2003. The Rabbit algorithm uses 128 bits of secret key or often called KEY and 64 bit Initialization Vector (IV) as input to generate 128 bits of randomized output blocks. pseudo-random which

is a combination of bits in the internal status, for each iteration.

The encryption or decryption process is done by XOR the false random block with plaintext or cipher text. The size of the internal status is 513 bits divided into 8 status variables with a length of 32 bits, 8 counters with a length of 32 bits, and 1 bit carry to counter. The eight status variables are updated with 8 non-linear functions.

Rabbit is designed to be faster than commonly used cryptographic algorithms, and uses a 128-bit key to encrypt plaintext with blocks up to 264. So to find out plaintext without having a key, it can only be done with exhaustive search, with 2128 possible keys.

The essence of the rabbit stream cipher algorithm is the next state function that scrambles the input bits into pseudo bit outputs that are stored to be made as a result of decryption. The following is an illustration of the next state function which can be seen in Image 1.

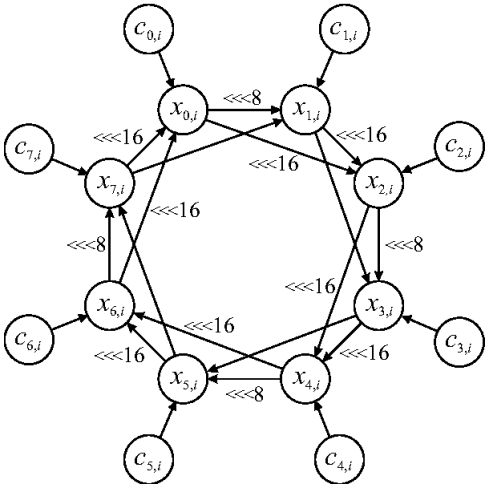


Image 1. *Illustration of images from the Next-state function*

The main key of this Rabbit Stream Cipher Algorithm is the Next-State function. Next State function functions This is based on the g function, which converts two 32-bit inputs into one 32-bit output where the Next-State function can be described as follows:

Table 1. *rabbit stream cipher encryption algorithm*

```
procedure nextState()
Kamus:
t: longint
i: integer
G,C,X : array[1..8] of integer
algoritma:
for i ← 0 to 7 do
    t ← (C[i] & 0xFFFFFFFFL) + (A[i] & 0xFFFFFFFFL) + b;
    b ← (byte) (t >>> 32);
    C[i] ← (int) (t & 0xFFFFFFFF);
endfor
for i ← 0 to 7 do
    t ← (X[i] + C[i]) & 0xFFFFFFFFL;
    G[i] ← (int) ((t *= t) ^ (t >>> 32));
endfor
for i ← 0 to 7 do
    X[i] ← G[i]+rtl(G[(i + 7)% 8], (i+1&1) + 1<<3) + ((i&1) == 0 ? rtl(G[(i+6)%8],16) : G[(i+6)%8]);
endfor
endprocedure
```

2.3 The proposed workflow system

The system to be built aims to facilitate the Mangkubumi Tasikmalaya District Police in securing files. The workings of the system for the process of encrypting and storing files are divided into two, namely the workflow of the system on the computer that aims to encrypt and decrypt as well as to upload into the web server, and the workflow of the system on the smartphone that aims to retrieve the list contained on the web server. The following is a system workflow that aims to encrypt and decrypt files. on the computer to be built is as follows in Figure 2.

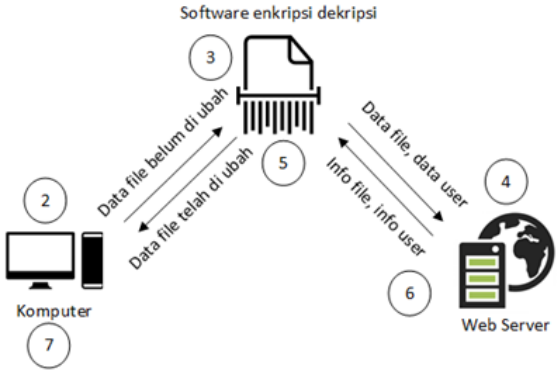


Image 2. *System Overview process of encrypting and storing files on a computer*

Based on Image 2 can be explained the system workflow as follows:

1. Members of the police use a computer that has the encryption file decryption software installed.

2. The computer displays a login form that requires police officers to fill in the NRP field and password.

3. Police File Decryption Encryption Software sends user data to the web server for validation.

4. The Web Server checks the user input received.

5. If the user data entered by police officers is appropriate then the Police File Decryption Encryption Software processes encryption.

6. Encrypted files are then sent to the web service to be saved to the server.

7. The computer also stores the encrypted files from the police file decryption encryption software.

2.4 Implementation of Decryption Encryption Algorithm

The implementation of Rabbit Stream Cipher Algorithm will be used in the main function of the Police Decryption File Encryption application. Rabbit Stream Cipher algorithm uses Java programming language that will be used on desktop and smartphone devices. The following is Rabbit.java which is used as the main function of the Encryption File for the Policing of Mangkubumi Police. Here is the Rabbit.java source code.

Table 2. *Source Code Rabbit.java*

```
import java.util.Arrays;

public class Rabbit {
   private static final int KEYSTREAM_LENGTH = 16;
   private static final int[] A = new int[]
         {
               0x4D34D34D, 0xD34D34D3, 0x34D34D34, 0x4D34D34D, 0xD34D34D3, 0x34D34D34, 0x4D34D34D,
               0xD34D34D3
         };

   private static final int rotl(final int value, final int shift)
   {
      return value << shift | value >>> 32 - shift;
   }

   private final int[] X = new int[8];
   private final int[] C = new int[8];
   private byte b;
   private int keyindex = 0;
   private byte[] keystream = null;

   public Rabbit()
   {
      b = 0;
   }

   public char decryptChar(byte byt)
   {
      byte [] b = {byt};
      //hack to get char from encrypted byte(array of lenght 1)
      return (new String(crypt(b))).charAt(0);
   }

   /**
    * Should be fed an array with a length that is a multiple of 16 for proper
    * key sequencing.
    */
   public byte[] crypt(final byte[] message)
   {
      int index = 0;
      while (index < message.length)
      {
         if (keystream == null || keyindex == KEYSTREAM_LENGTH)
         {
            keystream = keyStream();
            keyindex = 0;
         }
         for (; keyindex < KEYSTREAM_LENGTH && index < message.length; ++keyindex)
         {
            message[index++] ^= keystream[keyindex];
         }
      }
      return message;
   }

   /**
    * returns 16 bytes
    */
   private byte[] keyStream()
   {
      nextState();
      final byte[] s = new byte[16];
         /*
          * unroll
          */
      int x = X[6] ^ X[3] >>> 16 ^ X[1] << 16;
      s[0] = (byte) (x >>> 24);
      s[1] = (byte) (x >> 16);
      s[2] = (byte) (x >> 8);
      s[3] = (byte) x;
      x = X[4] ^ X[1] >>> 16 ^ X[7] << 16;
      s[4] = (byte) (x >>> 24);
      s[5] = (byte) (x >> 16);
      s[6] = (byte) (x >> 8);
      s[7] = (byte) x;
      x = X[2] ^ X[7] >>> 16 ^ X[5] << 16;
      s[8] = (byte) (x >>> 24);
      s[9] = (byte) (x >> 16);
      s[10] = (byte) (x >> 8);
      s[11] = (byte) x;
      x = X[0] ^ X[5] >>> 16 ^ X[3] << 16;
      s[12] = (byte) (x >>> 24);
      s[13] = (byte) (x >> 16);
      s[14] = (byte) (x >> 8);
      s[15] = (byte) x;
      return s;
   }

   private void nextState()
   {
         /*
```

```
            * counter update
            */
        for (int j = 0; j < 8; ++j)
        {
            final long t = (C[j] & 0xFFFFFFFFL) +
(A[j] & 0xFFFFFFFFL) + b;
            b = (byte) (t >>> 32);
            C[j] = (int) (t & 0xFFFFFFFF);
        }
            /*
            * next state function
            */
        final int G[] = new int[8];
        for (int j = 0; j < 8; ++j)
        {
            // TODO: reduce this to use 32 bits only
            long t = X[j] + C[j] & 0xFFFFFFFFL;
            G[j] = (int) ((t *= t) ^ t >>> 32);
        }
            /*
            * unroll
            */
        X[0] = G[0] + rotl(G[7], 16) + rotl(G[6], 16);
        X[1] = G[1] + rotl(G[0], 8) + G[7];
        X[2] = G[2] + rotl(G[1], 16) + rotl(G[0], 16);
        X[3] = G[3] + rotl(G[2], 8) + G[1];
        X[4] = G[4] + rotl(G[3], 16) + rotl(G[2], 16);
        X[5] = G[5] + rotl(G[4], 8) + G[3];
        X[6] = G[6] + rotl(G[5], 16) + rotl(G[4], 16);
        X[7] = G[7] + rotl(G[6], 8) + G[5];
    }

    /**
     * Clears all internal data. You must set the key
again to use this cypher.
     */
    public void reset()
    {
        b = 0;
        keyindex = 0;
        keystream = null;
        Arrays.fill(X, 0);
        Arrays.fill(C, 0);
    }

    /**
     * @param IV An array of 8 bytes
     */
    public void setupIV(final byte[] IV)
    {
        short[] sIV = new short[IV.length >> 1];
        for (int i = 0; i < sIV.length; ++i)
        {
            sIV[i] = (short) ((IV[i << 1] << 8) | IV[(2
<< 1) + 1]);
        }
        setupIV(sIV);
    }

    /**
     * @param iv array of 4 short values
     */
    public void setupIV(final short[] iv)
    {
            /*
            * unroll
            */
        C[0] ^= iv[1] << 16 | iv[0] & 0xFFFF;
        C[1] ^= iv[3] << 16 | iv[1] & 0xFFFF;
        C[2] ^= iv[3] << 16 | iv[2] & 0xFFFF;
        C[3] ^= iv[2] << 16 | iv[0] & 0xFFFF;
        C[4] ^= iv[1] << 16 | iv[0] & 0xFFFF;
        C[5] ^= iv[3] << 16 | iv[1] & 0xFFFF;
        C[6] ^= iv[3] << 16 | iv[2] & 0xFFFF;
        C[7] ^= iv[2] << 16 | iv[0] & 0xFFFF;
        nextState();
        nextState();
        nextState();
        nextState();
    }

    /**
     * @param key An array of 16 bytes
     */
    public void setupKey(final byte[] key)
    {
        short[] sKey = new short[key.length >> 1];
        for (int i = 0; i < sKey.length; ++i)
        {
            sKey[i] = (short) ((key[i << 1] << 8) |
key[(2 << 1) + 1]);
        }
        setupKey(sKey);
    }

    /**
     * @param key An array of 8 short values
     */
    public void setupKey(final short[] key)
    {
            /*
            * unroll
            */
        X[0] = key[1] << 16 | key[0] & 0xFFFF;
        X[1] = key[6] << 16 | key[5] & 0xFFFF;
        X[2] = key[3] << 16 | key[2] & 0xFFFF;
        X[3] = key[0] << 16 | key[7] & 0xFFFF;
        X[4] = key[5] << 16 | key[4] & 0xFFFF;
        X[5] = key[2] << 16 | key[1] & 0xFFFF;
        X[6] = key[7] << 16 | key[6] & 0xFFFF;
        X[7] = key[4] << 16 | key[3] & 0xFFFF;
            /*
            * unroll
            */
        C[0] = key[4] << 16 | key[5] & 0xFFFF;
        C[1] = key[1] << 16 | key[2] & 0xFFFF;
        C[2] = key[6] << 16 | key[7] & 0xFFFF;
        C[3] = key[3] << 16 | key[4] & 0xFFFF;
        C[4] = key[0] << 16 | key[1] & 0xFFFF;
        C[5] = key[5] << 16 | key[6] & 0xFFFF;
```

```
        C[6] = key[2] << 16 | key[3] & 0xFFFF;
        C[7] = key[7] << 16 | key[0] & 0xFFFF;
        nextState();
        nextState();
        nextState();
        nextState();
            /*
             * unroll
             */
        C[0] ^= X[4];
        C[1] ^= X[5];
        C[2] ^= X[6];
        C[3] ^= X[7];
        C[4] ^= X[0];
        C[5] ^= X[1];
        C[6] ^= X[2];
        C[7] ^= X[3];
    }
}
```

2.5 Interface Implementation

Interface implementation contains exposure from each interface that has been designed in the analysis phase. The implementation of the interface in the development of the encryption system for decrypting the police files of the Mangkubumi police department of Tasikmalaya consists of a desktop interface and a smartphone interface. The implementation of the interface on the desktop platform can be seen as follows. Implementation of desktop platform interfaces can be seen in table 4.3 as follows.

Table 3. *Interface implementation*

Menu	***Description***	***Process***	***File***
Home page	*File Encryption Application Start Page*	*Display Main Page*	MainJFrame.form
Encryption page	*The page used to encrypt files*	*File Encryption*	EncryptJFrame.form
Encrypted Successful Page	*The page used to display a notification that encryption was successful*	*File Encryption Notification*	EncryptSuccess.form
Decryption page	*The page used to decrypt files*	*Decryption of Files*	DecryptJFrame.form
Successful Decryption page	*The page used to display notifications that successfully decrypt*	*File Decryption Notification*	DecryptSuccess.form

2.6 Program Implementation

Program implementation contains the results of the experiment running the program that has been made. By implementing the algorithm and interface, a program is created to test the encryption and decryption of the file. The following are steps for encrypting and decrypting files. For the first step, the user is required to login to use the decryption encryption application, here is the login step.

Table 4. *Experiment Login*

NRP	12345678
Password	12345678



Image 3. *Login page*

The user will then display a home page containing two buttons. Each key is useful for encryption and decryption processes. If the user chooses encryption then the program will display the encryption page as shown in Image 4. Then the user can enter the input field such as select the file to be encrypted, select the result folder and password or key that is used for viewing.

Table 5. *Encryption Experiment*

Select File	Surat tugas.docx
Output folder	Documents
Password	M4ngkubum1



Image 4. *Encryption Page Display*

After successfully encrypting the user will be displayed successfully encryption page which can be seen in Image 5.

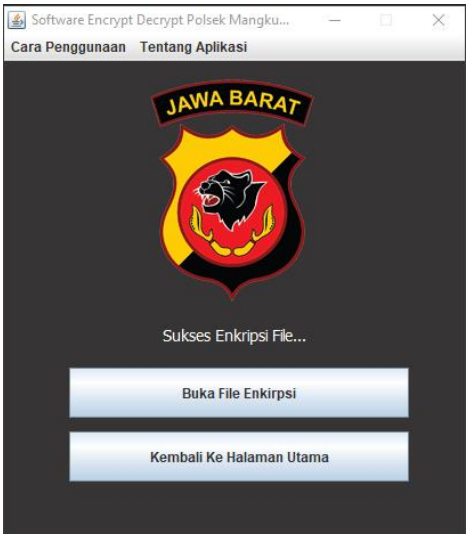


Image 5. *Display of Encrypted Successful Pages*

3. CLOSING

In the case of security in the police file decryption encryption program, the Rabbit algorithm is very good to implement, because until now there has been no attack on the Rabbit algorithm that is faster than exhaustive search. While the exhaustive search for the Rabbit algorithm is a search for 2128 possible keys, because the length of the key used is 128 bits. Rabbit algorithm can be used to encrypt plaintext with size up to 264 bytes. In terms of speed and output size, Rabbit algorithm can be said to be faster than the flow cipher algorithm that is commonly used and more efficient for storage because input and output produce the same size.

The suggestion that the writer can convey is that the rabbit stream cipher algorithm is better combined with compression algorithms so that the size of encryption is smaller. And you should be able to combine it with other encryption algorithms.

BIBLIOGRAPHY

[1] Ariyus, Doni, Pengantar Ilmu Kriptografi Teori, Analisis dan Implementasi, Penerbit Andi, 2008.
[2] Munir, Rinaldi., Kriptografi, Informatika, 2007.
[3] Boesgaard, Martin, et al, The Rabbit Stream Cipher – Design and Security Analysis, Cryptico, 2003.
[4] Boesgard, Martin, Mette Vesterager, Thomas Pedersen, Jesper Christiansen, and Ove Scavenius. Rabbit: A New High-Performance Stream Cipher. Cryptico A/S. 2003. http://www.cryptico.com/. 8 Agustus 2018 pukul 15:00.
[5] Suryana, Taryana. "Pengelolaan Dokumen sebagai Sarana Komunikasi Internal UNIKOM.". Jurnal Komputer dan Informatika (KOMPUTA), 2012.
[6] Paar, Prof. Dr.-Ing. Christof., Pelzl, Dr.-Ing. Jan., Understanding Cryptography: A Textbook for Students and Practitioners, Springer-Verlag, Berlin, 2010.
[7] Boesgaard, Martin, et al, The Rabbit Stream Cipher – Design and Security Analysis, Cryptico, 2003.
[8] Binanto, Iwan. 2005. Konsep Dasar Program. Jakarta: PT. Elex Media. Komputindo. Indrajani. 2007
[9] Jogiyanto. 2005. Analisis dan Desain Sistem Informasi. Yogyakarta: Penerbit Andi. Juju, Dominikus. 2006
[10] Wahana Komputer, The Best 40 Java Aplications, Penerbit PT. Gramedia Pustaka Utama, Jakarta, 2010.