Partial Image Encryption using RC4 Stream Cipher Approach and Embedded in an Image

Renas R. Asaad, Saman M. Abdulrahman and Ahmad A. Hani

Department of Computer Science and I.T, College of Computer Science, Nawroz University, Duhok, Kurdistan Region – F.R. Iraq

ABSTRACT

Cryptography is a science concerned with securing data transmission. By applying cryptographic operations to data, readable information is converted into unreadable data. Currently, security is of great importance. Data transferred over the internet must have some form of encryption. Different forms of multimedia contents are moved over the internet. This paper focuses on image encryption. It tries to amend the multiple selective region image cryptography techniques using a RC4 stream cipher. This approach is derived from the standard RC4 algorithm. The proposed method can highly improve security for images transferred over the internet.

KEY WORDS: RC4 Cryptography, Stream Cipher, Image Cryptography, Partial Image Encryption, Symmetric Cipher.

1. INTRODUCTION

This paper is concerned with an encryption software that encrypts parts of an image and embeds them into other images for extra security. Bitmap extension images are used in the process since they are large in size and it is easy to hide data in them. The main advantage of this method is that an attacker must first find the hidden image data inside the transferred image and then decrypt it (Dawson and Nielsen, 1996).

In this article, the following encryption techniques are used.

1. RC4
2. Alternating step generator
3. Shrinking generator.

2. METHODOLOGY OF RC4 ALGORITHM

Usually, image encryption begins with taking an image and applying encryption algorithms to it, producing the encrypted image. However, regarding selective image encryption, first parts or regions of the selected image are chosen so that the encryption algorithm is applied to it, i.e., the algorithm only works on parts of input and not the whole image. Afterward, the selected parts are encrypted while the rest of the image is retained.

Regarding decryption, the encrypted image is taken and the decryption algorithm is applied which results in the original image (Fluhrer et al., 2001).

2.1. Encryption Keys

Encrypting data usually involve taking characters of data and comparing them against the encryption key. Keys are responsible for determining which characters of data are encoded into which characters. Modern encryption methods rely on complex and large keys for higher security. Encryption keys fall into two categories, private and public keys.

2.2. Private Key Encryption

Private or symmetrical key encryption technology is where both the sender and receiver have the same key, which is used to encrypt and decrypt all messages. Hence, it is difficult to establish the communication for the first time because the key must be transmitted safely for other messages to be secure. Thus, public keys are used (Fluhrer et al.,2001).

2.3. Public Key Encryption

Public or asymmetrical key encryption also known as a Diffie-Hellman algorithm involves using two keys to

Academic Journal of Nawroz University (AJNU)
Volume 6, No 3(2017), 6 pages
Received 01 May 2017; Accepted 01 July 2017
Regular research paper: Published 18 July 2017
Corresponding author's e-mail: renas.rekany@nawroz.edu.krd

Original article | doi: 10.25007/ajnu.v6n3a76

encrypt and decrypt the data, a public and a private key. The sender encrypts the messages using the public key, which are then decrypted on the receiver's end using the receivers private key. This is a one-way communication. The same principal can be used by the receiver to return messages to the original sender. Digital certificates are used in case the sender does not have a public key, which are responsible for verifying the identity of the sender. Fig. 1 shows public key - encrypted communication between two units, User X and User Y (Fluhrer et al., 2001; Knudsen and Eschen, 1998).

2.4. Stream Cipher Structure

Stream ciphers are design to encrypt one bit or units larger than a byte at a time. However, they are used to encipher one byte of data at a time. Fig. 2 is a representative diagram of stream cipher structure. In the structure, a key if used as input to a pseudorandom bit generator, which results in a stream of 8-bits that are apparently random. The resulted pseudorandom stream is produced by an algorithm which is unpredictable without the key input. The resultant key stream is combined with the plaintext one byte at a time using the Exclusive-OR operation. For example, if the next byte generated by the generator is 01101100 and the next plaintext byte is 11001100, then the resulting ciphertext byte is:

11001100 plaintext
01101100 key stream
⊕
10100000 cipher text

Decryption requires the use of the same pseudorandom sequence:

10100000 cipher text
01101100 key stream
⊕
11001100 plaintext

There are imperative design considerations which must be taken to design a stream cipher:

1. Since the pseudorandom bit stream generator relies on a function that produces a finite stream of numbers, it ultimately repeats the bit stream. Thus, the generator must have a set of results. The larger the repeat period, the harder it is for cryptanalysis.
2. The generator should produce bitstreams that resemble a true random stream. There should be an approximate equal number of repetitions of bits and bits reams, i.e., the number of 0's and 1's should be approximately equal in a stream. The more randomized a keystream, the more difficult it is for cryptanalysis.
3. The key input to the pseudorandom stream generator should be long enough to protect against brute force attacks. Since the generator solely relies on the key input to produce a stream of bits. Key size is recommended at 128 bit currently.

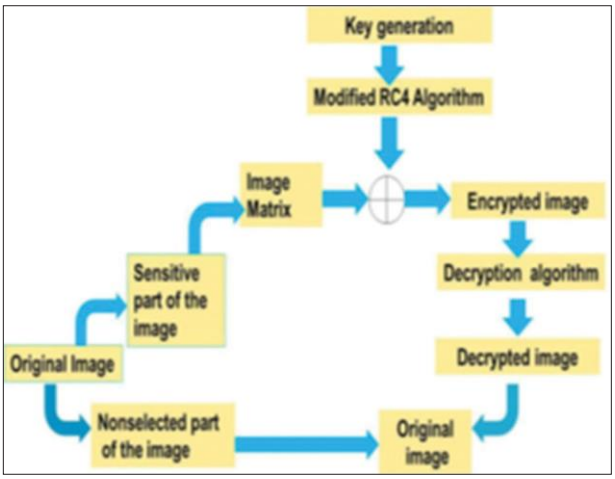

Fig 1: Methodology of RC4 algorithm.

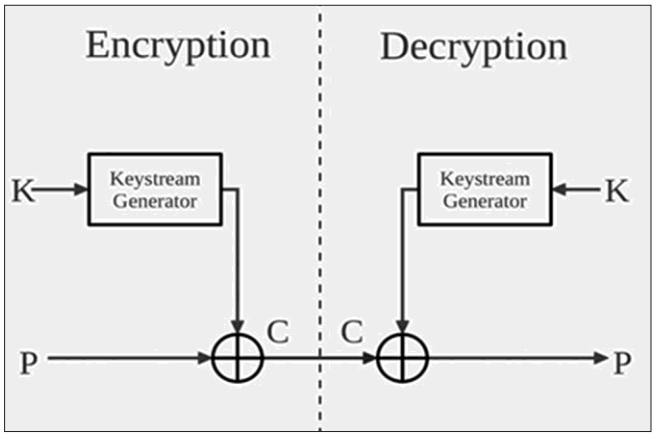

Fig 2: Stream cipher diagram.

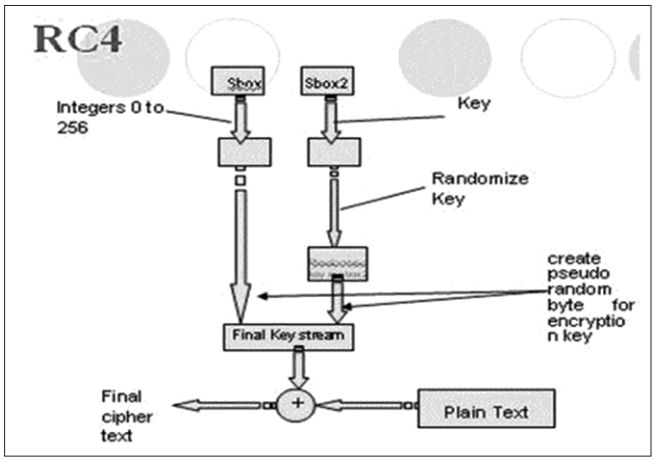

Fig 3: RC4 stream generator.

A stream cipher with a well-designed pseudorandom number generator can be as secure as a block cipher with a similar key length. Moreover, stream ciphers are usually faster and much shorter regarding code compared to block ciphers. The RC4 algorithm can be as short as a few lines of code. Table 1 shows the comparison of the execution time of RC4 against three popular symmetric

TABLE 1
Speed comparisons of symmetric ciphers on a Pentium II

Cipher	Key length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	Variable	0.9
RC4	Variable	45

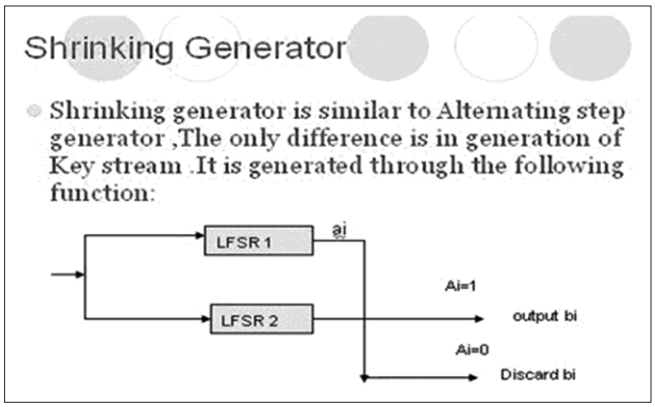

Fig 4: Shrinking generator.

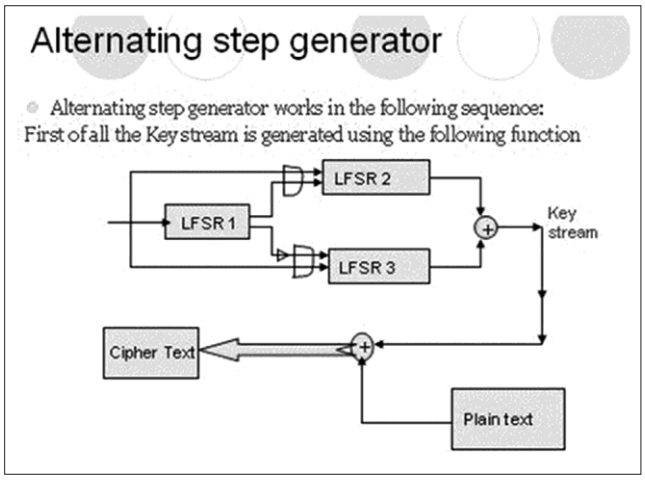

Fig 5: Alternating step generator.

block ciphers. The disadvantage of stream ciphers is that if two plaintexts are encrypted with the same key, then cryptanalysis is often easy. However, it is possible to reuse keys in block ciphers. In a stream cipher, if two ciphertexts enciphered with the same key are XORed together, then the result is a XOR of the original plaintexts. Hence, if the ciphertexts were originally text information such as strings, numbers, credit card information, or other known property streams, then cryptanalysis can be easily successful. Stream ciphers are better suited for application that encipher a stream of data over a communication channel or a web/browser link. However, block ciphers can more appropriate with applications that encrypt a block of data; these can be e-mail, file transfer, and database applications. That being said, both ciphers can be virtually used in any type of application (Knudsen and Eschen, 1998).

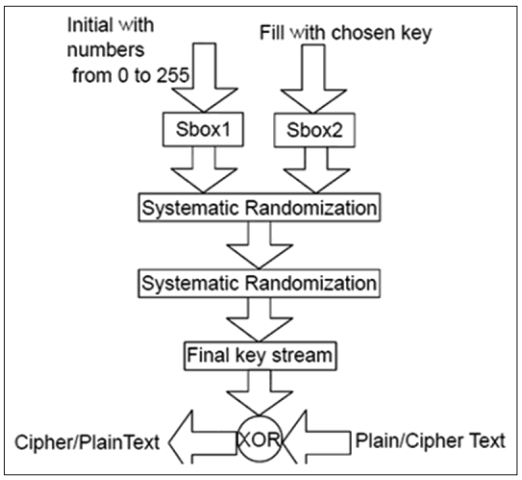

Fig 6: Encryption RC$.

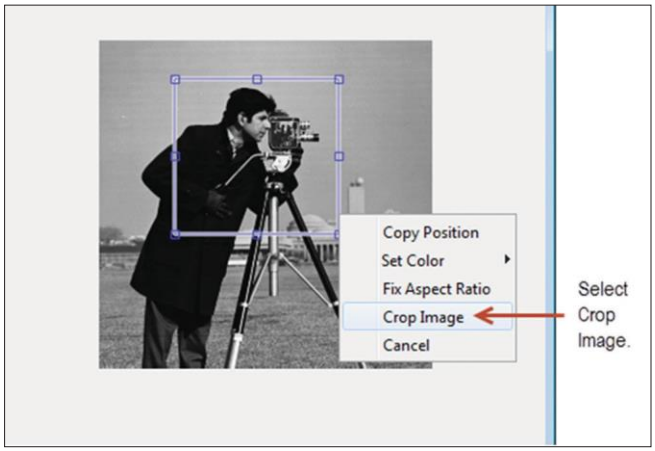

Fig 7: Cropping partial image.

3. THE RC4 ALGORITHM

In 1987, Ron Rivest design RC4 for RSA security. RC4 is a stream cipher with byte-oriented operations and a flexible key size. The algorithm is mainly based on random permutation. Studies show that the algorithm has an overwhelming period of probably >10100. An output byte requires eight to sixteen machine operations. RSA held the RC4 algorithm as a trade secret until September 1994, when the algorithm was anonymously published on the internet on Cypherpunks anonymous remailers list. RC4 is quite simple and straightforward to explain. A key of variable length of 1 to 256 bytes can be used to initialize a 256-byte state vector S, with elements S[0], S[1],…, S[255]. S always contains a permutation of all 8-bit numbers from 0 to 255. A byte k is produced from S by selecting one of the 255 entries in a systematic fashion. When each K value is generated, the entries in S are permutated once again (Kumar, 1997; Mantin and Shamir, 2001).

3.1. Steps of RC4 Stream Cipher

The RC4 algorithm steps of encryption:

1. Provide the data to be encrypted and select the encryption key.

2. Generate two string arrays.
3. Array one should be instantiated with numbers 0-255.
4. The second array should be instantiated with the encryption key.
5. Array one should be randomized using the key array.
6. Randomize array one with regards to itself to produce the final key stream.
7. The final key stream is XORed with the data to produce the ciphered text.

Summarization of RC4 algorithm features:

1. Symmetric stream cipher
2. Various key size
3. Quickly in software
4. For secure communications can be used as in the encryption of traffic to secure websites using the SSL protocol.

3.2. Initialization of S

First, S is initialized with values from 0 to 255 in ascending order, S[0], S[1],…, S[255]. Furthermore, vector T is created temporarily. The vector K (key vector) should also be 256 bytes, if so K is transferred to T else if key length equals keylen of bytes, then K is transferred to T and repeated as required to fill vector T. The operation described can be summarized as blow:

```
//Initialization
for m = 0 to 255 do
S[m] = m;
t[m] = k[m mod keylen];
```

Next, S is produced from the initial permutation of T. This operation starts with S[0] through to S[255], and for each S[m], swapping S[m] with another byte in S according to a scheme dictated by T[m]:

```
//Initial Permutation of S
n = 0;
for m = 0 to 255 do
n = (n + S[m] + r[m]) mod 256;
Swap (S[m], S[n]);
```

Permutation is the only operation effecting S since S is only swapped. S retains all its values 0-255 out of order.

3.3. Stream Generation

After producing the S vector for the first time, the input key is no longer used. The stream generation afterward relies on going through S[0] to s[255], swapping each S[i] with another byte of S dictated by the current configuration scheme of S. The process restarts at S[0] after reaching the end of S s[255] (Mister and Tavares, 1998).

```
//Stream Generation
m, n = 0;
while (true)
m = (m + 1) mod 256;
n = (n + S[m]) mod 256;
Swap (S[m], S[n]);
t = (S[m] + S[n]) mod 256;
k = S[t];
```

The Fig. 3 is showing the encryption process involves XORing the value of K with the next byte of plaintext. While the decryption involves XORing the value of K with the next byte of ciphertext.

3.4. Strength of RC4

There has been great research on the weaknesses of RC4 and how it can be attacked [e.g., [KNUD98], [MIST98], [FLUH00], and [MANT01]). [FLUH01] reported a tangible weakness in it's report. It is demonstrated in the paper that the WEP 802.11 security protocol in vulnerable to a particular attack approach. However, the issue is not in the RC4 algorithm itself but in the method in which keys are produces to be used as input for the algorithm. This approach has not been successful against other applications that use the RC4 algorithm, and thus the weakness in the WEP protocol can be rectified by modifying the method used to generate keys for input. This issue indicated that the difficulty relies in designing systems that include cryptographic algorithms and protocols that make efficient use of these algorithms (Mister and Tavares, 1998; Rescorla, 2001).

Fig. 4 explain the process of random byte streams that are used by the shrinking generator and the alternating step generator, which are most suitable for hardware implementation. An LFSR R1 controller is used to govern the second LFSR R2 controller output. The steps below are repeated to produce a desired key length:

1. R1 and R2 are clocked.
2. R2 output bit is used to form part of the key if R1 output bit is 1.
3. R2 output bit is discarded and clocked again if the R1 output bit is 0.

Finally, the key stream and the plain text stream are XORed to produce the final cipher text. As shown in Fig. 5. Alternative step generator process:

1. Register 1 is clocked.
2. When R1 output is 1, then R2 is clocked, and R3's previous bit is repeated.
3. When R1 output is 0, then R3 is clocked, and R2's previous bit is repeated.

Then, key length equals the plain text stream length, and the key is XORed with the plain text to produce the ciphered text (Rescorla, 2001; Pieprzyk and Pieprzyk, 2003).

The result is final cipher text: Values are provided to the edit boxes initial state and key in a "1,1,0,0,1" format. Hence, LFSR 1, LFSR 2, and LFSR 3 key and function should have equal parameter length. Primarily, user input is taken in the form of a CString object for each polynomial and key. If the shrinking generator is enabled, then the third LFSR input is disabled using tahir1.Setreadonly (TRUE); as follows:

```
if(SG.GetCheck()==TRUE)
{
ASG=FALSE;
ASG1.SetCheck(BST_UNCHECKED);
tahir1.SetReadOnly(TRUE);
```

```
tahir2.SetReadOnly(TRUE);
}
else
{
ASG=TRUE;
ASG1.SetCheck(BST_CHECKED);
tahir1.SetReadOnly(FALSE);
tahir2.SetReadOnly(FALSE);
}
if(ASG==TRUE)
EncryptAsg();
else
EncryptSg();
UpdateData(FALSE)
```

3.5. Encryption and Decryption

Regarding encryption, first the header in extracted and stored in the file. Afterward, the remaining portion of the string is transferred to the encryption or decryption function. The encryption returns the cipher text from the provided input and stores it in the file attached to the 70 bytes previously stored (Asaad, Renas R., Saman M. Abdulrahman, and Ahmed A. Hani,2017). As shown in Fig. 6.

3.6. Image Embedding

To secure the data inside an image, the enciphered data are embedded in a bitmap image. There should not be any distortion of data if the image size is efficiently large. The image header is saved first in the encryption process because if the header is interrupted, then the image will be invisible (Allam, 2005; Glover and Grant, 2004).

```
UpdateData(true);
CString ct="",tahir="";
char *temp2;
char *temp1;
char  *temp3;
int control=800;
int fsize=0;
CFile myfile;
if(m_pt=="" && FileNamex!="")
{
CFile mfile;
mfile.Open(FileNamex,CFile:: modeRead);
int fs=mfile.GetLength();
temp3=new char[fs];
mfile.Read(temp3,fs);
mfile.Close();
for(int d=0;d<fs;d++)
m_pt+=temp3[d];
}
if(m_pt!="" && m_k !="")
{
int ctl,f=0,s=0;
if(FileName!="")
{
myfile.Open(FileName,CFile:: modeRead);
fsize=myfile.GetLength();
temp1=new char[fsize];
myfile.Read(temp1,fsize);
myfile.Close();
}
else
AfxMessageBox("NO IMAGE ");
ct=Encrypt(m_pt,m_k);
ctl=ct.GetLength();
temp2=new char[fsize+ctl];
for(int i =0;i<401;i++)
{
temp2[i]=temp1[i];
}
tahir.Format("%d",ctl);
temp2[401]=tahir.GetLength();
f=(fsize-1000)/ctl;
for(int a=0;a<tahir.GetLength();a++)
temp2[a+402]=tahir[a];
for(int k=0;k<ctl&&control<fsize;k++)
{
for(s=0;s<f;s++)
{
control++;
temp2[control]=temp1[control];
}
control++;
temp2[control]=ct[k];
}
for(int p=control;p<fsize;p++)
temp2[p]=temp1[p];
SaveName=myfile.GetFileName();
SaveName="Embeded.bmp"+SaveName;
myfile.Open(SaveName,CFile:: modeCreate | CFile::
modeWrite);
myfile.Write(temp2,fsize);
myfile.Close();
AfxMessageBox("Embeding Completed");
}
else {
AfxMessageBox("Enter Plain text or File Name and Key");
}
"Retrieving the Data back "
UpdateData(true);
CString ct="",tahir="",strz="",str;
int ctl;
CFile myfile;
myfile.Open(FileName,CFile:: modeRead);
int fsize=myfile.GetLength();
int control=800,f=0;
char *temp1=new char[fsize+1];
char *temp2=new char[fsize+1];
myfile.Read(temp1,fsize);
myfile.Close();
int t,r, q,s=0;
t=temp1[401];
for(q=0;q<t;q++)
tahir+=temp1[402+q];
r=8-t;
for(int z=0;z<r;z++)
```

```
strz+='0';
str=strz+tahir;
ctl=atoi(str.GetBuffer(str.GetLength()));
tahir.Format("%d",ctl);
ctl=atoi(tahir.GetBuffer(tahir.GetLength()));
f=(fsize-1000)/ctl;
for(int k=0;k<ctl&& control<fsize;k++)
{
for(s=0;s<f;s++)
{
control++;
}
control++;
temp2[k]=temp1[control];
}
for(int a=0;a<ctl;a++)
ct+=temp2[a];
m_pt=Encrypt(ct,m_k);
UpdateData(FALSE);
AfxMessageBox(" Retrieval Completed ");
```

4. RESULT AND CONCLUSION

In conclusion, this paper deals with the idea of partial image encryption. Based on the RC4 algorithm, an enhanced algorithm is introduced to further secure the algorithm against attacks. The enhanced algorithm is faster and more attack proves when compared to the standard RC4 algorithm when applied to selective image encryption.

This is achieved after detailed analysis of the enhanced RC4 algorithm. The analysis result confirms that the speed and security of the enhanced algorithm are greater than DES, Triple DES, and RC2 algorithms. As the time required to crack an image encrypted using this algorithm is significantly greater by hardware, let alone software attacks may not be able to break it. The Fig. 7 is the result of the selecting partial image to encrypt it.

REFERENCES

Allam, M. (2005). Data Encryption Performance Based on Blowfish. In: *47th International Symposium ELMAR-2005 Focused on Multimedia Systems and Applications*. Zadar, Croatia, 08-10 June 2005. p131-134.

Dawson, E & Nielsen, L. (1996). Automated cryptanalysis of XOR plaintext strings. *Cryptologia*, 20, 4.

Fluhrer, S., Mantin, I & Shamir, A. (2001). Weakness in the key scheduling algorithm of RC4. In: *Proceedings, Workshop in Selected Areas of Cryptography*.

Glover, P & Grant, M. (2004). *Digital Communications*. 2nd ed. Harlow: Person Education.

Knudsen, L & Eschen, D.J. (1998). Analysis method for alleged RC4. In: *Proceedings, ASIACRYPT' 98*.

Kumar, I. (1997). *Cryptology*. Laguna Hills, CA: Aegean Park Press.

Mantin, I & Shamir, A. (2001). A practical attack on broadcast RC4. In: *Proceedings, Fast Software Encryption*.

Mister, S & Tavares, S. (1998). Cryptanalysis of RC4-like ciphers. In: *Proceedings, Workshop in Selected Areas of Cryptography, SAC' 98*.

Pieprzyk, J & Pieprzyk, J. (2003). *Fundamentals of Computer Security*. Cambridge, UK: Springer.

Rescorla, E. (2001). *SSL and TLS: Designing and Building Secure Systems*. Reading, MA: Addison-Wesley.

Robshaw, M. (1995). *Stream Ciphers*. RSA Laboratories Technical Report TR-701, July 1995.

Asaad, Renas R., Saman M. Abdulrahman, and Ahmed A. Hani. "Advanced Encryption Standard Enhancement with Output Feedback Block Mode Operation." Academic Journal of Nawroz University 6.3 (2017): 1-10