

An approach in improving transposition cipher system

Massoud Sokouti ¹, Babak Sokouti ² and Saeid Pashazadeh ¹

¹ Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran

² Faculty of Engineering, Islamic Azad University -Tabriz Branch, Tabriz, Iran
m_sokouti@yahoo.com

Abstract: Transposition ciphers are stronger than simple substitution ciphers. However, if the key is short and the message is long, then various cryptanalysis techniques can be applied to break such ciphers. By adding 8 bits (one byte) for each byte using a function and another mathematical function to position the bits in a binary tree and using its in-order tour, this cipher can be made protected. Using an in-order tour of binary tree can diffuse the eight bits (includes 7 bits produced by the function and 1 random bit) and eight bits of the plaintext. This can highly protect the cipher. However, if the key management processes are not secured the strongest ciphers can easily be broken.

Keywords: Cryptography, binary tree, cipher protection, transposition cipher.

Introduction

With the growth of Internet, hackers and spies make big issues for army forces, organizations and companies. Transposition ciphers are still the most important kernel techniques in the construction of modern symmetric encryption algorithms. We will clearly see combinations of substitution and transposition ciphers in two important modern symmetric encryption algorithms: DES and AES (Mao Hewlett W-Packard, 2003). DES is not very secure because of the limitation of the space of the key is 56 bits. On the other hand AES is a new cipher. The benefit of these 2 ciphers is that they have two factors of cryptology and security, diffusion and confusion (Schinier, 1996). In part 3 we will completely clarify these two factors. In this paper we present a new algorithm that mostly uses transposition cipher, diffusion and modification of block cipher.

Transposition ciphers

Transposition ciphers are an important family of classical ciphers, in additional substitution ciphers, which are widely used in the constructions of modern block ciphers (Mao Hewlett W-Packard, 2003).

In a transposition cipher the

plaintext remains the same, but the order of characters is shuffled around. In a simple columnar transposition cipher, the plaintext is written horizontally onto a piece of graph paper of fixed width and the cipher text is read off vertically. Decryption is a matter of writing the cipher text vertically onto a piece of graph paper of identical width and then reading the plaintext off horizontally (Schinier, 1996).

Cryptanalysis of these ciphers is discussed in (Sinkov, 1966; Gaines, 1956). Since the letters of the cipher text are the same as those of the plaintext, a frequency analysis on the cipher text would reveal that each letter has approximately the same likelihood as in English. This gives a better clue to a cryptanalyst, who can apply a variety of techniques to determine the right ordering of the letters to obtain the plaintext (Schinier, 1996)

Putting the cipher text through a second transposition, cipher greatly enhances security. There are even more complicated transposition ciphers, but computers can break almost all of them. The German ADFGVX cipher, used during World War I, is a transposition cipher combined with a simple substitution. It was a very complex algorithm for its day but was broken by Georges Painvin, a French cryptanalyst (Kahn, 1967; Schinier, 1996).

We can give an example for transposition cipher. In our example the key is a small number for example 5. In order to encrypt a message using this key, we write the key in rows of 5 letters and encrypt by writing the letters of the first column first, then the second column, etc. If the length of the plain text is not multiple of 5 then we add the appropriate number of "z"s at the end before we encrypt (Table 1, 2).

Transpositions of this type are easy to break. Since the key must be a divisor of the cryptogram length, an attacker has only to count the length of the cryptogram and try each divisor in turn (Piper & Murphy, 2002).

Table 1
Plain text:
ifnighclocksaredesiredstartthegreenplan
Key: 5

i	f	n	i	g
h	t	c	l	o
c	k	s	a	r
e	d	e	s	i
r	e	d	s	t
a	r	t	t	h
e	g	r	e	e
n	p	l	a	n

Cipher text:
ihceraenftkdergpnscsedtrilassteagorithen

Table 2
Plain text:
ifnighclocksaredesiredstartthegreenplan
Key: 3

i	f	n
i	g	h
t	c	l
o	c	k
s	a	r
e	d	e
s	i	r
e	d	s
t	a	r
t	t	h
e	g	r
e	e	n
p	l	a
n	z	z

Cipher text:
iitosesetteepnfaccadidataelznhlkrersrhznaz

Rotor machines

In the 1920s, various mechanical encryption devices were invented to automate the process of encryption. Most were based on the concept of a rotor, a mechanical wheel wired to perform a general substitution.

Go!

Keyword

Go!

A rotor machine has a keyboard and a series of rotors, and implements a version of the Vigenère cipher. Each rotor is an arbitrary permutation of the alphabet, has 26 positions, and performs a simple substitution. For example, a rotor might be wired to substitute "F" for "A," "U" for "B," "L" for "C," and so on. And the output pins of one rotor are connected to the input pins of the next. For an example, in a 4-rotor machine the first rotor might substitute "F" for "A," the second might substitute "Y" for "F," the third might substitute "E" for "Y," and the fourth might substitute "C" for "E"; "C" would be the output cipher text. Then some of the rotors shift, so next time the substitutions will be different.

It is the combination of several rotors and the gears moving them that make the machine secure. Because the rotors all move at different rates, the period for an n -rotor machine is 26^n . Some rotor machines can also have a different number of positions on each rotor, further frustrating cryptanalysis.

The best-known rotor device is the Enigma. The Enigma was used by the Germans during World War II. The idea was invented by Arthur Scherbius and Arvid Gerhard Damm in Europe. It was patented in the United States by Arthur Scherbius (Schinier, 1996; Scherbius, 1928). The Germans beefed up the basic design considerably for wartime use.

The German Enigma had three rotors, chosen from a set of five, a plug board that slightly permuted the plaintext, and a reflecting rotor that caused each rotor to operate on each plaintext letter twice. As complicated as the Enigma was, it was broken during World War II. First, a team of Polish cryptographers broke the German Enigma and explained their attack to the British. The Germans modified their Enigma as the war progressed, and the British continued to cryptanalyze the new versions. Explanations of how rotor ciphers work and how they were broken have already been reported (Kahn, 1967; Barker, 1977; Deavours & Kruh, 1985; Diffie & Hellman, 1979; Deavours, 1980; Konheim, 1981; Rivest, 1981; Welchman, 1982; Hagelin, 1994). Two fascinating accounts of how the Enigma was broken were provided by Hodges (1983) and Kahn (1991). For the enigma simulator vide Carlson, Andy, Enigma simulator, http://homepages.tesco.net/~andycarlson/enigma/eigma_i.html

Confusion and diffusion

The two basic techniques for obscuring the redundancies in a plaintext message are confusion and diffusion (Shannon, 1949). Confusion obscures the

relationship between the plaintext and the cipher text. This frustrates attempts to study the cipher text looking for redundancies and statistical patterns. The easiest way to do this is through substitution. A simple substitution cipher, like the Caesar Cipher, is one in which every identical letter of plaintext is substituted for a single letter of cipher text. Modern substitution ciphers are more complex: A long block of plaintext is substituted for a different block of cipher text, and the mechanics of the substitution change with each bit in the plaintext or key. This type of substitution is not necessarily enough; the German Enigma is a complex substitution algorithm that was broken during World War II.

Diffusion dissipates the redundancy of the plaintext by spreading it out over the cipher text. A cryptanalyst looking for those redundancies will have a harder time finding them. The simplest way to cause diffusion is through transposition (also called permutation). A simple transposition cipher, like columnar transposition, simply rearranges the letters of the plaintext. Modern ciphers do this type of permutation, but they also employ other forms of diffusion that can diffuse parts of the message throughout the entire message. Stream ciphers rely on confusion alone, although some feedback schemes add diffusion. Block algorithms use both confusion and diffusion. As a general rule, diffusion alone is easily cracked (although double transposition ciphers hold up better than many other pencil-and-paper systems) (Schinier, 1996).

Block cipher

In a block cipher, the bit-string is divided into blocks of a given size and the encryption algorithm acts on that block to produce a cryptogram block that, for most symmetric ciphers, has the same size.

Block ciphers have many applications. They can be used to provide confidentiality, data integrity, or user authentication, and even be used to provide the key stream generator for stream ciphers. With stream ciphers, it is very difficult to give a precise assessment of their security. Clearly the key size provides an upper bound of an algorithm's cryptographic strength. However with the Simple Substitution Ciphers, having a large number of keys is no guarantee of strength. A symmetric algorithm is said to be well designed if an exhaustive key search is the simplest form of attack. Of course, an algorithm can be well designed but, if the number of keys is too small, also be easy to break.

Designing strong encryption algorithms is a specialized skill. Nevertheless there are a few obvious properties that a strong block cipher should possess and which are easy to explain. If an attacker has obtained a known plaintext pair for an unknown key, then that should not enable them deduce easily the cipher text corresponding to any other plaintext block. For example, an algorithm in which changing the plaintext block in a known way produces a predictable change in the cipher text, would not have this property. This is just one reason

for requiring a block cipher to satisfy the diffusion property which is that a small change in the plaintext, maybe for example in one or two positions, should produce an unpredictable change in the cipher text.

During the threat posed by exhaustive key searches it is likely that the attacker tries a key that differs from the correct value in only a small number of positions. If there were any indications that the attacker had, for instance, tried a key which disagreed with the correct key in only one position, then the attacker could stop his search and merely change each position of that specific wrong key in turn. This would significantly reduce the time needed to find the key and is another undesirable property. Thus block ciphers should satisfy the confusion property which is, in essence, that if an attacker is conducting an exhaustive key search then there should be no indication that they are 'near' to the correct key (Singh, Simon, 2000; Piper & Murphy, 2002).

Since block ciphers are used in CBC mode, the plaintext must be converted into an integral sequence of blocks. For this we append padding to the plaintext and a padding length of 1 byte. The padding length must be equal to all bytes of the padding, and the total length (the plaintext, the padding, and the padding length) must be a multiple of the block size. When the cipher text is decrypted, the last byte specifies the length of the padding to be removed. The padding structure is also checked and an error is issued if it is not valid.

Note that the padding does not need to be the shortest one. It can actually be longer in order to hide the real size of the plaintext to a potential adversary (Vaudenay, 2006).

Method

We represent an algorithm in turbo C++ programming language. The benefit of this algorithm is that it uses diffusion of 15 bits for every one byte using a binary tree and doubling the size into two bytes. Also the block cipher is used for every byte. We use a function to diffuse the 15 bits between the bits of plain text, $F(x)$, consists of a

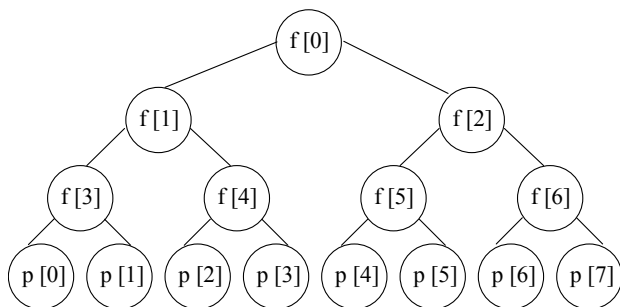


Fig1. Diffusion of 15 bits, $f[]$ are the $F(x)$ bits and $p[]$ are the plain text bits in level-order tour of a binary tree

$T(0)=f[0]; T(1)=f[1]; T(2)=f[2]; T(3)=f[3]; T(4)=f[4];$
 $T(5)=f[5]; T(6)=f[6]; T(7)=p[0];$
 $T(8)=p[1]; T(9)=p[2]; T(10)=p[3]; T(11)=p[4]; T(12)=p[5];$
 $T(13)=p[6]; T(14)=p[7];$

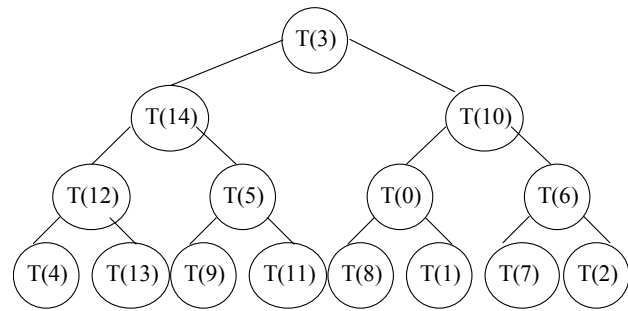


Fig.2. A typical sample of the new tree which is used to make the in-order tour of the binary tree

prime number more than the length of the plaintext (p), the key of the transposition cipher (k), a (g) number generator less than p , a (c) positive constant number less than eight and x that represents the ($n-1$)th character of the plaintext which starts from 0. The equation (1) shows the $F(x)$ function. To reduce the result of $F(x)$ in the range of 0 to 127 (to make the binary tree), $F(x)$ will be reduced by performing $F(x) \text{ mod } 128$.

$$F(x) = (k.g^x + c) \text{ mod } p \quad (1)$$

Now suppose that the outputs of $F(x)$ in bits are $f[0], f[1], f[2], \dots, f[6]$ and the bits of plain text are $p[0], p[1], p[2], \dots, p[7]$. We can diffuse these 15 bits in the binary tree shown in Fig 1 that can be written as follows if we name the binary tree nest as $T(0..14)$ and will be substituted by $f(0..6)$ and $p(0..7)$ (Fig.1):

In this paper, the messages can be represented as $M=m_0||m_1||\dots||m_n$ ($n: 0 \dots$ infinite) in which m_i is a character that is consists of 8 binary bits while converted from the ASCII codes $m_i=b_0||b_1||\dots||b_8$ ($i:1..n$). We used another function for filling the binary tree in a manner that is not predictable, the function $Y(j)$ is as follows in equation (2): $Y(j)=(g^k+c) \text{ mod } 15$ $j:0..14$, $Y(j): 0 .. 14$ (2)

In which the key of the transposition cipher (k), a (g) number generator less than p , a (c) positive constant number less than eight. Notice that in this function we will repeat the calculations until all the numbers of nests in the binary tree (i.e. $0..14$) are produced since some of the produced numbers will be produced two times. In the random number generation of $0..14$ that are produced in a loop, c and k are increased plus one in each loop which is repeated.

Now in this stage the new tree will be constructed from the normal tree with $T(0..14)$ components in a manner that the first component of the array $Y(j)$ is the position of the first component of the array T and will be continued until the new tree is filled that is illustrated in Fig 2.

Recall that in an *in-order tour* of a binary tree, codes are in Appendix 1, we traverse the tree by first visiting all the nodes in the left subtree using an in-order tour, then visiting the root, and finally visiting all the nodes in the right subtree using an in-order tour. To use the transposition cipher with k key the in-order tour of the tree will be written and after converting to the particular ASCII

code of the 16 bits in which the last bit is randomly produced, the transposition cipher algorithm is applied to the ASCII characters.

But in a manner that "z" character is not used for filling the empty positions of the transposition cipher, instead we use random characters from 256 characters of ASCII code.

Encryption algorithm

- Input Message (M)
- Input Keys (k,p,g,c)
- Convert Message Characters to Bits
- Calculate $F(x) = ((kg^x + c) \bmod p) \bmod 128$ for every 8 bits (one character)
- Construct primary level-order tour of binary tree using extra bits (f[0],...,f[6]), plaintext character bits (p[0],...,p[7])
- Calculate $Y(j) = (g^k + c) \bmod 15$ to find the new positions of the tree bits $Y(j), j:0..14$
- Construct the secondary binary tree according to the new situations calculated by $Y(j)$
- In-order tour of the secondary binary tree is calculated
- Calculate the 16th bit of the double character randomly
- Convert the 16bits in two 2 ASCII characters concatenated to each other to form the new plaintext
- Encrypt the new plaintext using transpositional cipher algorithm and k key and a random character instead of "z" to form the ciphertext

Decryption Algorithm

- Input Cipher text
- Input keys (k,g,c)
- Decrypt the cipher text using transposition cipher algorithm and k key
- Convert the ASCII characters to bits
- Every 16 bits (left to right) will be presented as one secondary binary tree
- Reverse in-order tour of the decrypted cipher text is calculated to form the secondary binary tree
- Reverse level-order tour of the secondary binary tree.
- Calculate $Y(j) = (g^k + c) \bmod 15$ to find the positions of the tree bits $Y(j), j:0..14$
- Construct the primary binary tree using extra bits (f[0],...,f[6]), plaintext character bits (p[0],...,p[7]) by using $Y(j)$
- Plaintext will be resolved from converting the resulted bits into ASCII characters

Discussion

Using this algorithm with this kind of transposition cipher can make the cipher unbreakable. It doubles the length of the plaintext and diffuses the bits of plain text by using the in-order tour of binary tree.

Since we have used the in-order tour of binary tree, no one can guess the original plaintext. This algorithm has another advantage that the results of $F(x)$ bits are diffused by the in-order tour of binary tree too. This cipher uses 256 characters of ASCII code. The key which is needed for encryption is:

Key: (k,p, g, c)

And the key which is needed for decryption is:

Key: (k,g,c)

When determining the order of an algorithm, we are only concerned with its category, not a detailed analysis of the number of steps. We obviously want to use the most efficient algorithm in our programs. Whenever possible, choose an algorithm that requires the fewest number of steps to process data. We want to show that this method can be strongly stable against the attacks of transposition cipher which are listed as below:

1. If the attacker checks all divisors of the length of the cipher text as a key of transposition cipher he/she will find the plaintext. And the order of finding the plaintext is as follows:
2. We had to use just the character z at the end of the cipher text according to the key.
3. If we give a key more than length of the key the plaintext will not be encrypted. For example if key is more than plaintext length then: Plaintext = **flower**
Key = 7
Cipher text = **flowerz**
4. The order of encryption of transposition cipher according to the following algorithm is $O(2n)$:

Input(text, key)

$j \leftarrow -1;$

for $i \leftarrow 0$ *to* length of (text)-1 *do*

if $(i \bmod \text{key} = 0)$ *then* $++j;$

$\text{cipher}[j][i \bmod \text{key}] \leftarrow \text{text}[i];$

//construct the column model using k key with the order of $O(n)$

if (length of (text) mod key != 0) *then*

for $z \leftarrow$ length of (text) mod key *to* (key-1) *do*

$\text{cipher}[(\text{length of (text)} / \text{key})][z] = 'z';$

//adding 'z' characters where ever needed with the order of $\Theta(k) = O(n) \quad k \leq n$

Return

5. The order of decryption of transposition cipher according to the following algorithm is $O(n)$:

Input(cipher, key)

$a \leftarrow 0;$

for $i \leftarrow 0$ *to* length of (cipher)/key-1 *do*

for $j \leftarrow 0$ *to* key-1 *do*

$\text{text}[a] \leftarrow \text{cipher}[(i/\text{key}) * j + i];$

$++a;$

//decrypt the ciphertext using transposition cipher and k key with the order of $O(n)$

6. The complexity of brute force attack against transposition cipher according to the following algorithm is $O(n^2)$:

Input(cipher)

for key $\leftarrow 1$ *to* length of (cipher) *do*

if (length of (cipher) mod key = 0)

//finds the divisors of cipher text and takes it as a

key with the order of $O(n)$

for $i \leftarrow 0$ *to* (length of (cipher) / key) - 1 *do*

for $j \leftarrow 0$ *to* $j \leq \text{key}-1$ *do*

```

text[a] ← cipher[(length of (cipher / key)
* j + i];
++a;
//checks the divisor keys in the cipher text to reveal
the plaintext

```

$$Order = \sum_{k=1}^n \sum_{i=0}^{\frac{1}{k}-1} \sum_{j=0}^{k-1} 1 = \sum_{k=1}^n \sum_{i=0}^{\frac{1}{k}-1} k = \sum_{k=1}^n l = nl = n^2$$

Our method characteristics:

- 1.If the attacker can break the transposition part, again the brute force is the only way to break the cipher and its order is calculated as below.
- 2.Instead of using the character z we use a random character from the 256 characters of ASCII code. This makes the cipher safer.
- 3.Because we use 2 functions to fill the binary tree and in-order tour so we can give a key more than the length of the plaintext.

Plaintext = **flower**

Key = 7 p=19 g=5 c=2

Cipher text = 'rC6j?!#!#r`π¼6τ

The order of encryption method according to the following algorithm is $O(3n^2)$:

Input(plaintext,key,p,g,c)

$v \leftarrow 0, z \leftarrow 1, y \leftarrow 1, flag \leftarrow 1, l \leftarrow 0, c1 \leftarrow c, k \leftarrow$

key;

```

while (l != 15) do
  for i ← 1 to key do
    y ← g * y;
    form1[i] ← (y + c) mod 15;
    if (l >= 1) then
      for j ← 0 to l-1 do
        if (form1[j] = form1[i]) then
          flag ← 0;
          if (flag = 1) then
            ++l;

```

$++key, ++c, flag \leftarrow 1, y \leftarrow 1;$

//creates 15 numbers between 0 to 14 with the order of $O(2r^2)$

```

for i ← 0 to length of (plaintext) - 1 do
  for j ← 0 to 7 do
    bit_text[j] ← j'th bit of plaintext[i];
    x ← i;

```

```

y ← 1;
for j ← 0 to x-1 do
  y ← g * y;

```

//order of the power according to the last for is $O(r^2)$

$form2 \leftarrow ((k * y + c1) \bmod p) \bmod 128;$

$y \leftarrow 1;$

```

for j ← 0 to 7 do
  bit_form2[j] ← j'th bit of form2[i];
  for j ← 1 to 7 do
    tree1[j-1] ← bit_form2[j];
  for j ← 7 to 14 do
    tree1[j] ← bit_text[j-7];
  for j ← 0 to 99 do
    tree2[j] ← '';

```

```

for j ← 0 to 14 do
  tree2[form1[j]] ← tree1[j];
  tree4 ← inorder(tree2,tree2[0],0);
//the order of in_order tour is O(n) but because we know
the shape of tree it is O(1)
  tree4[15] ← a random bit;
  cipher1[2 * j] ← character of tree4[0...7];
  cipher1[2 * j + 1] ← character of tree4[8...15];
  v ← 0;
  j ← -1;
  for i ← 0 to length of (cipher1) - 1 do
    if (i mod k = 0) then
      ++j;
      cipher2[j][i mod k] ← cipher1[i];
//starts the transposition cipher with the order of O(n)
  if (length of (cipher1) mod k != 0) then
    for u ← length of (cipher1) mod k to k-1 do
      cipher2[length of (cipher1) / k][u] ← a random character;
//puts the random characters according the key with the
order of O(n)
  return cipher2;

```

- 4.The order of decryption method according to the following algorithm is $O(2n^2)$:

Input(cipher,key,g,c)

$a \leftarrow 0, k \leftarrow key, v \leftarrow 0, m \leftarrow 0;$

$l \leftarrow$ length of (cipher);

while (t != 15) **do**

for i ← 1 to key **do**

$y \leftarrow g * y;$

$form1[t] \leftarrow (y + c) \bmod 15;$

if (t >= 1) **then**

for j ← 0 to t-1 **do**

if (form1[j] = form1[t]) **then**

$flag \leftarrow 0;$

if (flag = 1) **then**

$++t;$

$++key, ++c, flag \leftarrow 1, y \leftarrow 1;$

//creates 15 numbers between 0 to 14 with the order of $O(2r^2)$

for i ← 0 to l/k-1 **do**

for j ← 0 to k-1 **do**

$ciph[a] \leftarrow cipher[(l / k) * j + i];$

$++a;$

//opens the transposition cipher with the order of $O(n)$

for j ← 0 to 98 **do**

$tree1[j] \leftarrow '';$

for j ← 0 to 14 **do**

$tree1[j] \leftarrow 2;$

$invinor \leftarrow invinorder(tree1,tree1[0],0);$

//finds the positions of a default tree in in_order tour form with the order of $O(1)$

for i ← 0 to l-1 **do**

for j ← 0 to 7 **do**

$bit_ciph[j] \leftarrow j'th \text{ bit of } ciph[i];$

for j ← 8 to 15 **do**

$bit_ciph[j] \leftarrow (j-8)'th \text{ bit of } ciph[i+1]$

for j ← 0 to 14 **do**

```
inorder[j] ← bit_ciph[j];
for j ← 0 to 14 do
tree1[invinor[j]] ← inorder[j];
for j ← 0 to 14 do
tree2[form1[j]] ← tree1[j];
for j ← 7 to 14 do
bit_text[j-7] ← tree2[j];
text[m] ← character of bit_text[0..7];
++i, ++m;
//decrypts the tree cipher with the order of O(n)
return text;
```

5. The only way to break this cipher system is to brute force and make exhausted searches with the complexity of $((2^8)^2)^n$ in which it represents the double sized cipher text and n is the plaintext length.

This problem has the complexity of 512^n , where n is the plaintext length. The plaintext of eight or more characters would provide more resistance to a brute force attack in comparison to 128-bit AES and has the same resistance to brute force attack as 256-bit AES.

This methodology does have some potential drawbacks. The most important drawback is that the size of the encrypted plaintext will be doubled. For areas with low bandwidth or limited storage capacity this cipher is not useful. However for most communication channels where encryption is required, an increase in the plaintext size will not have a significant impact. Also using a good random generator in function $F(x)$ can make the cipher effective. All of the bits are important because if one of the bits is lost the remainder of the message will be useless.

Conclusion

Although the classical transposition cipher is not secure when it is used to encrypt unpadded messages, this and other stream ciphers can be made as secure as most block ciphers by using two functions, one ($F(x)$) for producing the 7 extra bits of padding to each byte to diffuse the language characteristics and the second function ($Y(j)$) for positioning the bits in the binary tree to use its in-order tour in the encryption and decryption process that will make it as secure as 256-bit AES cipher system. This method can resolve all the limits of the classical transposition cipher. The methodology can also be used for a 64 bit plaintext resulting in 128 bit cipher text including of 127-nest binary tree plus 1 random bit that will produce the whole 128 bits. Considering that our method was used for 8 bit plaintext and was as secure as the 256-bit AES and if it is used for a 64-bit plaintext its security will be increased dramatically.

It is worth mentioning that if the key management is not well-provided, all strong cipher will be compromised easily. Our investigations find that there is a paucity of data on improvements of transposition ciphers. When designing solutions to programming problems, we are concerned with the most efficient solutions regarding time and space. We will consider memory requirements at a later time. Speed issues are resolved based on the

number of steps required by algorithms. Finally we mention that if in this method, the number of Child is not the same (we did it with $\text{maxChild}=2$), it will add to its strength dramatically that our future work will be done on this subject.

References

1. Barker WG (1977) Cryptanalysis of the Hagelin Cryptograph. Aegean Park Press.
2. Deavours CA (1980) The black chamber: a column; how the British broke enigma. *Cryptologia*. 4 (3), 129-132.
3. Deavours CA and Kruh L(1985) Machine Cryptography and Modern Cryptanalysis. Norwood MA, Artech House.
4. Diffie W and Hellman ME(1979) Privacy and authentication: an introduction to cryptography. *Proc. IEEE*. 67 (3) 397-427.
5. Gaines HF (1956) Cryptanalysis. Am. Photographic Press, 1937.
6. Hagelin BCW (1994) The story of the Hagelin cryptos. *Cryptologia*. 18 (3), 204-242.
7. Hodges A (1983) Alan Turing: The Enigma of Intelligence. Simon & Schuster.
8. Kahn D (1967) The Code breakers: The story of secret writing. New York, Macmillan Publ. Co.
9. Kahn D (1991) Seizing the enigma. Boston: Houghton Mifflin Co.
10. Konheim AG (1981) Cryptography. *A Primer*, New York, John Wiley & Sons.
11. Mao Hewlett W-Packard (2003) Company modern cryptography: theory & practice. Prentice Hall PTR.
12. Piper F and Murphy S (2002) Cryptography: a very short introduction. Oxford Univ. Press.
13. Rivest RL (1981) Statistical analysis of the Hagelin cryptograph. *Cryptologia*, 5 (1), 27-32.
14. Scherbius A (1928) Cipherring machine. U.S. Patent #1,657,411.
15. Schinier B (1996) Applied cryptography. John Wiley & sons Inc. New York.
16. Shannon CE (1949) Communication theory of secrecy systems. *Bell System Tech. J.* 28 (4), 656-715.
17. Singh, Simon (2000) The code book: the science of secrecy from ancient Egypt to quantum cryptography. Anchor Book.
18. Sinkov A (1966) Elementary cryptanalysis, Math. Asso. Am.
19. Vaudenay S (2006) A Classical introduction to modern cryptography, Springer Sci.+Business Media, Inc.
20. Welchman G (1982) The hut six story: breaking the enigma codes. New York, McGraw-Hill.

Appendix

In-order tour function:

With $mch = 2$;

```
func inorder (str1[],st,po)
  if (st != '') then
    inorder(str1,leftMostChild(str1,po,
mch),(mch*po)+1);
    str2[v]= st;
    ++v;
    inorder(str1,rightSibling(str1,(mch * po) + 1,mch),
(mch * po) + mch);
  return str2;
```

func leftMostChild(str1,pos,maxChild)

```
x = 1;
for i--0 to 98 do
  str[i] = str1[i];
while (x <= maxChild)
  i = maxChild * pos + x;
  if (str[i] != '')
    return str[i];
  ++x;
return '';
```

func rightSibling(str1,pos,maxChild)

```
for i--0 to 98 do
  str[i] = str1[i];
p = pos mod maxChild;
if (p = 0) then
  return '';
if (str[pos] != '')
  while (p < maxChild) do
    if (str[pos + 1] != '') then
      return str[pos + 1];
    ++p, ++pos;
  return '';
```

Reverse in-order tour function:

With $mch = 2$;

```
func invinorder(str1[],st,po)
  if (st != '') then
    invinorder(str1,leftMostChild1(str1,po,
mch),(mch*po)+1);
    str2[v]= st;
    ++v;
    invinorder(str1,rightSibling1(str1,(mch * po) +
1,mch), (mch * po) + mch);
  return str2;
```

func leftMostChild1(str1,pos,maxChild)

```
x = 1;
for i--0 to 98 do
  str[i] = str1[i];
while (x <= maxChild)
  i = maxChild * pos + x;
  if (str[i] != '')
    return str[i];
  ++x;
return '';
```

func rightSibling1(str1,pos,maxChild)

```
for i--0 to 98 do
  str[i] = str1[i];
p = pos mod maxChild;
if (p = 0) then
  return '';
if (str[pos] != '')
  while (p < maxChild) do
    if (str[pos + 1] != '') then
      return str[pos + 1];
    ++p, ++pos;
  return '';
```