HRO ENCRYPTION SYSTEM

KHALID MOHAMMAD NAHAR, OSAMA MAHMOUD ABU ABBAS, AND MOHAMMAD AHMAD TUBISHAT
Computer Science Department, IT Faculty, Yarmouk University, Jordan

ABSTRACT

Encryption is the process of translating data into a secret code. This paper intends to introduce a new encryption system (HRO system). This system is a combination of three known algorithms: hash based encryption algorithm, RSA algorithm, and one-time pad algorithm. The purpose of HRO system is to expedite the encryption and decryption processes and to reduce the vulnerability of the system to external attacks. HRO system uses hash function and random number generator to increase security. Some conclusions that are arrived at concern the performance of the algorithm, hash function, random number generator and the time of encryption and decryption.

Keywords: *Encryption, Decryption, Hash Based Encryption, RSA, One-time pad, Collisions.*

1. INTRODUCTION

"Computer security" refers to techniques employed to ensure that the stored data cannot be accessed or compromised except by authorized individuals. Most security measures involve data encryption and passwords. Data encryption is the translation of data into a form that is intelligible only through a specific deciphering mechanism. Encryption is the most effective way for achieving data security [11, 12, 10].

Cryptography is the art of protecting information by transforming (encrypting) it into an unreadable format called ciphertext. Only those who possess a secret "key" can decipher (or decrypt) the message into a plaintext. Although modern cryptography techniques are virtually unbreakable, encrypted messages can still be broken by cryptanalysis (referred to as codebreaking). As the Internet and other forms of electronic communication became more prevalent, electronic security has become increasingly important [11, 12, 2, 10].

To improve the security level, some researchers worked on improving the encryption algorithms, others implemented new ones, while a third group studied and compared different encryption algorithms to select the most efficient. Some of the previous studies that show and analyze new methodologies and improve old ones in different fields and applications are discussed below. But it is important to note that these studies differ from our HRO system in methodology and analysis.

- Kwok-Wo Wong, Sun-Wah Ho, and Ching-Ki Yung modified the chaotic cryptographic scheme so as to reduce the length of the ciphertext to a size slightly longer than that of the original message. Moreover, they introduced a session key in the cryptographic scheme so that the length of the ciphertext for a given message is not fixed [7].
- Kwok-Wo Wong extended the chaotic cryptographic scheme so that it can perform both encryption and hashing to produce the ciphertext as well as the hash value for a given message. He analyzed the collision resistance of the proposed hashing approach [5].
- H.S. Kwok and Wallace K.S. Tang proposed a fast chaos-based image encryption system with stream cipher structure. The proposed keystream generator did not only achieve a very fast throughput, but also passed the statistical tests of up-to-date test suite even under quantization [6].
- Jun Wei, Xiaofeng Liao, Kwok-wo Wong, and Tao Xiang proposed a new chaotic cryptosystem. Instead of simply mixing the chaotic signal of the proposed chaotic cryptosystem with the ciphertext, a noise-like variable is utilized to govern the encryption and decryption processes. This adds statistical sense to the new cryptosystem [4].
- Rastislav Lukac and Konstantinos N. Plataniotis introduced and analyzed a new secret sharing scheme capable of protecting image data coded with B bits per pixel. The proposed input-agnostic encryption solution generates B-bit shares by combining bit-level decomposition/stacking with a {k, n}-threshold sharing strategy. They achieved perfect reconstruction by performing decryption through simple logical operations at the decomposed bit-levels without the need for any postprocessing operations [9].
- Chien-Yuan Chen, Cheng-Yuan Ku, and David C.Yen found ways to utilize the LLL algorithm to break the RSA system even when the value of d is large. According to their proposed cryptanalysis, if d satisfes $|\lambda - d| < N0.25$, the RSA system will be possible to be resolved computationally [2].
- Chang-Doo Lee, Bong-Jun Choi, and Kyoo-Seok Park proposed a block encryption algorithm which is designed for each encryption key value to be applied to each round block with a different value. This algorithm needs a short processing time in encryption and decryption, has a high intensity, and can apply to electronic commerce and various applications of data protection [1].
- Mohammad Peyravian, Allen Roginsky, and Nev Zunic described a symmetric-key encryption algorithm based on the use of an underlying one-way hash function. It is computationally more efficient and, most importantly, more secure than using traditional symmetric encryption tools [8].
- Timothy E. Lindquist, Mohamed Diarra, and Bruce R. Millard presented an implementation of One-Time Pad as a Java Cryptography Architecture (JCA) service provider, and demonstrated its usefulness on Palm devices [12].

2. METHODOLOGY

Our methodology in HRO system can be summarized as follows:

1. Creating an array of 100 elements (the charactecrs of the text). Each 16-bit element is created randomly by a random number generator.
2. Creating a 16-bit one-time pad (OTP) element randomly by a random number generator and storing this element somewhere in the array created in step 1.
3. Building the hash table depending on the array elements, one-time pad algorithm and a hash function.
4. Encrypting the array, including the one-time pad element, by RSA algorithm.
5. Sending to the receiver the encrypted array along with the one-time pad element and the encrypted message.
6. Decrypting, by the receiver, the array using RSA and one-time pad algorithms.

2.1 BUILDING THE ARRAY

In this phase an array of 100 elements is created. This process is performed only once. It is possible to re-create this array, for example once a month, to increase the security and to reduce the possibility of hostile attacks. Each element is 0/1 16-bit long and represents one character of the text (see figure 1). The characters used are: capital letters A-Z (26 characters), small letters a-z (26 characters), the digits 0-9 (10 characters), in addition to all the special characters [ ] ( ) _ ~ ! @ # $ % ^ & * + - / + \ | < = > . , ; ’ ” ? { } : ` enter, tab and the blank (36 characters).

The elements of this array are created randomly by a very powerful random number generator that changes its seeds every second. Because each element is 16-bit long, then the random number generator can choose each element from 2^{16} elements (permutations). Every time an element is created, we have to ensure that it was not previously created by searching the elements of the array that were created earlier. If the created element is already found in the array, then the random number generator must retry to create a different element, and so on. Because the number of permutations that the random number generator can choose from is very large (2^{16}), the chance of duplication or redundancy is very unlikely.

The size of the array is equal to the number of the characters of the text that can be used (explained above) plus one for the one-time pad element.

Figure 1: The general format of the array

		…				…				…				…
A	B		Z	a	b		z	0	1		9	!	#	

2.2 BUILDING ONE-TIME PAD ELEMENT

An 16-bit OTP element is created randomly by the same random number generator that is used in building the array. Like all other elements, the OTP element must be unique in the array and could be stored in one of the first 12^{th} positions in the array. The OTP element is stored in one of the twelve positions once a month. It is reasonable to make the position of OTP element secret between both the sender and receiver, for example, in the first position in January and in the second position in February, and so on. However, any other agreement between the sender and the receiver can be used to store this element in any position in the array. Because the agreement is secret between the two ends of communication (parties), HRO system becomes more secure. The OTP element is used in building the hash table in the next section.

2.3 BUILDING THE HASH TABLE

This table is constructed only once. The function of the hash table is to retrieve the encoded characters from it by O(1) time complexity only. This makes the speed of the encryption process faster than the traditional RSA and other algorithms. Due to the fact that the size of the hash table is equal to the load factor multiplied by the number of the characters, the size of this table is five times the number of the characters (see figure 2). We chose the load factor of the hash table in our system to be 5 in order to reduce the chance of occurring collisions in the process of building the hash table. Since HRO system does not send the hash table to the receiver, there will be no problem if its size is relatively large. The ideal size of the hash table can be computed according to the following formula:

Hash Table Size = Load Factor * no. of Characters
= 5 * no. of Characters

Figure 2: The general format of the hash table

0	0101010011010110
1	1110010010111011
2	0011001111101111
3	1111101010101000
.	.
.	.
.	.
499	1001111011111001

For Storing the Encoded Characters

The steps of building the hash table is summarized as follows:

1. For each element in the array created in section 2.1, excluding the OTP element, the OTP algorithm is applied to encrypt each character in the text according to the following formula:

   Hash[L] = C ⊕ OTP

   Where L represents the location of the encoded letter in the hash table, C represents the array letter code, and OTP represents the one-time pad element.
2. The following hash function is used to find the locations of the encoded elements produced in step 1 in the hash table (the previous step):

   L = D Mod S

   Where L represents the location of the encoded letter in the hash table, D represents the decimal letter code, and S represents the size of the hash table. The decimal letter code is computed by

converting the 16-bit binary code in the array to the equivalent decimal code.

3. Should collisions occur due to the hash function used in step 2, then a separate chaining method is employed to handle them. In this method each position in the hash table is an item and a link, essentially, the head of a list [3]. Obviously, when the number of permutations that the random number generator can choose from is large, the probability of occurring collisions becomes very high. If, on the other hand, we reduce the number of permutations, the probability of duplicate elements produced by the random number generator becomes high. However, and as previously explained, we chose the load factor to be 5 in order to reduce the chance of occurring collisions.

The one-time pad (OTP) algorithm mentioned in step 1 is summarized as follows [11]:

1. Choose a random bit string as the key. In HRO system, this key is chosen randomly as previously explained in section 2.2 (building one-time pad element).
2. Convert the plaintext into a bit string. Here, in HRO system, the plaintext is already converted into a 16-bit for each character and is stored in the array created in section 2.1.
3. Compute the XOR (eXclusive OR) of these two strings (produced by step 1 and step 2), bit by bit.

2.4 ENCRYPTION OF THE ARRAY

A new array of all the characters including the OTP element is now created. Note that the creation of this array, as in case of the hash table, is performed only once. But it is possible to re-build it again on a monthly basis in order to increase the security level by making the possibility of attacking the array a very difficult process. In this phase each element in the array (including the OTP element) is encrypted by RSA algorithm.

RSA algorithm is summarized as follows [11, 10]:

1. Choose two large primes, p and q.
2. Compute n = p*q and z = (p-1)*(q-1).
3. Choose a number that is relatively prime to z and call it d.
4. Find e such that e*d=1 mod z.
5. To encrypt a message (plaintext), P, compute $C = P^e \pmod n$.
6. To decrypt a message (ciphertext), C, compute $P = C^d \pmod n$.

Note that in order to perform the encryption, e and n are needed, while d and n are required to perform the decryption. Therefore, the public key consists of the pair (e, n), and the private key consists of (d, n).

It is obvious that RSA is applied only to the 101 characters stored in the array and not to all the characters of the message. In the latter case, RSA may be applied to a single character, A, for example, hundreds of times, but in HRO system RSA is applied only one time to each character stored in the array and not in the message. This speeds up the encryption process.

2.5 DECRYPTION PROCESS

Now, the OTP element, the encrypted array, and the hash table are constructed. The sender sends the encrypted message and the encrypted array which includes the OTP element to the receiver. The message can be a file, an e-mail message or any other piece of information. As explained above, the sender and the receiver know the position of the OTP element in the encrypted array. The sender then performs the following steps in order to decrypt the ciphertext:

1. Divide the message into 16-bit elements.
2. Apply RSA algorithm to each 16-bit element in the array including the OTP element.
3. Retrieve the encrypted OTP element from its position in the encrypted array.
4. Apply the OTP algorithm to each 16-bit element by using the decrypted OTP element that is produced in step 2.
5. Match the code to the corresponding character in the encrypted array for each element produced in step 4.

Figures 3 and 4, on page 4, illustrate the encryption and decryption processes in details.

3. EXAMPLE

Suppose that the sender sends the following message to the receiver: ABED, then the steps for encrypting and decrypting this message using HRO system are:

1. Building the array: suppose the random number generator generates the array of characters (shown in figure 5) with the OTP element in the first position:

Figure 5: The random generated array.

1001000000001011	1101000111010010
Pad	A
1001001110011011	0111000111011011
B	C
0011101001110011	0111010011111001
D	E

2. Building the hash table: The hash table corresponds to the array generated in step 1 is constructed as follows:

   2.1 Computing the decimal letter code for the characters: The decimal letter code for the character A = 53714, for B = 37787, for C = 29147, for D = 14963, and for E = 29945. The decimal letter code for each character mod 500 represents its location in the hash table. So, the location of the character A is 53714 mod 500 = 214, the location of character B is 287, the location of character C is 147, the location of character D is 463, and the location of character E is 445.

Figure 3:The Encryption Process

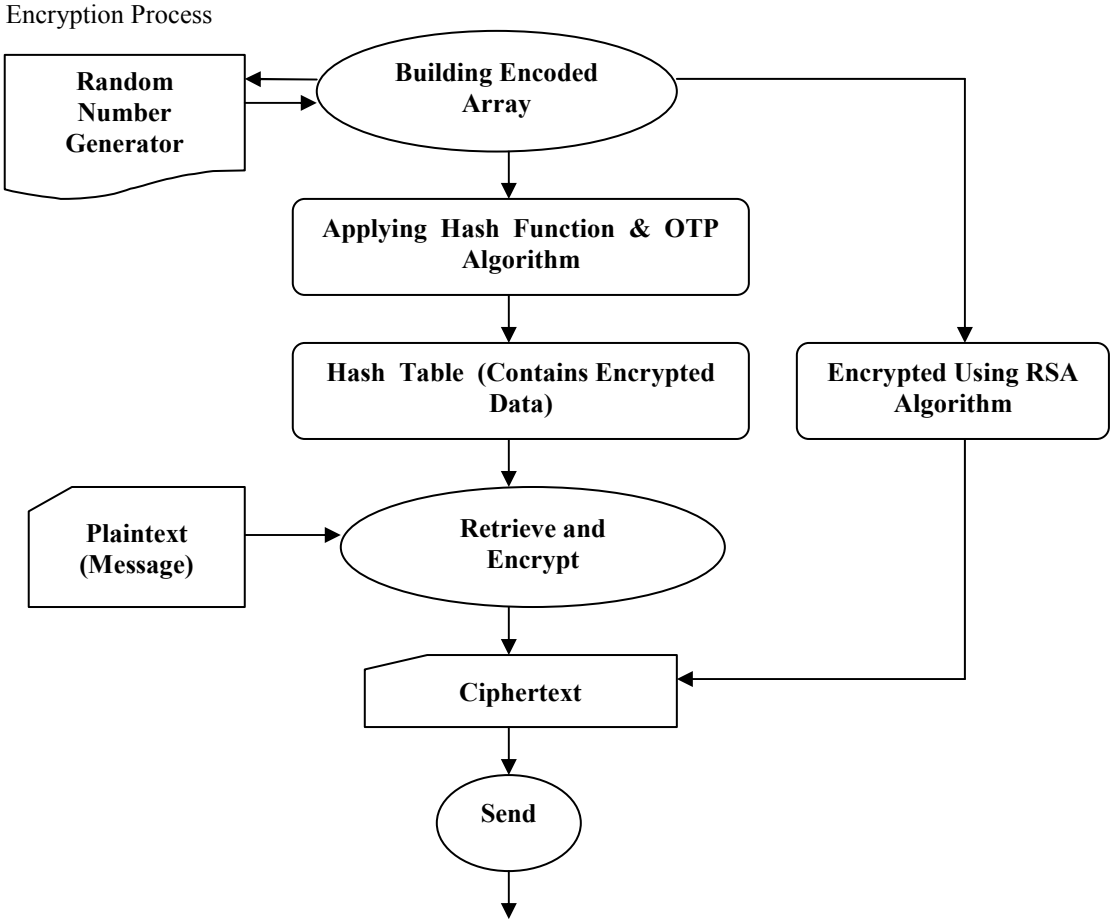
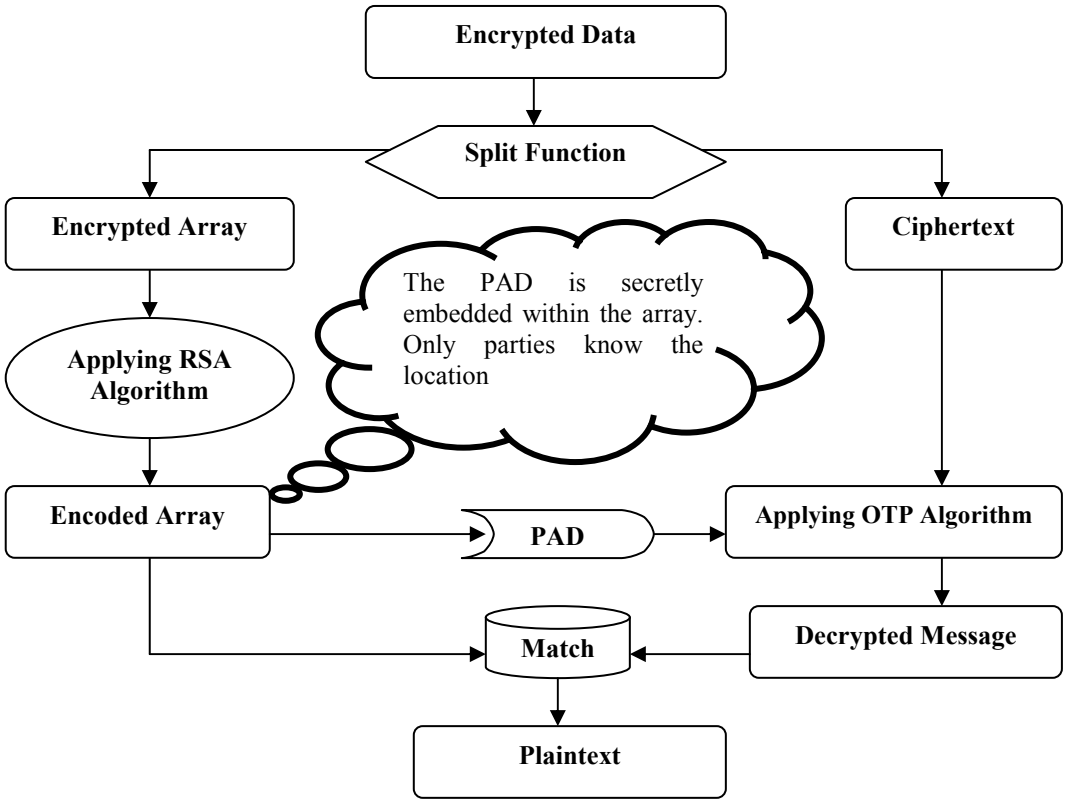


Figure 4:The Decryption Process



2.2 Apply OTP algorithm to the keys of the array, i.e. each 16-bit character XOR the OTP element. For character A:

1101000111010010 ⊕ 1001000000001011 = 0100000111011001.

See figure 6 for the other characters: B, C, D, and E.

Figure 6: The corresponding hash table

.	
.	
214	0100000111011001
.	
.	
147	1110000111010000
.	
.	
287	0000001110010000
.	
.	
445	1110010011110010
.	
.	
463	0101010100111000
.	
.	

3. Applying RSA algorithm: The array produced in step 1 is encrypted by the RSA algorithm. p and q are chosen randomly as prime numbers. Suppose they are respectively 227 and 317, then n = 71959, and z = 71416. Suppose d = 47611, then according to the formula d*e = 1 mod z, e = 3. And according to the following formula $C = P^e \pmod n$ the OTP element is encrypted to (0111101111110000). Character A is encrypted to: (0111101111001010). Figure 7 shows RSA encryption for the other letters: B, C, D, E.

Figure 7: The encrypted array by RSA

0111101111110000	0111101111001010
Pad	A
0101110011010000	0100100001110001
B	C
1100000100011101	0100111010111010
D	E

4. The array produced in step 3 along with the following encrypted message are sent to the receiver:
0100000111011001000000111001000001110010011110010010101010100111000
5. The encrypted message is divided into 16-bit elements by the receiver. In our example, dividing the message gives the following form:
0100000111011001
0000001110010000
1110010011110010
0101010100111000
6. Applying RSA algorithm: RSA algorithm is applied to the array received in step 4 according to the previous values of p, d, n, z, d, and the following formula: $P = C^d \pmod n$. For example the decryption of the OTP element is 1110100101110100. Figure 8 shows the result of decrypting the array using RSA algorithm.

Figure 8: The decryption array after applying RSA.

1001000000001011	1101000111010010
Pad	A
1001001110011011	0111000111011011
B	C
0011101001110011	0111010011111001
D	E

7. Applying OTP algorithm: The receiver retrieves the OTP element from a certain position in the array according to a predetermined agreement between the sender and receiver. In our example the OTP element is stored in the first position of the array produced in step 6. OTP algorithm is applied to the encrypted array. For example, the character A is decrypted by OTP algorithm as follows: ⊕ = , and so on. Figure 9 shows the result of decrypting the array using OTP algorithm.

Figure 9: The decryption of the array after applying OTP.

1001000000001011	0100000111011001
Pad	A
0000001110010000	1110000111010000
B	C
0101010100111000	1110010011110010
D	E

8. Finally, each element in the array produced in step 7 will correspond to one of the encrypted characters in the message. For example, the first 16-bit in the encrypted message (0100000111011001), see step 5, is matched with the second position in the array which represents character A. The second 16-bit in the encrypted message (0000001110010000), is matched with the third position in the array which represents character B. The third 16-bit in the encrypted message (1110010011110010), is matched with the sixth position in the array which represents character E. Finally, the fourth 16-bit in the encrypted message (0101010100111000), is matched with the fifth position in the array which represents character D. So, the decrypted message is ABED.

4. POSSIBLE ATTACK

Attackers need to penetrate two levels of security to break the encryption of HRO system. First, they need to attack RSA algorithm. This is a complicated process because the security of the RSA is based on the difficulty of factoring large numbers. Factoring a 500-digit number, for example, requires 10^{25} years using

brute force and 300 years using mathematical equations and methods [11]. Second, they need to know the position of the OTP element in the encrypted array, which is a secret between the sender and the receiver, to break the encryption. Therefore, they will have to try all the hundred positions of the array in each of the one hundred iteration. So the complexity needed is O(100*100) which is O(10000). As a result, it is very difficult to break our HRO system because it is a combination of several levels and subsystems.

5. TIME AND SPACE COMPLEXITY

Efficiency of any algorithm is measured by its complexity which means both space and time consuming. Our concern focus on the worst case.

Fortunately, HRO system has a fixed space complexity for the hash table and the array which allows for both to be placed in the RAM. The hash table consumes about 500 locations because of the load factor which is 5 times the number of elements. The array consumes about 100 locations which has also a fixed space complexity. According to the rules of complexity, fixed complexity could be overlooked.

The time complexity of HRO system includes:

- The time needed to generate the array in the best case is O(100) if random number generator generates all the hundred elements without duplications. Should duplications occur, the time complexity is still fixed and needs O(100*100) in the worst case.
- The time needed to build the hash table, which is also fixed, is O(500) in the worst case.
- The time needed to encrypt the array using RSA algorithm in the worst case is O(n), where n is the length of the array encrypted. In HRO system the length of the array is 100 elements. This one is also fixed.
- The time needed for decryption is based mainly on the length of the message M. We need to pass on all the characters in the message. So, in the worst case we need O(100*n), where n is the length of the message M.

  As a result, the time complexity of HRO system is O(n), where n is the length of the message M. However, if the receiver rebuilds the hash table, the time complexity of decryption is reduced to O(n).

6. PRACTICAL ANALYSIS

Several experiments are performed on HRO system. HRO system is tested for building the encoded array when each element in this array is 8-bits long. In this case the number of permutations that the random number generator can choose from is 2^8. The results show that the average number of times the random number generator generates duplicate elements in an array of size 100 elements is 25.9. The same test is repeated when each element in the array is 16-bit long. In this case the results show that the average number of times the random number generator generates duplicate elements in the same array becomes 0.15.

However, HRO system has been also tested for building a hash table when the size of each element in the array is 8-bit and again when the size is 16-bit. The results show that the average number of occurring collisions when the size of the element is 8-bit is 0 and when the element size is 16-bit it becomes 12.4.

- HRO system is tested also for calculating the time of encryption and decryption processes. Figure 10 shows the results:

Figure 10: Time of Encryption and Decryption

Text Size (in characters)	Encryption Time	Decryption Time
100	0.06 Milliseconds	0.06 Milliseconds
1000	0.8 Milliseconds	0.8 Milliseconds
10000	50 Seconds	50 Seconds
Average	**1304.7**	**2196.3**

As it is clear from the above table, the average time in milliseconds for encrypting and decrypting different sizes of texts is 1304.7and 2196.3 respectively. However, should the receiver rebuilds the hash table, then the decryption time is reduced.

7. CONCLUSIONS

This paper presents an encryption system that is more efficient than other encryption systems. It has many properties of RSA, OTP, and the hash-based algorithms. After analyzing the results of testing HRO system and running it under various circumstances, the following conclusions are obtained:

- The hash table expedites the process of retrieving the encoded characters which simply becomes a matter of picking a given character from the table since the location of that character is calculated in advance. Obviously, this retrieving process is much simpler and faster than searching all the elements of the array in time complexity of O(n).
- The time complexity for each encryption process and decryption process is O(n). This time is very fast comparing with other algorithms which need to re-build the array and the hash table every time.
- Using RSA and OTP algorithms in certain phases in our system makes HRO system more secure and less vulnerable to attacks.
- Also, storing the OTP element in an unknown position in the encrypted array makes HRO system more secure and less likely to be attacked.
- When the number of permutations that the random number generator can choose from is very large, the chance of duplication or redundancy of elements becomes very unlikely. This expedites the process of building the array.
- But when the number of permutations that the random number generator can choose from is very large, the probability of collisions to occur in the hash table becomes relatively high.

- The practical tests indicate that the random number generator and the hash function used in HRO system are very powerful.
- The average time for encrypting and decrypting different sizes of texts is relative very fast to other systems. This efficiency comes from applying RSA only on the characters stored in the array, not on each character in the message. This indicates that HRO system is very efficient system.

8. FUTURE WORK

Since binary representation is used for representing the characters of the message, then HRO system could be used to encrypt images as well. This is because a given image could be represented by a set of pixels with a set of bits represent the attributes of each pixel. Moreover, any bitmap images could be directly encrypted by HRO system with some modifications on the OTP element which may be a selected image or shape determined by an agreement between the parties.

REFERENCES

[1] Chang-Doo Lee, Bong-Jun Choi and Kyoo-Seok Park, "Design and evaluation of a block encryption algorithm using dynamic-key mechanism". *Future Generation Computer Systems*, Volume: 20, Issue: 2, Pages: 327 – 338, 2004.

[2] Chien-Yuan Chen, Cheng-Yuan Ku b and David C.Yen, "Cryptanalysis of large RSA exponent by using the LLL algorithm", *in Proceedings of The Tenth National Conference on Information Security,* Taiwan, Pages:45-50, 2000.

[3] Horowitz , Ellis /Sahni and Sartaj, *Fundamentals of computer algorithms*, Computer Science Press, 1978.

[4] Jun Wei, Xiaofeng Liao, Kwok-wo Wong, and Tao Xiang, "A new chaotic cryptosystem", *Chaos Solitons & Fractals* 30 (5): 1143-1152 Dec 2006.

[5] Kwok-Wo Wong, "A combined chaotic cryptographic and hashing scheme", *Physics Letters A*, Volume 307, Number 5, Pages: 292-298,2003.

[6] Kwok H.S., and Wallace K.S. Tang, "A fast image encryption system based on chaotic maps with finite precision representation", *Elsevier, Chaos, Solitons & Fractals*, 2006.

[7] Kwok-Wo Wong, Sun-Wah Ho, and Ching-Ki Yung, "A chaotic cryptography scheme for generating short ciphertext", *Physics Letters A*, Volume 310, Number 1, Pages: 67-73, 2003.

[8] Peyravian Mohammad, Roginsky Allen and Zunic Nev. "Hash-Based Encryption System". *Computers & Security,* Volume 18, Issue 4, Pages: 345-350, 1999.

[9] Rastislav Lukac and Konstantinos N. Plataniotis, "Bit-level based secret sharing for image encryption", *Pattern Recognition*, Volume 38, Number 5, Pages: 767-772, May 2005.

[10] Rivest, R., Shamir A., and Adleman L. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM,* Volume 21, Number 2, Pages: 120-126, 1978.

[11] Tanenbaum, Andrew S., *Computer Networks,* Fourth Edition, Prentice Hall PTR, 2003.

[12] Timothy E. Lindquist, Mohamed Diarra, and Bruce R. Millard, "A Java cryptography service provider implementing one-time pad", *in Proceedings of the 37th Annual International Conference on System Sciences*, Hawaii, USA ACM, IEEE Computer Society, 2004.