Scientific and Practical Cyber Security Journal (SPCSJ) 3(1): 45-57 ISSN 2587-4667 Scientific Cyber Security Association (SCSA)

Comparative Study of Cryptographic Algorithms

Mohanad Abdulhamid[1], and Nyagathu Gichuki[2]
[1]AL-Hikma University, Iraq
[2]University of Nairobi, Kenya

ABSTRACT: This paper presents a comparative study of two cryptosystems, Data Encryption Standard (DES) and the Rivest-Shamir-Adleman (RSA) schemes. DES is a symmetric (or private) key cipher. This means that the same key is used for encryption and decryption. RSA, on the other hand, is an asymmetric (or public) key cipher, meaning that two keys are used, one for encryption and the other for decryption. The objective of this paper is to implement these two schemes in software. The program is written in the Java™ language. It generates a key from a passphrase given by the user, encrypts and decrypts a message using the same key, for the case of DES. In RSA, decryption is done by computing the decryption key from the encryption key. Finally, the program returns the time taken to encrypt and decrypt a message.

KEYWORDS: Cryptographic algorithms; RSA; DES

The desire to communicate privately is a human trait that dates back to the earliest times, hence the need of cryptographic algorithms. The study of ways to disguise messages so as to avast unauthorized interception is known as cryptography. The terms encipher and encrypt refer to the message transformation done at the transmitter and the terms decipher and decrypt refer to the inverse transformation performed at the receiver. The primary reasons for using cryptosystems in communication are:

1. Privacy: Cryptosystems prevents unauthorized persons from extracting information from the channel (eavesdropping)

2. Authentication: Cryptosystems prevents unauthorized person from injecting information into the channel (spoofing). Sometimes as in the case of electronic funds transfer or contract negotiations, it is important to provide the electronic equivalent of a written signature in order to avoid or settle any dispute between the sender and receiver as to what message, if any, was sent.

3. Integrity: Integrity means that the content of the communicated data is assured to be free from any type of modification between the end points (sender and receiver).

4. Non-Repudiation: This function implies that neither the sender nor the receiver can falsely deny that they have sent a certain message.

The two widely accepted and used cryptographic methods are symmetric and asymmetric. Symmetric or private key ciphers use the same key for encryption and decryption, or the key used for decryption can be easily calculated from the key used in encryption. The main problem for symmetric key ciphers is the key distribution.

Asymmetric or public key ciphers are used to solve two of the most difficult problems of conventional encryption, one being the problem of key distribution and the other problem is associated with digital signatures for the purpose of authenticity of data and messages. In asymmetric keys, two keys are used; private and public keys. Public key is used for encryption and private key is used for decryption. Public key encryption is based on mathematical functions, computationally intensive and not very efficient for small mobile devices. They are almost 1000 times slower that symmetric techniques, because they require more computational processing power.

Symmetric encryption or private key systems fall into two general categories; Block Encryption, and Data Stream Encryption. In Block Encryption, the plain text is segmented into blocks of fixed size, each block is encrypted independently from the others. For a given key, a particular plain text block will therefore be carried into the same cipher text block each time it appears (similar to block encoding). With Data Stream Encryption, similar to convolution encoding, there is no fixed block size. Each

plain text bit, $\square_{\square}$ is encrypted with the ith element, $\square_{\square}$ of a sequence of symbols (key stream) generated with the key. The encryption is periodic if the key stream repeats itself after p characters for some fixed p, otherwise it's non-periodic.

An example of public key method is Rivest-Shamir-Adleman(RSA), while, Data Encryption Standard (DES) is an example of private key method. These two algorithms are considered in this paper. Some works on this topic can be found in literatures[1-5].

2- Cryptographic algorithms

2.1- The DES algorithm

2.1.1 Permutation of data: Plain text preparation

The numbers in the Tables 1 and 2 specify the bit numbers of the input to the permutation. The order of the numbers in the table corresponds to the output bit position; so for example, the initial permutation moves bit 58 to output bit 1 and input bit 50 to output bit 2.

Table 1 Initial permutation

Bit	0	1	2	3	4	5	6	7
1	58	50	42	34	26	18	10	2
9	60	52	44	36	28	20	12	4
17	62	54	46	38	30	22	14	6
25	64	56	48	40	32	24	16	8
33	57	49	41	33	25	17	9	1
41	59	51	43	35	27	19	11	3
49	61	53	45	37	29	21	13	5
57	63	55	47	39	31	23	15	7

Table 2 Final permutation

Bit	0	1	2	3	4	5	6	7
1	40	8	48	16	56	24	64	32
9	39	7	47	15	55	23	63	31
17	38	6	46	14	54	22	62	30
25	37	5	45	13	53	21	61	29
33	36	4	44	12	52	20	60	28
41	35	3	43	11	51	19	59	27
49	34	2	42	10	50	18	58	26
57	33	1	41	9	49	17	57	25

2.1.2 Key scheduling: Generating per round keys

The first step is to pass the 64-bit key through a permutation called Permuted Choice 1, or PC-1 for short. The table for this is given in Table 3. Note that in all subsequent descriptions of bit numbers, 1 is the left-most bit in the number, and *n* is the rightmost bit.

The 56-bit key is used to generate sixteen 48-bit sub keys, called K[1]-K[16], which are used in the 16 rounds of DES for encryption and decryption. The procedure for generating the sub keys, known as key scheduling is as follows:

Table 3 Permuted choice 1 (PC-1)

Bit	0	1	2	3	4	5	6
1	57	49	41	33	25	17	9
8	1	58	50	42	34	26	18
15	10	2	59	51	43	35	27
22	19	11	3	60	52	44	36
29	63	55	47	39	31	23	15
36	7	62	54	46	38	30	22
43	14	6	61	53	45	37	29
50	21	13	5	28	20	12	4

1. Set the round number R to 1.
2. Split the current 56-bit key, K, up into two 28-bit blocks, L (the left-hand half) and R (the right-hand half).
3. Rotate L left by the number of bits specified in Table 4, and rotate R left by the same number of bits as well.
4. Join L and R together to get the new K.
5. Apply Permuted Choice 2 (PC-2), Table 5, to K to get the final K[R], where R is the current round number.
6. Increment R by 1 and repeat the procedure until all the sixteen sub keys K[1]-K[16] have been generated. The above procedure is shown in Fig.1.

Table 4 Sub key rotation

Round Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Numberof bits to rotate	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 5 Permuted choice 2 (PC-2)

Bit	0	1	2	3	4	5
1	14	17	11	24	1	5
7	3	28	15	6	21	10
13	23	19	12	4	26	8
19	16	7	27	20	13	2
25	41	52	31	37	47	55
31	30	40	51	45	33	48
37	44	49	39	56	34	53
43	46	42	50	36	29	32

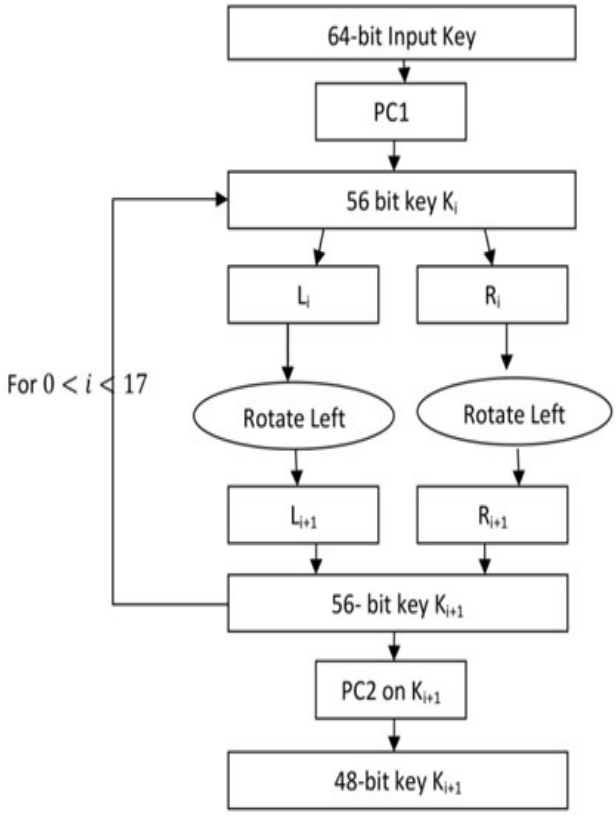

Fig.1 Key scheduling

2.1.3 DES core function

Once the key scheduling and plaintext preparation have been completed, the actual encryption or decryption is performed by the main DES algorithm. The 64-bit block of input data is first split into two halves, L and R. L is the left-most 32 bits, and R is the right-most 32 bits. The following process is repeated 16 times, making up the 16 rounds of standard DES. The 16 sets of halves are called L[0]-L[15] and R[0]-R[15].

1. R[I-1], where I is the round number, starting at 1 is taken and fed into the E-bit Selection Table, which is like a permutation, except that some of the bits are used more than once. This expands the number R[I-1] from 32 to 48 bits to prepare for the next step. This is given as in Table 6.

Table 6 E-bit selection

Bit	0	1	2	3	4	5
1	32	1	2	3	4	5
7	4	5	6	7	8	9
13	8	9	10	11	12	13
19	12	13	14	15	16	17
25	16	17	18	19	20	21
31	20	21	22	23	24	25
37	24	25	26	27	28	29
43	28	29	30	31	32	1

2. The 48-bit R[I-1] is XORed with K[I] and stored in a temporary buffer so that R[I-1] is not modified.
3. The result from the previous step is now split into 8 segments of 6 bits each. The left-most 6 bits are B[1], and the right-most 6 bits are B[8]. These blocks form the index into the S-boxes, which are used in the next step. The Substitution boxes, known as S-boxes, are a set of 8 two-dimensional arrays, each with 4 rows and 16 columns. The numbers in the boxes are always 4 bits in length, so their values range from 0-15. The S-boxes are numbered S[1]-S[8]. They are given in Tables 7 to 14.
4. Starting with B[1], the first and last bits of the 6-bit block are taken and used as an index into the row number of S[1], which can range from 0 to 3, and the middle four bits are used as an index into the column number, which can range from 0 to 15. The number from this position in the S-box is retrieved and stored away. This is repeated with B[2] and S[2], B[3] and S[3], and the others up to B[8] and S[8]. At this point, you now have 8 4-bit numbers, which when strung together one after the other in the order of retrieval, give a 32-bit result.
5. The result from the previous stage is now passed into the P-permutation, Table 15.
6. This number is now XORed with L[I-1], and moved into R[I]. R[I-1] is moved into L[I].
7. At this point we have a new L[I] and R[I]. Here, we increment I and repeat the core function until I = 17, which means that 16 rounds have been executed and keys K[1]-K[16] have all been used.

When L[16] and R[16] have been obtained, they are joined back together in the same fashion they were split apart (L[16] is the left-hand half, R[16] is the right-hand half), then the two halves are swapped, R[16] becomes the left-most 32 bits and L[16] becomes the right-most 32 bits of the pre-output block and the resultant 64-bit number is called the pre-output. This procedure is shown in Fig.2.

Table 7 Substitution box 1

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table 8 Substitution box 2

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Table 9 Substitution box 3

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Table 10 Substitution box 4

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Table 11 Substitution box 5

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Table 12 Substitution box 6

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Table 13 Substitution box 7

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Table 14 Substitution box 8

Row / Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 15 Permutation

Bit	0	1	2	3
1	16	7	20	21
5	29	12	28	17
9	1	15	23	26
13	5	18	31	10
17	2	8	24	14
21	32	27	3	9
25	19	13	30	6
29	22	11	4	25

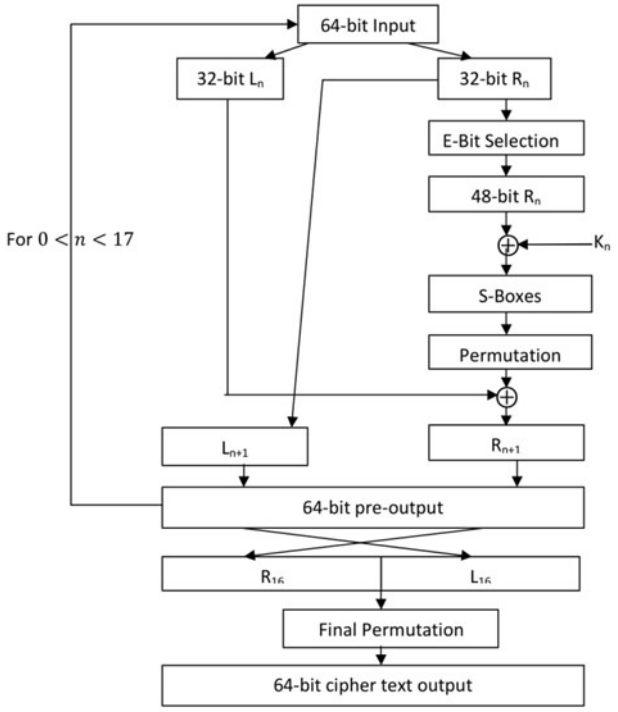

Fig.2. DES core function

2.1.4 How to use the S-Boxes

The purpose of this example is to clarify how the S-boxes work. Suppose we have the following 48-bit binary number:

11101100 10000100 10110111 11110110 00011000 10111100

In order to pass this through steps 3 and 4 of the Core Function as outlined in section 2.1.3, the number is split up into 8, 6-bit blocks, labeled B[1] to B[8] from left to right:

111011 001000 010010 110111 111101 100001 100010 111100

Now, eight numbers are extracted from the S-boxes, one from each box:

B[1] = S[1](11, 1101) = S[1][3][13] = 0 = 0000

B[2] = S[2](00, 0100) = S[2][0][4] = 6 = 0110

B[3] = S[3](00, 1001) = S[3][0][9] = 13 = 1101

B[4] = S[4](11, 1011) = S[4][3][11] = 11 = 1011

B[5] = S[5](11, 1110) = S[5][3][14] = 5 = 0101

B[6] = S[6](11, 0000) = S[6][3][0] = 4 = 0100

B[7] = S[7](10, 0001) = S[7][2][1] = 4 = 0100

B[8] = S[8](10, 1110) = S[8][2][14] = 5 = 0101

In each case of S[n][row][column], the first and last bits of the current B[n] are used as the row index and the middle four bits as the column index. The results are now joined together to form a 32-bit number which serves as the input to stage 5 of the Core Function (the P-permutation):

00000110 11011011 01010100 01000101

The final step is to apply the Final Permutation (Table 2) to the pre-output. The result is the completely encrypted text.

2.2. The RSA Algorithm

This algorithm can be summarized as:

1. Generate two large primes, □ *p* and *q*.

2. Compute $n = p \times q$ □ and $u(n)=(p-1)(q-1)$

3. Choose a number relatively prime to $u(n)$, and call it e (and should be less than $u(n)$). The program written has used 65537, the common value of e used in practice. It is possible that the Greatest Common Divisor of $u(n)$ and e is not equal 1. In this case, the key generation fails. A new set of p and q is required. Another case where the keys generated are not valid is when the message is greater than or equal to the product of p and q. This, too, is taken into account during the implementation.

4. Find d such that $e \times d=1 \mod u(n)$. d is determined using extended Euclidean algorithm.

A block of plaintext message M is encrypted to a block of ciphertext C by:

$$C= M^e \mod n$$

The plaintext block is then recovered by:

$$M= C^d \mod n$$

The written program consists of five classes, Cipher.java, Decipher.java and KeyScheduler.java for DES, RSA.java for RSA and finally Main.java, all Java files. They are all integrated so that they run as one program. The user chooses whether to encrypt or decrypt in DES or RSA. The results for the encryption and decryption are displayed on the Java console.

3- Simulation results

3.1 DES

For DES, an effective key size of 56 bits is used. Random text messages are encrypted and decrypted, and the results tabulated in Table 16. The computational execution time is shown. Fig.3 displays these results graphically.

Table 16 DES encryption and decryption time

Text size(bits)	Encryption (ms)	Decryption (ms)
128	29	42
256	33	50
512	47	57
1024	49	72
2048	54	74
4096	82	120
8192	144	152

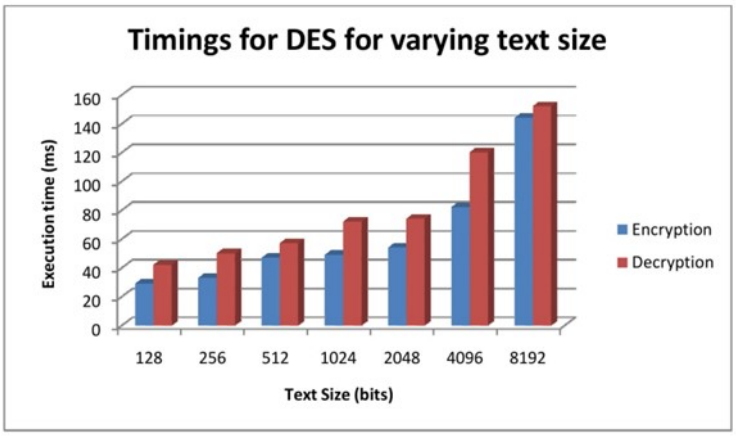

Fig.3 Encryption and decryption timing of DES

3.2 RSA

To evaluate RSA's performance, a variable encryption key size is used to encrypt and decrypt a message. A string message is composed of characters. Higher number of characters often violate the condition that the message *M* must lie in the interval [0, n – 1]. Each character in such a string is converted to its 3- character wide ASCII code, and the entire resulting numeric string is used as the message. The execution time is noted and recorded in Table 17. To view the results better, a graph of the execution time against the key length is drawn in Fig.4. In all cases, the same message is encrypted and decrypted.

Table 17 RSA encryption and decryption time

Key Length	Encryption(ms)	Decryption(ms)
512	54	90
768	65	210
1024	79	340
1280	90	447
1536	105	580
1792	116	743
2048	126	900

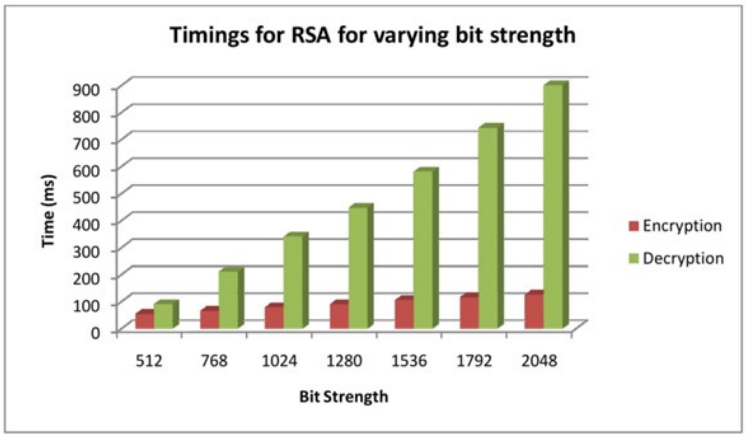

Fig.4 Encryption and decryption timing of RSA

4- Discussion

The performance of symmetric (DES) and asymmetric (RSA) key cryptosystems was studied. DES encrypted and decrypted data much faster than RSA. This is because RSA is based on mathematical functions which are computationally intensive, unlike DES which is mostly substitution of bits and thus not computationally intensive. DES is more suitable to the application which has decryption as the highest priority.

Although DES is faster in producing a cipher text and producing its equivalent plain text compared to RSA, it faces a major problem of key distribution. Asymmetric key cryptographic systems provide high security in all ways. For instance, RSA's security is based on the assumption that factoring a large number is hard. It would not be feasible to design an RSA cracker because the cryptanalyst is denied information on the sizes and values of p and q. These two values dictate how well RSA can handle a cryptanalyst during brute-force attack. The larger the product, the better the security of RSA. DES, on the other hand, is a standard, in that, the key must be 56 bits, message is encrypted in 64-bit blocks and so on. A DES cracker can be implemented in hardware which will try all the possible 2^{56} keys. DES is therefore useful only when the message to be transmitted is valid for a short time. The 'short' time is defined by how long a cryptanalyst with unlimited resources requires to break DES. For example, if a cryptanalyst can break DES in five hours, the message transmitted should not be valid for more than three hours. This can be useful in online transactions. Users should not be logged in with their credit card details to an online database for more than 10 minutes, for example. This gives

the cryptanalyst insufficient time to run through all possible keys encrypting the user's credit card details.

From the results in Table 16 and Fig.3, the encryption took a slightly lesser time than decryption. Since, for DES, the encryption and decryption are the same except that the keys are used in reverse order, these overhead can be attributed to the flipping of the keys. Another point to note is that the execution time increases exponentially with the text size.

From the results in Table 17, the RSA decryption time is much more than the encryption time. The encryption time takes significantly lesser time since the numbers involved are not as large as those used during decryption.

5- Conclusion

Two cryptographic algorithms (DES and RSA) have been explored and implemented in software. Performance comparisons were carried out and the parameter being tested was the execution time of the two algorithms. DES is fast especially in the decryption. RSA is slower than DES in both encryption and decryption and very processor intensive. Unfortunately, a dedicated hardware can be implemented as a DES cracker, since DES follows rules for encryption and decryption. There are 2^{56} DES keys so the cracker can be designed to search though the available keys with a bit of cleverness and exhaustive search in brute-force attack. On the other hand, RSA will resist a brute-force attack since the cryptanalyst will hit a snag when he is required to find the private key. Therefore RSA's security lies on the choice of p and q. Also if a message is to be padded, great care should be taken to avoid an attacker form forging a signature on valid messages.

References

[1]. N. Gichuki, "Comparative DES/RSA performance evaluation", Graduation Project, University of Nairobi, Kenya, 2009.

[2]. G. Chhabra, "Computer trend with security by RSA, DES and Blowfish algorithm", International Journal of Computer Science and Technology, Vol.4, Issue 2, PP.618-620, 2013.

[3]. P. Patil, "A Comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish", Procedia Computer Science, Vol.78, PP.617-624 , 2016.

[4]. A. Rao, "Survey paper comparing ECC with RSA, AES and Blowfish Algorithms", International Journal on Recent and Innovation Trends in Computing and Communication, Vol.5, Issue 1, PP.44-47, 2017.

[5]. S. Kaur, " Study of multi-level cryptography algorithm: Multi-Prime RSA and DES", International Journal Computer Network and Information Security, Vol.9, PP.22-29, 2017.