INFORMATION THEORY AND CODING

J-COMPONENT REVIEW - 3

"PERFOMANCE ANALYSIS OF EL-GAMAL & DES ALGORITHM "

Class Number (EPJ) : VL2021220502246
Slot : B1+TB1
Course Code : ECE4007
Faculty : Dr. K.S. Preetha

BATCH : 11

Sree Lalitha P.V.S - 19BEC0530
Aastha Singh - 19BEC0339
Nakshatra Rathore - 19BEC0299

Abstract:

As the pandemic continues, electronic communication such as e-mail, text messages is increasing day by day at a very fast pace, and hence the need to protect our information is the biggest challenge. Recently, Domino's Pizza had revealed that their database was hacked and sensitive information about customers such as e-mail, home address and phone number was revealed and sold on the dark web. Data security is the essential requirement while transmitting data over communication networks. To deal with this issue there are various techniques used to secure the data from illegal access throughout transmission. The solution of the above problem is to use data encryption techniques like cryptography, watermarking and steganography. Cryptography is a technique where we encrypt and decrypt the data to protect it from unauthorized access. There are several algorithms which are used to encrypt the data from illegal access like, Data Encryption Standard (DES), Advanced Encryption Standard (AES), RSA algorithm, etc. The objective of our project is to compare the performances of the DES(Symmetric) and the El-Gamal (Asymmetric) encryption algorithm in terms of time taken to encrypt, avalanche effect and time taken to decrypt.

Introduction & Literature Survey :

Encryption is critical for securing personally identifiable information and mitigating the threats for companies that perform payment transactions every minute of the day. This makes cryptography crucial. There are mainly two types of cryptography: symmetric and asymmetric cryptography.

Symmetric Key Cryptography, or Symmetric Encryption, uses a secret key for both encryption and decryption. This approach is the inverse of Asymmetric Encryption, which uses one key to encrypt and another to decrypt. Data is translated to a format that cannot be interpreted or inspected by someone who does not have the secret key used to encrypt it during this phase.

Year & Author	Paper Title	Methodology	Metric Analysed & Remarks
March 2019 & Rao, G. Mallikharjuna, and K. Deergha Rao	A Scheme for Latency Analysis of Different Cryptography Methods for Security in 5G Era	**Symmetric ciphers(shared secret key) :** AES, DES, Blowfish, IDEA; **Asymmetric ciphers(private & public key) :** RSA, ECC, DSA. Text message & audio signals are used for encryption & decryption using the above algorithms to analyse their performance. Audio signals are first converted into array of data and into strings before applying encryption algorithms.	**Metrics :** 1. Encrypt & decrypt time of text vs payload (both ciphers) 2. Encrypt & decrypt time of audio vs payload (both ciphers) 3. Latency **Remarks:** There is a trade off between latency & security in case of symmetric ciphers. In symmetric ciphers AES has high security, low latency & high speed and in asymmetric ciphers ECC have high security, high speed and low latency w.r.t both text and audio file. Hence these algorithms may be useful in 5G communication.
2017 Fang-Yu Rao	On the Security of a Variant of ElGamal Encryption Scheme	The paper is based on the comparison between the **ElGamal** and **Paillier Cryptosystem** and to prove that the latter is not as secure as it claims to be. A Variant on the Latter is created which is a distributed ElGamal Cryptosystem where the secret key is shared among 4 parties and the cipher text can be recovered if the 4 parties simultaneously decrypt it. However an attack from outside is feasible and the private key can be derived given the prime order of the underlying group	The main metric analysed is the Security and the efficiency of the variant of the ElGamal Cryptosystem. The New variant is definitely more efficient but it loses its purpose as the possibility of its ciphered texts being decrypted is openly possible with higher probability.

2021 W. Yihan and L. Yongzhen	Improved Design of DES Algorithm Based on Symmetric Encryption Algorithm	This paper addresses the issues in security posed to the **DES algorithm** by brute force cracking. It includes changing the block length from **64** to **128** bits and expanding the key length of the DES algorithm of double length. **Iterative** encryption is performed according to the new plaintext grouping and expanded key length of the two groups are iteratively encrypted. This paper includes a comparison between **DES**, **3DES**, and the **improved DES algorithm**(DESnew).	**Metrics :** Algorithm efficiency of DES, 3DES and DESnew is compared upto a run time of encrypted 1000K files/ms. **Remarks:** The improved **DES** betters three aspects of the algorithm which include increasing the packet length, key length and the exchange of each round of iterative results, thus improving efficiency and security. The efficiency observed here is much better than **3DES algorithm**.

Tools:

Python

Methodology:

Data Encryption Standard(DES) Algorithm: (Aastha Singh - 1BEC0339)

The DES (Data Encryption Standard) algorithm is a symmetric-key block cipher created in the early 1970s by an IBM team and adopted by the National Institute of Standards and Technology (NIST). The algorithm takes the plain text in 64-bit blocks and converts them into ciphertext using 48-bit keys.

Since it’s a symmetric-key algorithm, it employs the same key in both encrypting and decrypting the data. If it were an asymmetrical algorithm, it would use different keys for encryption and decryption.

Fig 1 : Encryption and decryption with DES

At the encryption side, DES takes 64 bit plain text and creates a 64-bit cipher text; at the decryption side, DES takes 64-bit cipher text and converts it back to 64-bit plain text. The same 56-bit cipher key is used for both encryption and decryption to generate 16 48-bit sub keys for each round.

There are four parts in DES algorithm, they are :

1. Initial permutation
2. Round
3. Final permutation
4. Key generation

Fig. 2 shows the overall structure of the DES algorithm. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions.The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to

produce the pre-output. Finally, the pre-output is passed through a permutation that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.

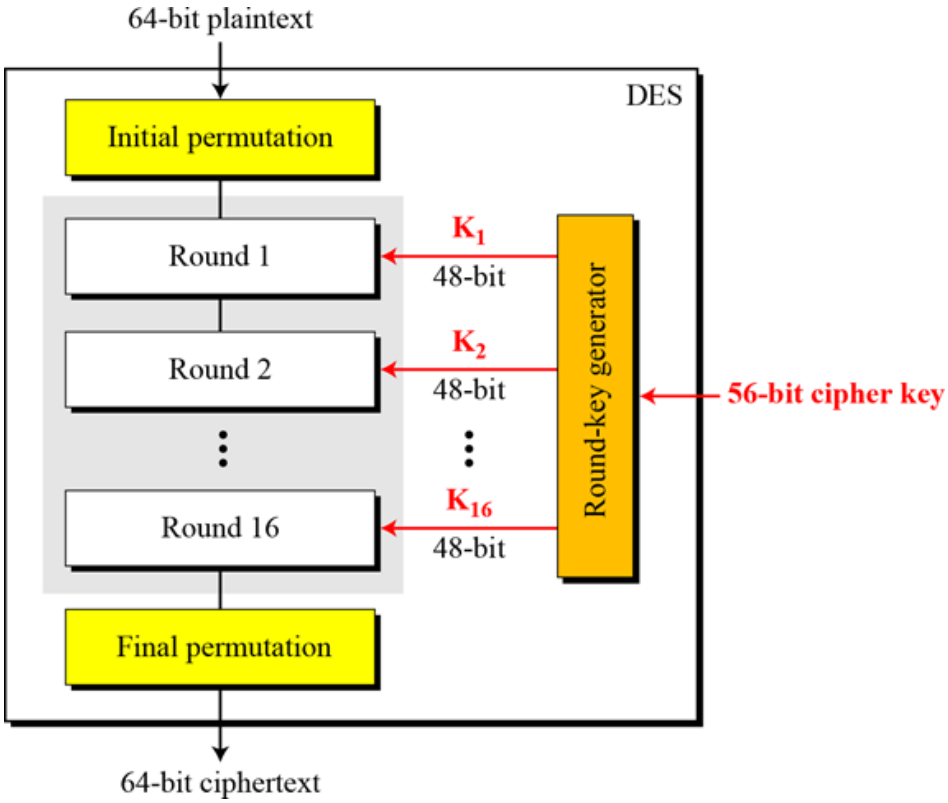


Fig. 2 : General structure of DES

Initial and Final Permutation :

In the first step, the 64 bit plain text block is handed over to an initial Permutation (IP) function. The initial permutation performed on plain text. Next the initial permutation (IP) produces two halves of the permuted block; says Left Plain Text (LPT) and Right Plain Text (RPT). Now each LPT and RPT go through 16 rounds of encryption. In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block. The result of this process produces 64 bit cipher text. The Initial permutation happens only once and it happens before the first round. And final permutation happens after the 16 rounds. Each of these permutations takes 64-bit input and permutes according to the predefined rule.

These permutations are keyless straight permutations and the inverse of each other. The following shows the initial and final permutation tables (P-boxes).

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

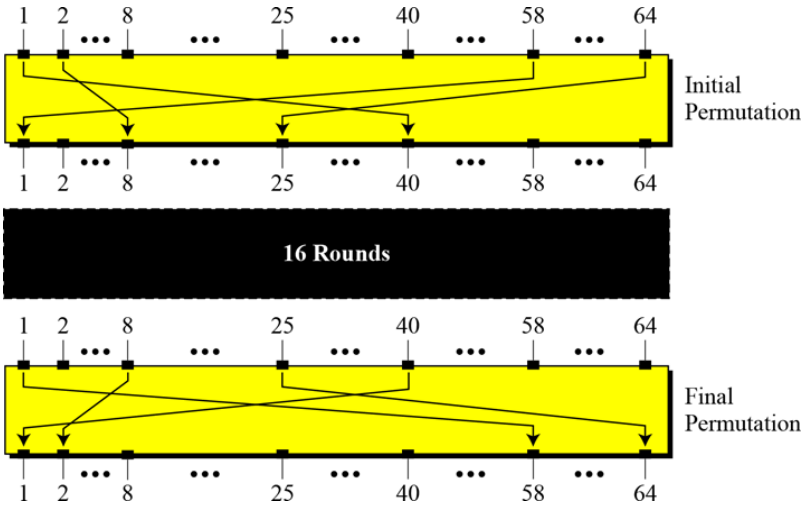


Fig. 3 : Initial and final permutation

Round :

DES uses 16 rounds. Each round of DES is a Feistel Cipher (It is a design model from which many different block ciphers are derived). There are different stages in a round.

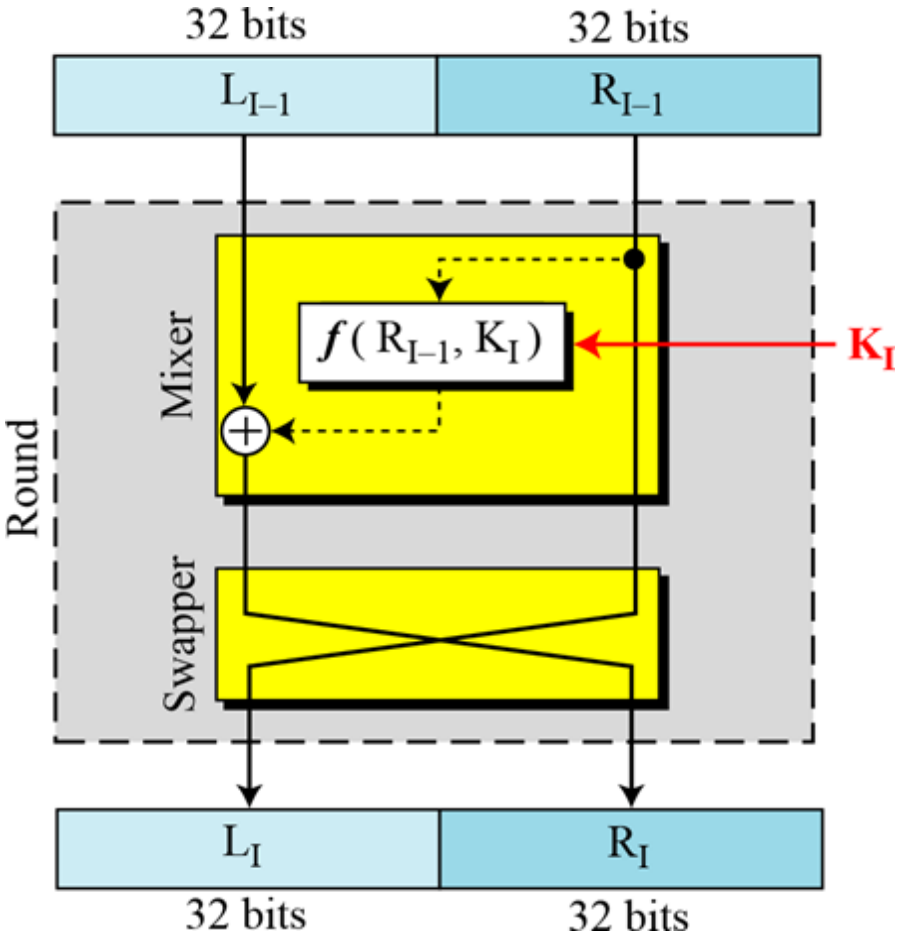


Fig 4 : A round in DES

A round takes left half 32-bits(L_{I-1}) and right half 32-bits (R_{I-1}) from the previous round or initial permutation box and creates new left 32-bits (L_I) and right 32-bits (R_I) which will go to the next round or final permutation box. The round consists of two cipher elements: mixer and swapper. Swapper swaps the left half and right half. Both mixer and swapper are invertible hence they can be used for decryption as well.

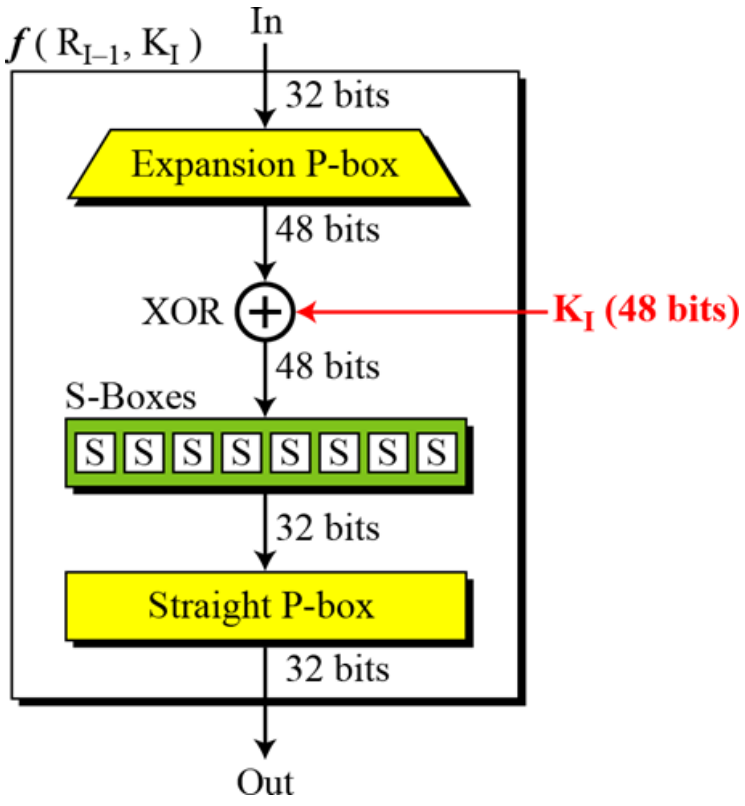


Fig 5 : Function f in Mixer

First the right half 32-bits (R_{I-1}) enters the expansion P-box. Since R_{I-1} is a 32-bit input and key K_I is 48-bits, we expand the 32 bits into 48 bits using expansion P-box. The following table is used for this purpose.

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

Fig 6 : Expansion P-box

Now both the round key and right half are 48-bits in length. XOR operation is performed between the round key and the expanded 48-bits (R_{I-1}). Then the result is passed into S-boxes (Substitution boxes) which helps in confusion. DES uses 8-boxes, each with 6-bit input and a 4-bit output. The 48-bit output from XOR is segregated into 8 groups each 6-bits and each group is fed into the box. The result of each box is a 4-bit chunk; when all the 8-boxes output are combined, we get 32 bit text. The substitution follows a predefined rule, where we take a group of 6-bits. The first and last bit will determine the row of S-box and the remaining bits in the center determine the column, the corresponding value in the table.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Fig. 7 : S-box 1

This is the S-box used for round 1, similarly there are 16 S-boxes for each round. The last operation performed in DES function is a straight permutation with 32-bit input and 32-bit output.

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Fig. 8 : Straight Permutation Table

The last operation in Mixer is performing XOR between L_{I-1} and the output from DES function. Once that is done, we have to pass it through the Swapper, where it

swaps left half and right half. Thus we can get L_I and R_I. This is used for the next round. Same operations happen for 16 rounds.

And the output from the 16th round will be swapped and sent to the final permutation box which will give 64-bit cipher text.

DES algorithm is a block cipher which means it can only take 64-bits at a go. So the input message should be divided into groups of 64-bits and for each group encryption should be performed. And at the receiver side, since DES is an invertible algorithm, decryption takes place and the original message is retrieved.

DES Decryption :

After all the substitutions, permutations, XORS, and shifting around, it might be thought that the decryption algorithm is completely different and just as confusing as the encryption algorithm. On the contrary, the various operations were chosen to produce a very useful property: The same algorithm works for both encryption and decryption.

With DES it is possible to use the same function to encrypt or decrypt a block The only difference is that the keys must be used in the reverse order. Lets take the encryption keys for each round are K_1, K_2, ... Kn then the decryption keys are Kn, ... K_2 , K_1**.** The algorithm that generates the key used for each round is circular as well. The key shift is a right shift and the number of positions shifted is 0,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1.

In the below figure we can see that the function for both DES encryption and decryption follow the same process. Except that, in encryption we use plain text as the input and cipher text as the output while in the decryption process cipher text is the input and plain text will be the output. And the keys used in each round of decryption is just the reverse of the order of keys used in the encryption.

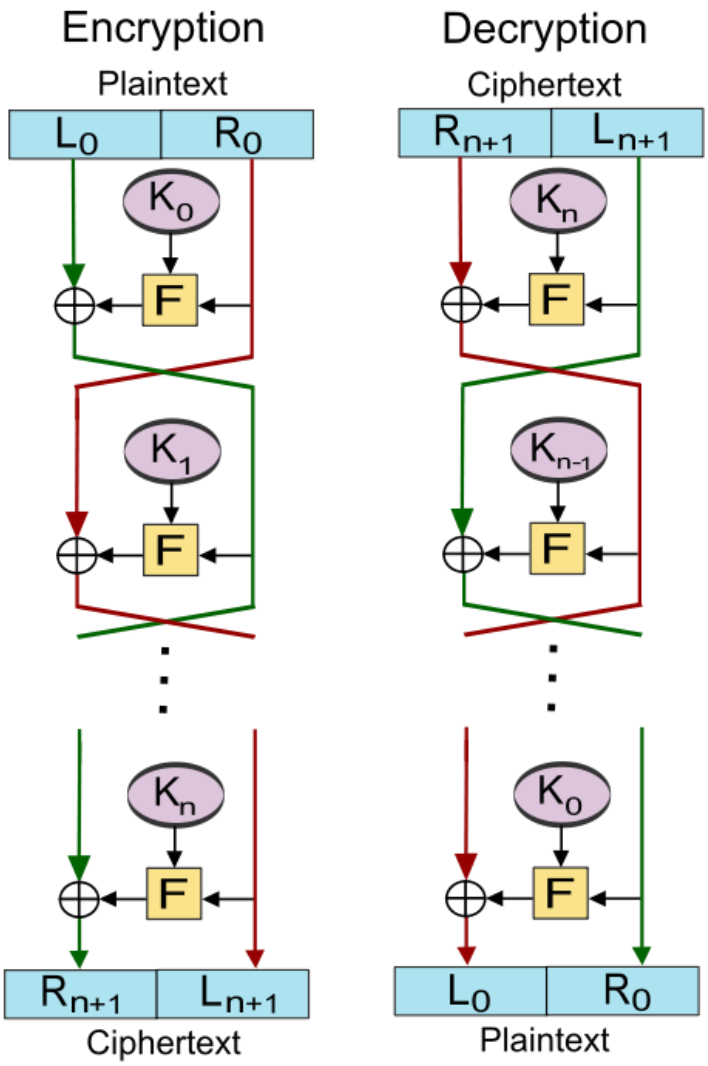


Fig 9 : DES Encryption and Decryption

As you can see, even though one half of the ciphertext is passed through the one-way function, there's always a copy of it remaining. In this case, Ln+1 goes through the one-way function to mask Rn+1, but still becomes one half of the ciphertext, which lets you use it again during decryption to undo the XOR operation you used to mask the other half of the ciphertext.

Then we keep doing that for each round (using the round keys in reverse) and we end up with the plaintext. No information is lost during the encryption process, the

one-way function is simply used to mask each half in turn in an interleaved fashion (which can be done again during decryption in the opposite direction, but only if you have the key).

Ultimately decryption is very similar to encryption, a common feature of Feistel ciphers in general. In fact with some arrangements the only difference is the order of the subkeys, which is (or at least was) a big advantage as it makes implementation easier on limited devices, as you can mostly reuse the encryption code for decryption.

DES Key Generation : (Nakshatra Rathore- 19BEC0299)

The issue in symmetric-key encipherment is the generation of a secure key. Different symmetric-key ciphers need keys of different sizes. The selection of the key must be based on a systematic approach to avoid a security leak.
For example, if Alice and Bob generate a session key between themselves, they need to choose the key so randomly that no one can guess the next key. If a key distribution centre needs to distribute the keys, the keys should be so random that no one should find similarities between two different keys. This implies that there is need for the concept of key-generation
The idea of key-generation is a subset of the cipher and reverse cipher concept. This involves the use of mixers and swappers eac having 16-rounds. There can be multiple approaches, but here we consider two concepts and plan to implement the latter.

First approach: The idea is to make the last round different from others as it has only a mixer and no swapper. The rounds are not aligned but the elements(mixer or swapper) are aligned.
Alternative approach : We can make all 16 rounds the same by including one swapper in the 16th round and adding an extra swapper after that as well.
The round-key generator creates sixteen 48 bit keys out of a 56-bit cipher key.

Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

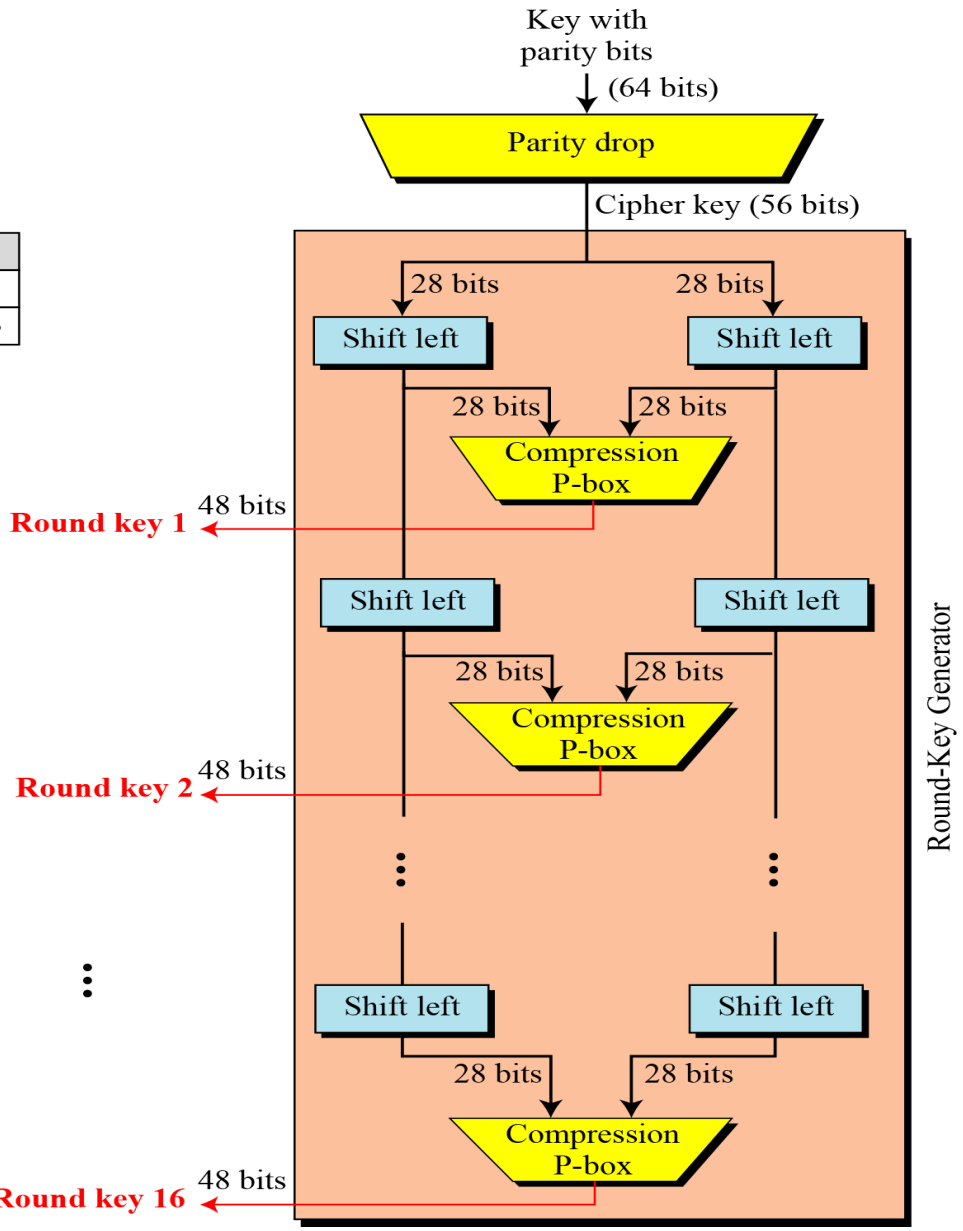


Fig. 9 : Key Generation

Parity Drop: The preprocess before key expansion is a compression transposition step that is popularly called the parity bit drop. It drops the parity bits(8,16,24,32,....64) from the 64-bit key and permutes the rest of the bits according to the table. The remaining 56-bit value is the actual cipher key which is used to generate key words.

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Fig. 10 : Parity bit drop table

Shift left: After straight permutation, the key is divided into two 28-bit parts. Each part is shifted left(circular) by one or two bits. In rounds 1,2,9 and 16, shifting is one bit, in the other rounds, it is two bits. The two parts are then combined to form a 56-bit part. The table below shows the number of shifts for each round.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Fig. 11 : Number of shifts

Compression D-box: The compression D-box changes the 58 bits to 48 bits which are used as a key for a round. The compression step is shown in the table below.

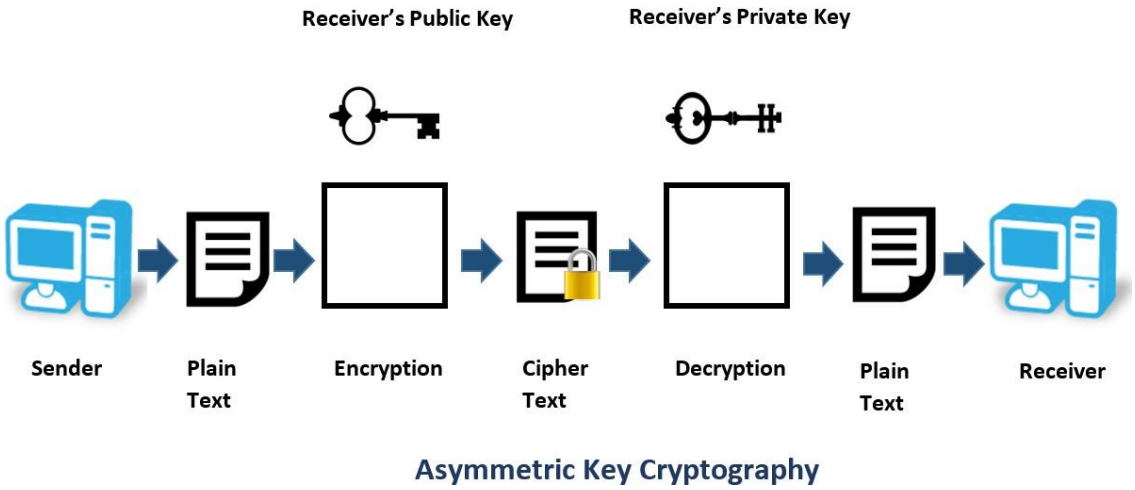
14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Fig. 12 : Key compression table

ElGamal Encryption : (Sree Lalitha P.V.S - 19BEC0530)

The ElGamal Encryption system is an asymmetric key encryption algorithm for public- key cryptography which is based on the Diffie-Hellman key exchange. Asymmetric cryptography, also known as public-key cryptography, is a process that uses a pair of related keys -- one public key and one private key -- to encrypt and decrypt a message and protect it from unauthorized access or use.Asymmetric encryption uses a mathematically related pair of keys for encryption and decryption: a public key and a private key. If the public key is used for encryption, then the related private key is used for decryption. If the private key is used for encryption, then the related public key is used for decryption.

Asymmetric cryptography is typically used to authenticate data using digital signatures. A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document. It is the digital equivalent of a handwritten signature or stamped seal.



Asymmetric Key Cryptography

Diffie-Hellman Key Exchange :

It is a very famous key exchange algorithm that is used by many commercial products of today. The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values. The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. The primary process begins with a Primitive root and a Prime number. A primitive root of a prime number is one whose powers modulo generate all the integers from 1 to (p-1) . That is, if a is a primitive root of the prime number p , then the numbers:

a modp, a^2modp…… a^{n-1}modp

are distinct. For any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

b = a^i(modp)

The exponent i is referred to as the **discrete logarithm** of b for the base a , mod p.

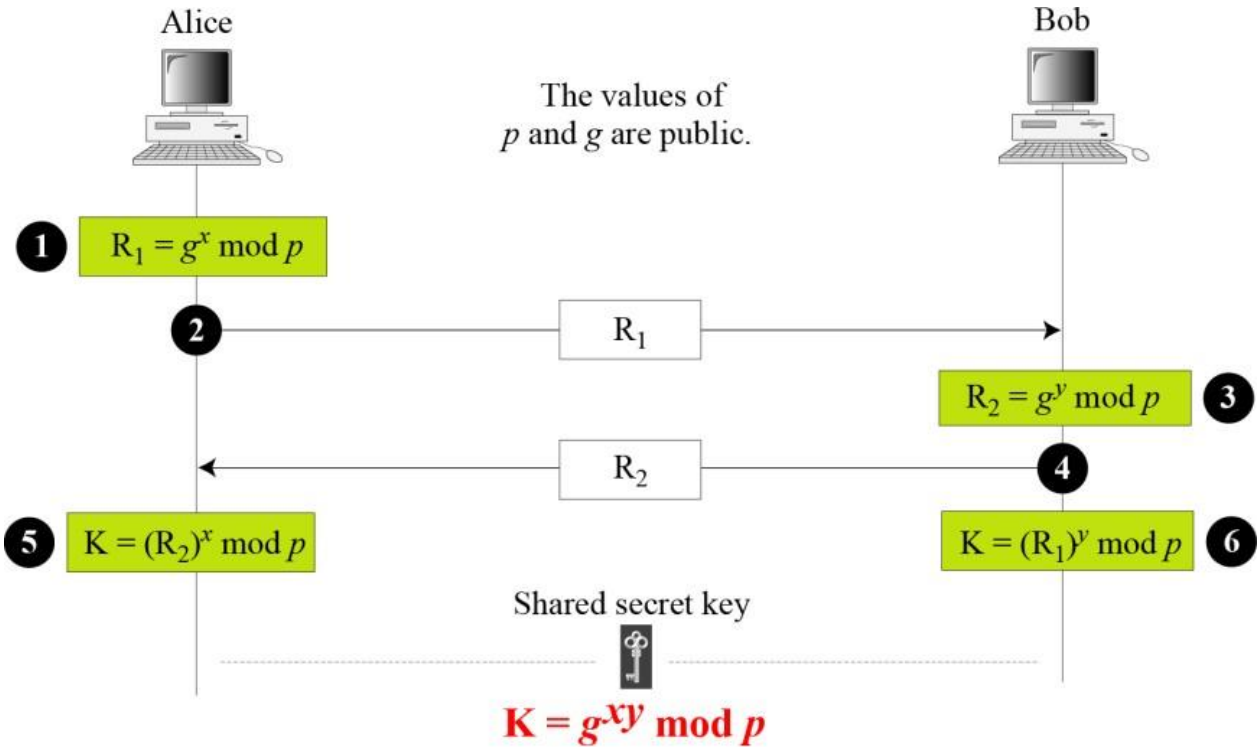


Figure 13: Diffie-Hellman Process

In the Diffie-Hellman process, the sender and receiver have a public and private key. The public key along with the generator(primitive root) and the prime number are shared in the insecure channel. A Random number is taken which is not beyond the prime number and this becomes the private key. The private key, generator and prime number are used to create the public key. The prime number is usually very large and due to the implications of discrete logarithms, it becomes virtually impossible to get the private key even if a third party candidate obtains the other three variables. This is what makes the Diffie-Hellman Key exchange very useful.

To summarize, there are two publicly known numbers: a prime number q and an integer α that is a primitive root of q . Suppose the users A and B wish to exchange a key. User A selects a random integer X_A < q and computes Y_A = α^{XA} (modq). Similarly, user B independently selects a random integer X_B< q and computes Y_B =α^{XB} (modq) . Each side keeps the X value private and B = α^{XB} makes the Y value available publicly to the other side. User A computes the key as K=$Y_B{}^{XA}$ (modq) and user B computes the key as K= $Y_A{}^{XB}$ (modq). By calculation, we can find out that these two calculations produce the same

value of K. The result is that the two sides have exchanged a secret value. Furthermore, because X_A and X_B are private, an adversary only has the following ingredients to work with: q, α, Y_A,Y_B and .Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

$X = \mathrm{d}\ log_{\alpha,q}(Y_B)$

The adversary can then calculate the key in the same manner as user B calculates it.With larger numbers, the problem becomes impractical and the private keys cant be determined using brute force.

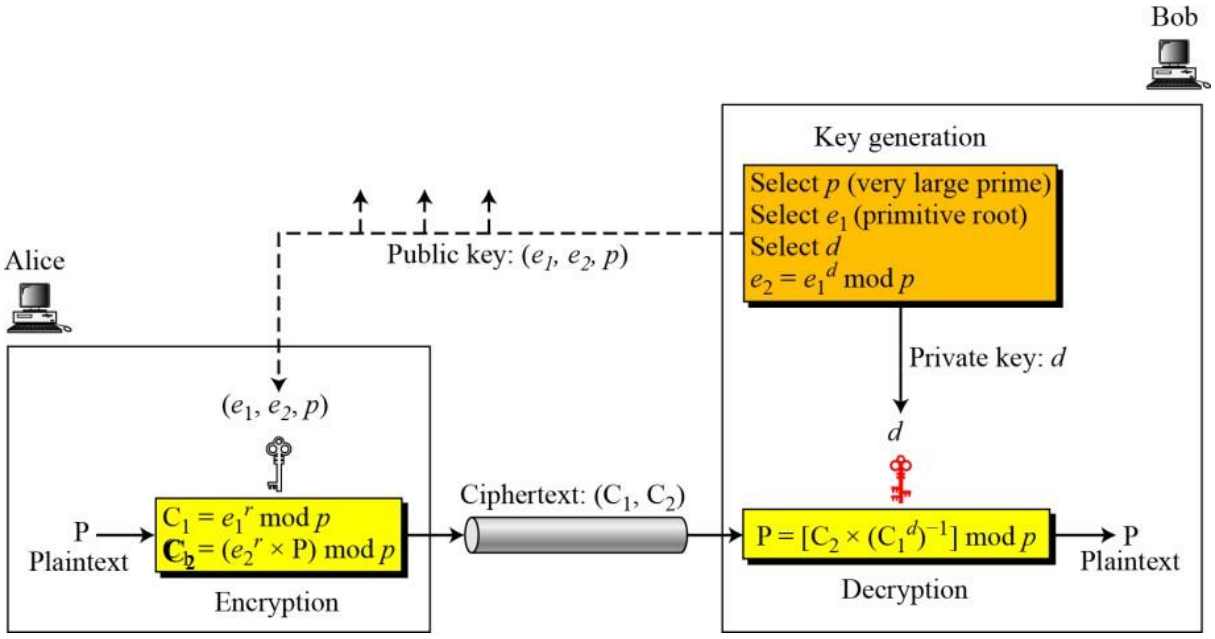


Figure 14: ElGamal Encryption Process

The ElGamal Encryption Algorithm very closely follows the Diffie-Hellman Key Exchange. We will now see the Summary of the Encryption and Decryption process.

Encryption:

1. Generate a random integer X_A , such that $1< X_A < q-1$.
2. Compute $Y_A = \alpha^{XA} (\text{mod} q)$.
3. A's private key is X_A ; A's pubic key is { q,α, Y_A } .

Any user B that has access to A's public key can encrypt a message as follows:

1. Represent the message as an integer M in the range $0 \leq M \leq q-1$. Longer messages are sent as a sequence of blocks, with each block being an integer less than q .
2. Choose a random integer k such that $1 \leq k \leq q-1$.
3. Compute a one-time key $K=Y_A^k \pmod q$.
4. Encrypt M as the pair of integers (C1,C2) where
   $C_1 = \alpha^k mod\ q$; $C_2 = KM mod\ q$

Decryption:

User A recovers the plaintext as follows:

1. Recover the key by computing $K=(C_1)^{XA} \bmod q$.
2. Compute $M = (C_2 K^{-1}) mod\ q$.

Results and Discussion :

In this project the working of DES and El Gamal was studied thoroughly. It includes going through literature reviews and understanding the working of these algorithms. El gamal is an asymmetric algorithm while on the other hand DES is a symmetric algorithm. The codes were written in Python3 using Google Collab. Here are respective output images.

```
EL-GAMAL Encryption
Original Message : ITC review 3
g used :  116737839886797988499315218206963671738714467082 9
g^a used :  167190221008506698658144261115682592802268000744 9
g^k used :  167190221008506698658144261115682592802268000744 9
g^ak used :  364699233847077937122443638998085206481516598649
time taken for encryption  0.0007959810000102152 seconds
time taken for decryption  0.0001606709997759026 seconds
Decrypted Message : ITC review 3
```

Fig: *El Gamal Encryption and Decryption*

```
DES Encryption
After initial permutation 14A7D67818CA18AD
Round  1   18CA18AD   5A78E394   194CD072DE8C
Round  2   5A78E394   4A1210F6   4568581ABCCE
Round  3   4A1210F6   B8089591   06EDA4ACF5B5
Round  4   B8089591   236779C2   DA2D032B6EE3
Round  5   236779C2   A15A4B87   69A629FEC913
Round  6   A15A4B87   2E8F9C65   C1948E87475E
Round  7   2E8F9C65   A9FC20A3   708AD2DDB3C0
Round  8   A9FC20A3   308BEE97   34F822F0C66D
Round  9   308BEE97   10AF9D37   84BB4473DCCC
Round  10   10AF9D37   6CA6CB20   02765708B5BF
Round  11   6CA6CB20   FF3C485F   6D5560AF7CA5
Round  12   FF3C485F   22A5963B   C2C1E96A4BF3
Round  13   22A5963B   387CCDAA   99C31397C91F
Round  14   387CCDAA   BD2DD2AB   251B8BC717D0
Round  15   BD2DD2AB   CF26B472   3330C5D9A36D
Round  16   19BA9212   CF26B472   181C5D75C66D
Cipher Text :  C0B7A8D05F3A829C
Time taken for encryption  0.015189467999789485 seconds
```

Fig: *DES Encryption*

```
DES Decryption
After initial permutation 19BA9212CF26B472
Round  1   CF26B472   BD2DD2AB   181C5D75C66D
Round  2   BD2DD2AB   387CCDAA   3330C5D9A36D
Round  3   387CCDAA   22A5963B   251B8BC717D0
Round  4   22A5963B   FF3C485F   99C31397C91F
Round  5   FF3C485F   6CA6CB20   C2C1E96A4BF3
Round  6   6CA6CB20   10AF9D37   6D5560AF7CA5
Round  7   10AF9D37   308BEE97   02765708B5BF
Round  8   308BEE97   A9FC20A3   84BB4473DCCC
Round  9   A9FC20A3   2E8F9C65   34F822F0C66D
Round  10   2E8F9C65   A15A4B87   708AD2DDB3C0
Round  11   A15A4B87   236779C2   C1948E87475E
Round  12   236779C2   B8089591   69A629FEC913
Round  13   B8089591   4A1210F6   DA2D032B6EE3
Round  14   4A1210F6   5A78E394   06EDA4ACF5B5
Round  15   5A78E394   18CA18AD   4568581ABCCE
Round  16   14A7D678   18CA18AD   194CD072DE8C
Plain Text :  123456ABCD132536
Time taken for decryption :  0.0034024470000986184 seconds
```

Fig: *DES Decryption*

The aim of this project was to study and compare the two algorithms in order to determine which algorithm is more efficient and safer based on the performance analysis. The performance analysis involves a comparison of encryption and decryption time of both algorithms as well as a comparison of their key lengths as well. This helps us understand more about the efficiency and the security of these algorithms.

	DES	El Gamal
Encryption Time	0.01518 seconds	0.00079 seconds
Decryption Time	0.00340 seconds	0.00016 seconds
Key Length	16 Hexadecimals	48 integers

Fig: *Performance Analysis*

The metrics measured in this project revolve around the performance analysis of both these algorithms. First we compare the efficiency of both these algorithms. In the table above we can observe that both the Encryption and Decryption time is much lesser in El Gamal compared to DES. This is primarily due to the fact that DES is a more powerful Algorithm with various layers.
Hence El Gamal is more efficient in terms of the time taken to run these algorithms.

Security:

The second aspect of the performance analysis revolves around the security of the algorithm. One important factor is the key length used. The key length can determine the safety of the algorithm to a certain extent. A short cipher key is a weakness for the DES algorithm and makes it fragile against brute force attacks. In these research papers, all the newer versions of the DES algorithm like 3DES were an improvement in aspects such as packet length and key length with certain changes to the iterative rounds as well. The 3DES version makes the algorithm resistant to brute force attacks.
DES is known to be not immune to differential cryptanalysis but the S-box is specifically designed to be resistant to this kind of attack. When we have 247 chosen plaintexts or 255 known plaintexts, differential cryptanalysis can be used to break DES. Although technique appears to be more efficient than a brute-force approach, it is impractical to find 247 chosen plaintexts or 255 known plaintexts. As a result, we can conclude that DES is immune to differential cryptanalysis.

Avalanche effect:

It is a desirable property of any encryption algorithm that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext. We checked the output after making changes to the plain text to see if this property was being exhibited.

DES algorithm :

First we observe results for a particular set of plain text and cipher key then introduce minute changes in the plain text to observe the avalanche effect.

```
pt = "124456ABCD132536"
key = "AABB09182736CCDD"
```

```
Cipher Text :  9D6DD3D034BC9DEE
Time taken for encryption  0.007592658000248775 seconds
```

Fig: *Initial plain text, key and Cipher text*

We change the very first letter/numeric of the plain text.

```
pt = "224456ABCD132536"
key = "AABB09182736CCDD"
```

```
Cipher Text :  4FD3761E73B9D40B
Time taken for encryption  0.008377659000871063 seconds
```

Fig: *New plain text, key and Cipher text*

We observe here that just by changing a single letter we were able to generate an entirely different cipher key. The Avalanche effect increases the security and credibility of the DES algorithm even more.

References:

1. Rao, G. Mallikharjuna, and K. Deergha Rao. "A Scheme for Latency Analysis of Different Cryptography Methods for Security in 5G Era." In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, pp. 1-5. IEEE, 2019.
2. F. Rao, "On the Security of a Variant of ElGamal Encryption Scheme," in *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 725-728, 1 July-Aug. 2019, doi: 10.1109/TDSC.2017.2707085.
3. W. Yihan and L. Yongzhen, "Improved Design of DES Algorithm Based on Symmetric Encryption Algorithm," 2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA), 2021, pp. 220-223, doi: 10.1109/ICPECA51329.2021.9362619.