RV COLLEGE OF ENGINEERING®
BENGALURU – 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

"Discrete Mathematics in Network Security Protocols"

Experiential Learning REPORT
Discrete Mathematical Structures (18CS36)
III SEMESTER

2021-2022

Submitted by

Malavika Hariprasad	1RV20CS080
Neha N	1RV20CS094
Nimisha Dey	1RV20CS098
Pratiksha Narasimha Nayak G	1RV20CS123

Under the Guidance of

Anitha Sandeep,
Assistant Professor
Department of CSE, RVCE,
Bengaluru - 560059

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the **Experiential Learning** work titled “Discrete Mathematics in Network Security Protocols” has been carried out by **Malavika Hariprasad (1RV20CS080), Neha N (1RV20CS094), Nimisha Dey (1RV20CS098), Pratiksha Narasimha Nayak G (1RV20CS123),** bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Assessment of Course: Discrete Mathematical Structures (18CS36) – Experiential Learning** during the year 2021-2022. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report.

Faculty Incharge
Department of CSE,
RVCE., Bengaluru –59

Head of Department
Department of CSE,
RVCE, Bengaluru–59

ACKNOWLEDGEMENT

It gives us immense pleasure to be associated with this project titled “Discrete Mathematics in Network Security Protocols”. The project was a joyous learning process that helped us in linking our theoretical knowledge to practical knowledge. We got to learn a lot from this project and gain in depth knowledge about the implementation and practical application of Discrete Mathematics.

Firstly, we would like to express our sincere gratitude to our teacher Prof. Anitha Sandeep as well as our Principal Dr. K N Subramanya who gave us the golden opportunity to do this wonderful project. I extend my heartfelt thanks to our teacher who has helped us in this endeavor and has always been very cooperative in providing us with the necessary information regarding the project.

Secondly, we would like to thank our family and friends for providing us with unfailing support and continuous encouragement through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Last but not the least, we would like to thank all those who had helped us (directly or indirectly) towards the completion of this project within the limited time frame. It was a great learning experience for us.

ABSTRACT

The modern world runs on a vast amount of data transfer between millions of interconnected nodes and systems. Much of the information is normally stored in remote cloud servers. Interaction with this data happens constantly via web connections. Hackers may be able to breach this data either when it is stored or in transit if it is not adequately encrypted to ensure data privacy. Data encryption refers to the process that ensures all this data is effectively protected. Encryption 'scrambles' the readable data using a secret code to ensure that only the intended recipient is able to read the data, thereby making it difficult to reconstitute for any unintended third-party who might breach the information.

Various encryption algorithms, symmetric and asymmetric, have been developed through decades aiming to provide better security. The AES encryption algorithm borrows ideas of Galois fields(or finite fields), confusion, and diffusion. Confusion is employed for making uninformed cipher text whereas diffusion is employed for increasing the redundancy of the plain text over the foremost part of the ciphertext to make it obscure. A symmetric key cipher has its own several problems. One of these is that, somehow, two people who want to use such a system must secretly agree on a shared key. This is quite difficult and is wholly impractical if there is a whole network of people who need to communicate. However, in the late 1970s, several people came up with remarkable new ways to solve this key distribution problem. This allows two people to publicly exchange information that leads to a shared secret without anyone else being able to figure out the secret. One of them, the RSA encryption algorithm, looks back to the work done in the 1760s by a Swiss mathematician, Leonard Euler. Euler had studied the distribution of prime numbers and proposed the phi function (also called the totient) which tells about the breakability of a number. This phi function is the protagonist of RSA encryption. Another one, the Diffie-Hellman key exchange, uses properties of cyclic groups as well as the modular arithmetic. It enables two parties communicating over a public channel to establish a mutual secret without it being transmitted over the Internet. Since each of these types of encryption algorithms had its own drawbacks, the idea of integrating symmetric and asymmetric encryption algorithms has been supported.

This project has provided a basic model of various cryptographic algorithms by familiarizing various discrete mathematical concepts and their implementation using C++ compiler and environment such as MATLAB. Cryptography has been a particularly interesting field because of the amount of work that is, by necessity, done in secret. However, secrecy is not the key to the goodness of a cryptographic algorithm. Any cryptographic scheme that stays in use year after year is most likely a good one. The strength of cryptography lies in the choice and management of the keys; longer keys resist attack better than shorter keys.

Table of Contents

List of Figures

List of Tables

GLOSSARY

AES	:	Advanced Encryption Standard
DES	:	Data Encryption Standard
FTP	:	File Transfer Protocol
HTTP	:	HyperText Transfer Protocol
IP	:	Internet Protocol
MAC	:	Media Access Control
MI	:	Multiplicative Inverse
NIST	:	National Institute for Standards and Technology
OSI	:	Open System Interconnection
PII	:	Personal Identifying Information
RSA	:	Rivest Shamir Adleman
SMTP	:	Simple Mail Transfer Protocol
SRS	:	Software Requirement Specification
TCP	:	Transmission Control Protocol
TLS	:	Transport Layer Security
UDP	:	User Datagram Protocol

Chapter 1

Introduction

This section gives a brief introduction to the OSI model that describes various network protocols with special emphasis on the working of the TLS layer and the different encryption algorithms that can be adopted. This section also covers the discrete mathematics concepts that are used in the implementation of encryption algorithms used in network protocols like Modular arithmetic and Group Theory. It provides a detailed explanation of the problem statement and objectives of this project along with the methodology and scope followed in order to achieve the objectives of the project.

The OSI Model

The OSI model is a framework that describes how various network protocols interact at various levels to run the communication between a client and a server.
The OSI model, as shown in Fig 1.1, mainly comprises of the following 7 layers:

1. Application layer: This is where the user interacts with the web browser for requesting data from the server. Protocols such as HTTP, SMTP, FTP work.
2. Presentation layer: This is where the data is translated from one format to another.
3. Session layer: A session refers to the time period for which the user interacts with the server. The session layer is responsible for establishing these sessions. It also maintains ‘checkpoints’ to synchronize the transfer of data packets.
4. Transport layer: This layer consists of protocols such as TCP and UDP. Here, the data is divided into chunks called packets.
5. Network layer: This layer is responsible for assigning the IP addresses of the source and the destination to each of the data packets. Its duty is also to find the best possible path to transfer these packets.
6. Data link layer: This ensures that the data packets reach the correct node based on the MAC address on them.
7. Physical layer: This is the direct medium through which data transfer takes place in bits. Eg: Optical fiber.

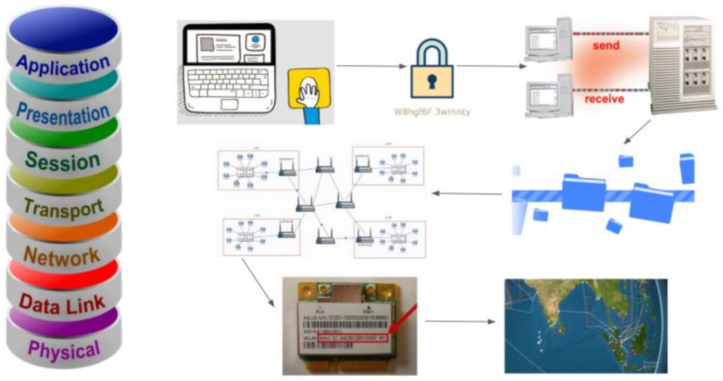


Fig 1.1 The OSI Model

Encryption

In today's cyber-world, there is an ever-present risk of unauthorized access to all forms of data. Most at risk is financial and payment system data that can expose the PII or payment card details of customers and clients. Encryption protects personal and sensitive data, and hence, enhances the security of communication between the client apps and the servers.
Encryption is the process of converting a plain text (message) into a cipher (unreadable or scrambled) text using various mathematical formulae as algorithms.
Special rules or relations, called keys, indicate the relationship between the original and the encrypted texts. They are usually a string of numbers measured in bits.

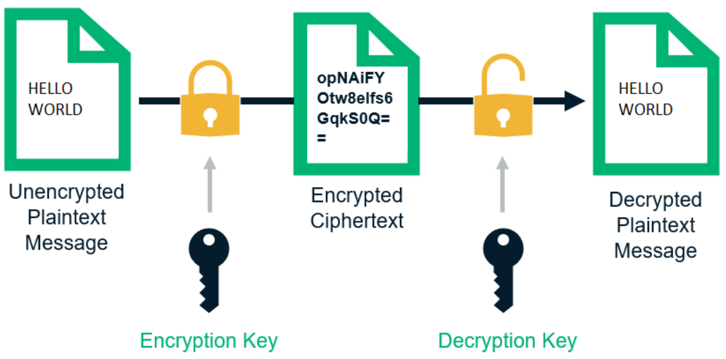
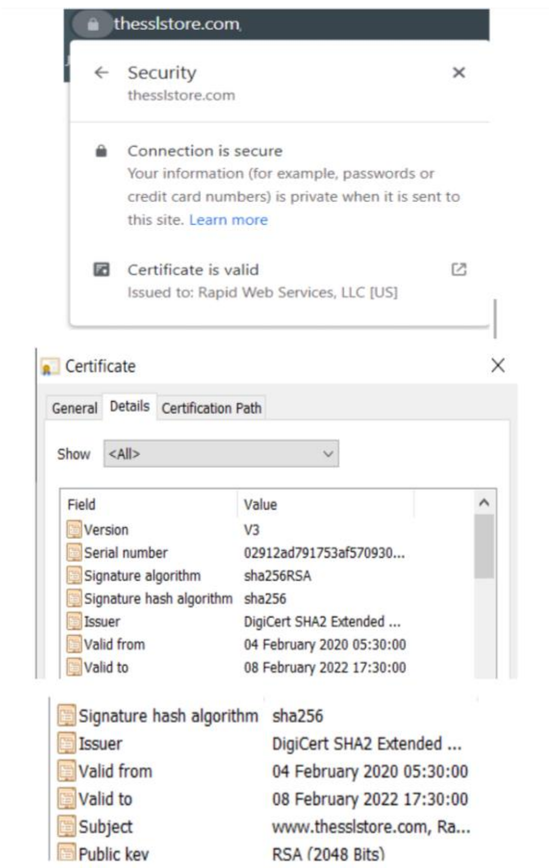

HELLO WORLD
Unencrypted Plaintext Message
opNAiFY Otw8elfs6 GqkS0Q= =
Encrypted Ciphertext
HELLO WORLD
Decrypted Plaintext Message
Encryption Key
Decryption Key

Fig 1.2 How encryption works

Working:
An encryption key is used to encrypt the message to be sent. At the other end, a message is perceived using a decryption key, as shown in Fig 1.2.

A protocol , called TLS, works at the encryption level to provide security in the communication between the client and the server.
TLS Handshake:
Step 1: Client 'Hello' - The client sends a 'Hello' message consisting of its public key.
Step 2: Server 'Hello' - The server sends its TLS certificates and its public key to the server.
Step 3: The client creates a 'Pre-master Secret' using the server's public key and a session key (later, shared key) and sends it to the server.
Step 4: The server, who has the private key, is able to decrypt the 'Pre-master Secret'.
Step 5: The server sends an acknowledgement message to the client with the help of the session key which it perceived in the previous step.
Step 6: Now, both the client and the server communicate using the shared (or session) key.

Fig 1.3 illustrates details such as the encryption algorithm implemented, the size of the key used, etc in the TLS connection between the client and the server of 'sslstore'.



thesslstore.com

Security
thesslstore.com

Connection is secure
Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more

Certificate is valid
Issued to: Rapid Web Services, LLC [US]

Certificate

General | Details | Certification Path

Show <All>

Field	Value
Version	V3
Serial number	02912ad791753af570930...
Signature algorithm	sha256RSA
Signature hash algorithm	sha256
Issuer	DigiCert SHA2 Extended ...
Valid from	04 February 2020 05:30:00
Valid to	08 February 2022 17:30:00

Signature hash algorithm	sha256
Issuer	DigiCert SHA2 Extended ...
Valid from	04 February 2020 05:30:00
Valid to	08 February 2022 17:30:00
Subject	www.thesslstore.com, Ra...
Public key	RSA (2048 Bits)

Fig 1.3 TLS connection details

Symmetric Encryption: In symmetric encryption, as shown in Fig 1.4, there is only one key, and all parties involved use the same key to encrypt and decrypt information.

- Drawback: The connection isn't secure when the shared key is exchanged. Hence, any third party on the network could grab the shred key and listen to the encrypted message.

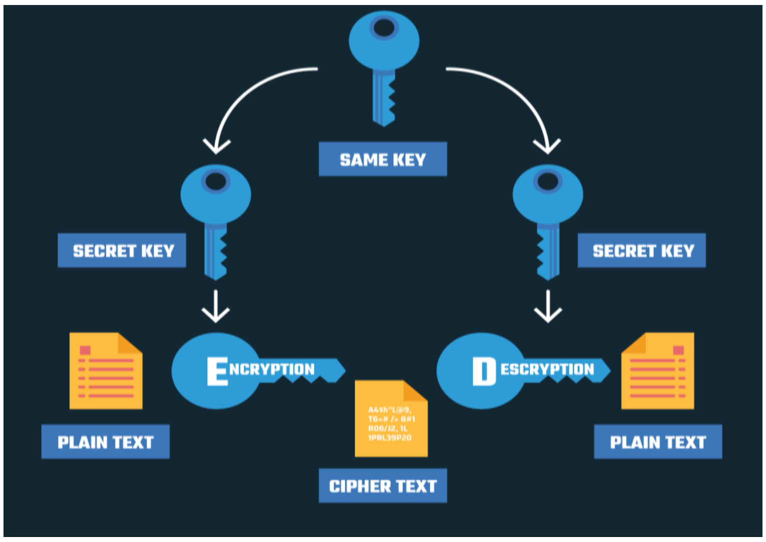


Fig 1.4 Symmetric Encryption

Asymmetric Encryption: In Asymmetric encryption, as shown in Fig 1.5, one key is made public so that anyone who wants to send the receiver any data, could use it for encryption, while the second key is kept private as the receiver will use it for decryption.

- It ensures that one sender wouldn't be able to read the messages sent by another sender, even though they both have the receiver's public key.
- It's also close to impossible to find out which private key belongs to which public key.

- Drawbacks: It takes more computing resources and time to encrypt and decrypt information.
    - Its keys have to be longer to provide the same level of security that symmetric encryption does.

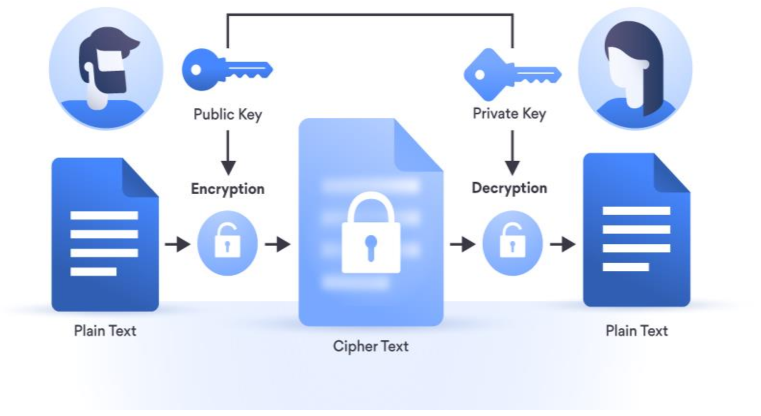


Fig 1.5 Asymmetric Encryption

1.1 Discrete Mathematical Concepts

1.1.1 Modular Arithmetic

Modular arithmetic is the system of arithmetic for integers which involves expressing numbers in terms of their remainder.

If A/B = Q and remainder is R then, R = A mod B

The 4 basic modular arithmetic operations are:
i)Modular addition: (A + B) mod C = (A mod C + B mod C) mod C
ii)Modular subtraction: (A - B) mod C = (A mod C - B mod C) mod C
iii)Modular multiplication: (A * B) mod C = (A mod C * B mod C) mod C
iv)Modular exponentiation: A^B mod C = ((A mod C)^B) mod C

In modular arithmetic there is no division operation, however modular inverse exists which involves the concept of modular congruence.

Congruence Modulo
For a positive integer n, the integers a and b are congruent mod n if their remainder when divided

by n are the same.

i.e if a/n = q1 remainder r and b/n = q2 remainder r then, a ≡ b(mod n)

a ≡ b(mod n) can also be written as

- a mod n = b mod n
- n | (a-b) (n divides a -b)
- a = k.n + b

Congruence modulo is an equivalence relation

- a≡a (mod n) (Reflexive)
- If a≡b (mod n) then If b≡a (mod n) (Symmetric)
- If a≡b (mod n) and If b≡d (mod n) then If a≡d(mod n) (Transitive)

Modular inverse

•The modular inverse of A (mod C) is A^-1
•(A * A^−1) ≡ 1 (mod C) or equivalently (A * A^−1) mod C = 1
•Only the numbers coprime to C (numbers that share no prime factors with C) have a modular inverse (mod C)

1.1.2 Euler's theorem

Fermat's Little theorem
If p is a prime number, then for any integer a, the number a p – a is an integer multiple of p.
Mathematically, $a^{p-1} \equiv 1 \pmod{p}$

Euler's Totient function
Euler's Totient function Φ (n) for an input n is the count of numbers in {1, 2, 3, …, n} that are relatively prime to n

Properties of Euler's Totient function

•For a prime number p, Φ (p) = p-1
•For 2 prime numbers a and b Φ (a.b) = Φ (a). Φ (b) = (a-1)(b-1)

Euler's theorem
Euler's theorem states that if n and a are coprime (or relatively prime) positive integers, then
$a^{\Phi(n)} \equiv 1 \pmod{n}$

1.1.3 Group Theory

Group Theory is a branch of mathematics and abstract algebra that defines an algebraic structure named as group. Generally, a group comprises a set of elements and an operation over any two elements on that set to form a third element also in that set (i.e., G×G→G). This operation is called a binary operation.

A group holds four properties simultaneously -

i) Closure

ii) Associative

iii) Identity element

iv) Inverse element

1.1.4 Cyclic Groups

A cyclic group is a group that can be generated by a single element. They are always abelian. Abelian groups are groups which follow the commutative property additionally along with the other properties. Every element of a cyclic group is a power of some specific element which is called a generator. A cyclic group can be generated by a generator 'g', such that every other element of the group can be written as a power of the generator 'g'. Consider the example of the group on the set {1,2,3,4} on the operation 'mod 5'. We find that:

21 mod 5=2;   22 mod 5=4;   23 mod 5=3;   24 mod 5=1

It can be noticed that the powers of 2 cycle through all the elements in the set chosen i.e., 1,2,3 and 4. Hence we can conclude that this mod 5 group is cyclic and that 2 is a generator of this group.

1.2 Proposed System

1.2.1 Problem statement and objectives

Problem statement
Transfer of information and data over the internet is vulnerable to various security threats and unauthorized access. Transmission of confidential and sensitive information among different networks is possible without compromise by using various encryption techniques.

Objectives
Through this project we aim to learn about various encryption techniques, our objectives being

- To implement AES, RSA and Diffie Hellman encryption algorithms.
- To understand the use of discrete mathematical concepts such as Modular Arithmetic and Group theory in network protocols.

1.2.2 Methodology

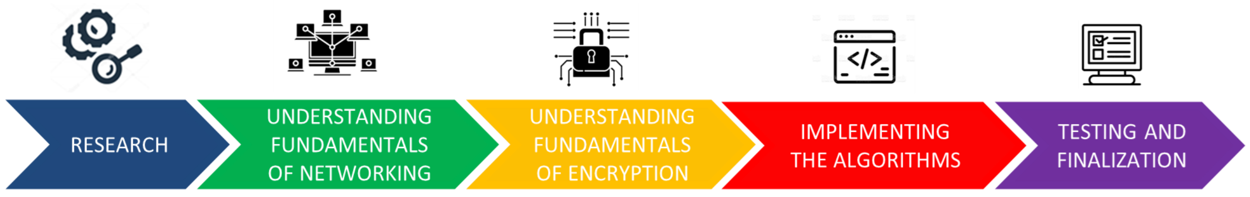

RESEARCH
UNDERSTANDING FUNDAMENTALS OF NETWORKING
UNDERSTANDING FUNDAMENTALS OF ENCRYPTION
IMPLEMENTING THE ALGORITHMS
TESTING AND FINALIZATION

Fig 1.6: Methodology followed

Fig 1.6 depicts the methodology followed while approaching the project. First, research was conducted to understand the current trends in the field of network protocols and encryption following which the scope and objectives of the project were defined. After this, we understood the fundamentals of network protocols, namely TLS encryption. We then studied the encryption algorithms covered in the paper, i.e., RSA, Diffie Hellman and AES. Following this, we implemented these algorithms by developing relevant programs to simulate and demonstrate the working of these algorithms, specifically in TLS.

1.2.3 Scope

Encryption algorithms, as shown in Fig 1.7, are chiefly classified as Symmetric and Asymmetric algorithms. DES and AES are symmetric encryption algorithms, while RSA and Diffie-Hellman are asymmetric encryption algorithms.

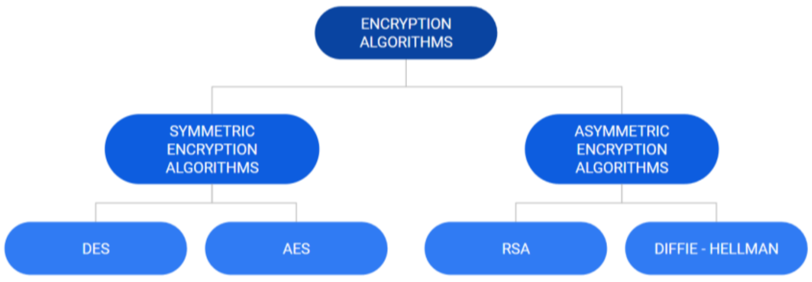

ENCRYPTION ALGORITHMS
SYMMETRIC ENCRYPTION ALGORITHMS
ASYMMETRIC ENCRYPTION ALGORITHMS
DES
AES
RSA
DIFFIE - HELLMAN

Fig. 1.7: Encryption Algorithms

In our project, we have implemented the RSA encryption algorithm with MATLAB. In this, the sender is asked to enter the message to be encrypted. The message is encrypted using the public key and can only be decrypted by the private key (available only to the receiver).
We have also integrated the AES and Diffie-Hellman encryption algorithms to demonstrate how both symmetric and associative encryption algorithms can be collated to build a powerful encryption technique. In this, we generate a shared key using the Diffie-Hellman algorithm and then, we use that shared key to encrypt messages using the AES algorithm.

Chapter 2

Requirement Specification

This section lists the various hardware and software requirements to achieve the objectives of the project. It catalogues the different hardware and software tools used in this project to implement AES, Diffie Hellman and RSA encryption algorithms.

2.1 Hardware Requirements

The hardware requirements needed to complete this project are listed in Table 1 along with their minimum and recommended size or speed.

Table 2.1: Hardware requirements

Hardware	Minimum	Recommended
Memory	512 GB	1 GB or more
Free Disc Space	300 MB	1 GB or more
Processor speed	800 GH	1.5 Ghz or faster

2.2 Software Requirements

The software requirements needed to complete this project are listed in Table 2 along with their minimum and recommended versions.

Table 2.2: Software requirements

Software	Minimum	Recommended
C++	C++ 14	C++ 17 or higher
VS Code IDE	VS Code 1.31	VS Code 1.58 or higher
MATLAB	MATLAB 9.4 - R2018a	MATLAB 9.11 - R2021b or higher

Chapter 3

System Design and Implementation

This section details the algorithm of the various encryption methods discussed in the literature. It provides an in-depth explanation of how each method works along with an example. The pseudo codes of the algorithms have also been provided in this section.

3.1 Modular Description

3.1.1 RSA algorithm

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. RSA is used for purposes when any sender can encrypt a message using the public key but only the receiver can decrypt it with the private key in their possession. It involves concepts of Euler's theorem and modular arithmetic.

Consider a scenario where Alice and Bob want to share a secret key. The algorithm for RSA has been described below

1. Bob starts with two large prime numbers p and q. He computes their product n=pq and another product φ(n) = (p-1)(q-1).
2. Now he chooses a coprime e with respect to φ(n) and announces e and n publicly.So {n, e} is the public key which will be used for encryption.
3. Bob computes de = 1 mod φ(n) and uses that to form his private key. The key that he uses later for decryption is {d, n}, of which n is provided in a public channel so essentially d is the private key.
4. Alice encrypts a message m as $c=m^e$ mod n using publicly available e and n and sends it back to Bob over a public channel.
5. Bob decrypts the message as $m = c^d$ mod n. Since only Bob has d the message can be safely decrypted.

Consider an example of the same. Let Bob choose 2 prime numbers p = 3 and q = 11 as shown in Fig. 3.1. He computes their product as n = p * q = 3 * 11 = 33 and φ(n) = (p - 1) * (q - 1) = 2 * 10 = 20. Now he chooses e = 7 following the rule, 1 < e < φ(n) and e and φ (n) are coprime. He announces n = 33 and e = 7 publicly accessible to Alice and Eve (Intruder). Bob also computes d = 3 as (d * e) % φ(n) = 1 => (3 * 7) % 20 = 1. Alice and Eve do not have access to d; it is Bob's private key. Alice chooses a message m = 2 and encrypts it using $c=m^e$ mod n, therefore $c = 2^7$ % 33 = 29. Alice sends the encrypted message over a public channel accessible to Eve and Bob, but only Bob will be able to decrypt the message as only he has private key d. Bob decrypts the message as $m = c^d$ mod n, therefore $m = 29^3$ % 33 = 2.

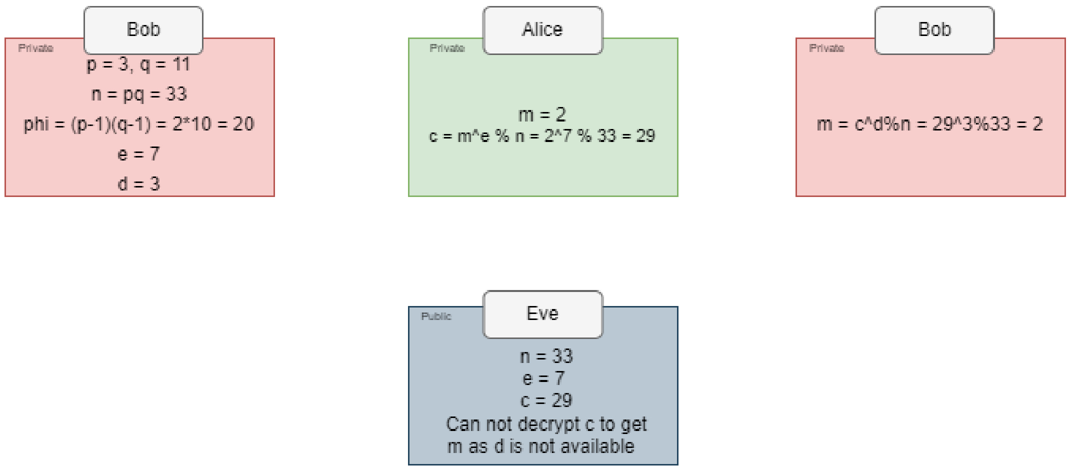


Fig 3.1: Procedure of RSA

RSA algorithm is based on the fact that prime generation and multiplication is easy (i.e It's easy to find random prime numbers p,q of a given size and compute their product n=pq) however factoring is hard - given a value of n, it appears to be quite hard to recover the prime factors p and q. Further, modular exponentiation is easy - given n, m, and e, it's easy to compute c = m^e mod n however modular root extraction is difficult if prime numbers are not given (i.e Given only n, e, and c in the public channel but not the prime factors p and q, it appears to be quite hard to recover the value m). Fig 3.2 shows the time complexity for factoring versus multiplying as size of input increases.

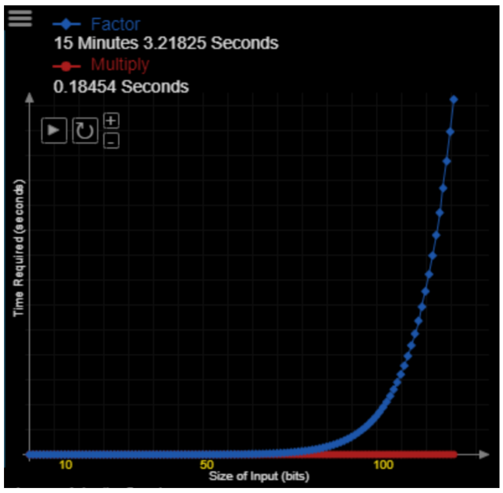


Fig. 3.2: Time Complexity graph for Factoring v/s Multiplying

Mathematical proof for RSA is as follows -

$c = m^e \mod n$ -----(i) Encryption

$m = c^d \mod n$ -----(ii) Decryption

Substitute (i) in (ii)

$m = (m^e \mod n)^d \mod n$

$m = m^{ed} \mod n$ ----- By modular exponentiation rule

To Prove: $m^{ed} \equiv m \pmod{n}$

If de = 1 mod φ(n), then de = 1 + k*φ(n) ------(iv)

$m^{\varphi(n)} \equiv 1 \pmod{n}$ ----- By Euler's theorem

$m^{\varphi(n)*k} \equiv 1^k \pmod{n}$

$m^{\varphi(n)*k} * m \equiv 1^k * m \pmod{n}$

$m^{\varphi(n)*k+1} \equiv m \pmod{n}$ -----(v)

Substitute (iv) in (v)

$m^{ed} \equiv m \pmod{n}$

3.1.2 Diffie Hellman Algorithm

This algorithm is an asymmetric encryption algorithm. It is commonly used in TLS encryption to establish the shared key (session key) which is later used for an asymmetric encryption algorithm like AES. It involves creation of a function using the concepts of group theory and modular arithmetic such that f(f(x, a), b) = f(f(x, b), a). This equivalent value is the shared key.

Consider a scenario where Alice and Bob want to share a secret key. The algorithm for Diffie Hellman has been described below:

1. Alice and Bob agree to use a prime number N (chosen arbitrarily) and base g (g is one of the generators of the mod N group)
2. Alice chooses a secret integer a, then sends the value $A = g^a \bmod N$ to Bob
3. Bob chooses a secret integer b, then sends the value $B = g^b \bmod N$ to Alice
4. Alice computes $s = B^a \bmod N$
5. Bob computes $s = A^b \bmod N$ and s is the shared secret key.

This works on the principle of modular exponentiation (i.e., $A^B \bmod C = ((A \bmod C)^B) \bmod C$). The net operation being performed on both sides g^{ab}= mod N which turns out to be the same number.

Consider an example of the same. Assume the prime number N=11, g=7, as shown in Fig 3.3. Let the secret integer (Alice's secret number) a=2, (Bob's secret number) b=4. Similar to the steps described in the algorithm, Alice sends the value A = 7^2 mod 11 = 5 to Bob and Bob sends the value B = 7^4 mod 11 = 3 to Alice. Alice computes s= 3^2 mod 11= 9 and Bob computes s= 5^4 mod 11= 9. These 2 numbers are equal (i.e., 9) which is the shared key.

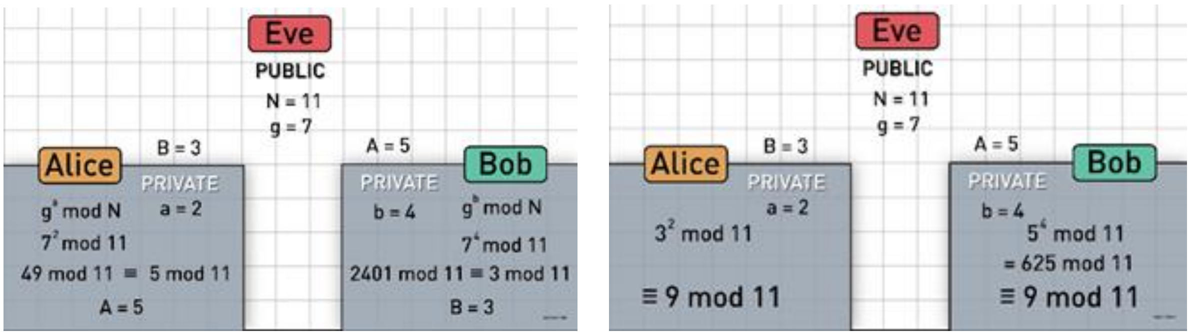


Fig 3.3: Procedure of Diffie Hellman method

3.1.3 AES Encryption

AES Encryption, also known as Rijndael Algorithm, is a symmetric block cipher algorithm with a block size of 128 bits. It converts these individual blocks into cipher text using keys of size 128 bits or 192 bits or 256 bits. The AES is a computer security standard for cryptographically securing electronic information, usually secret and top-secret government information. The standard is published and maintained by the NIST.

As shown in Fig. 3.4, the sender sends the plaintext to the encryption server where AES secret key is used to encrypt it, in order to generate the cipher text. The receiver can read the contents of the file only when the same AES secret key is used to decrypt it.

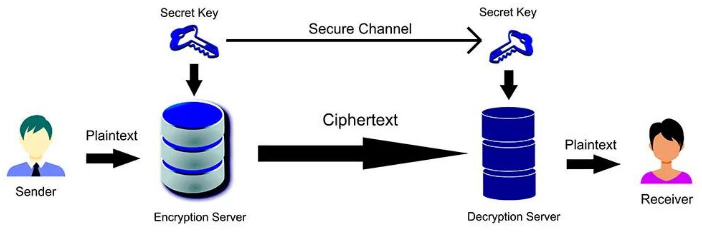


Fig. 3.4: AES Encryption and Decryption

Features of AES include the following:

i) AES is an example of key-alternating block ciphers. In such ciphers, each round first applies a diffusion-achieving transformation operation which may be a combination of linear and nonlinear steps to the entire incoming block, which is then followed by the application of the round key to the entire block. Key alternating ciphers lend themselves well to theoretical analysis of the security of the ciphers.

ii) AES is a byte-oriented cipher i.e., performs operations on byte data instead of bit data. Hence, software implementation of AES is fast.

iii) AES was designed using the wide-trail strategy.

(1) A local nonlinear transformation (as supplied by the substitution step in AES); and
(2) A linear mixing transformation that provides high diffusion.

AES Encryption process takes place in the following steps:
To begin with, information is divided into 128 bits, which is 16 bytes i.e., 4 by 4 matrix. This 4 x 4 array of bytes is referred to as a state array. After the data is split into 128 bits, the following transformation takes place:
i. Key expansion: In AES 128, the steps of adding shifting rows and mixing rounds has to be performed 11 times with different round keys, as shown in Fig. 3.5. Since there are 11 keys, each being 16 bytes long, an expanded key 176 bytes long has to be produced from an original 16 bytes long key. A key expansion core process, which takes previously generated 4 bytes of the key and performs left rotation, substitution and XORing the first byte with the Rcon value corresponding to the Rcon iteration number on the first 4 bytes of a key. Then, XOR the 4 bytes so obtained to the bytes 16 from the bytes generated.

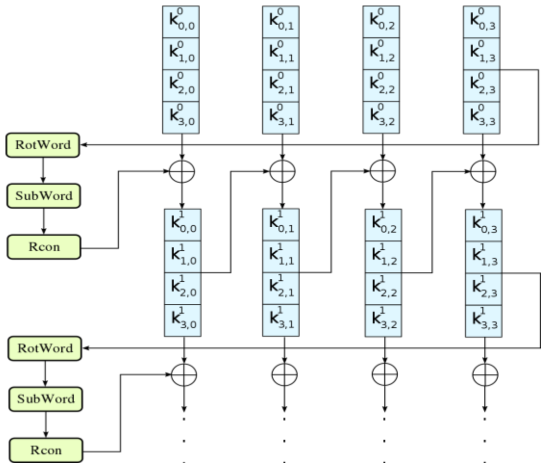
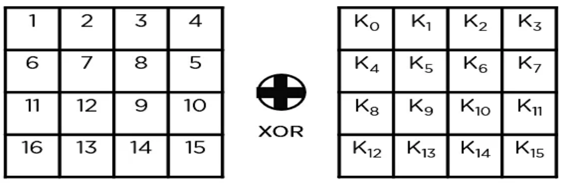


Fig. 3.5 : Key expansion

ii. Add Round Key: This step involves adding the state and the round key using a special type of arithmetic i.e., Galois fields(finite fields). Basically, the data stored in a state array is passed through an XOR function with the first key generated as shown in Fig. 3.6 .



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

K_0	K_1	K_2	K_3
K_4	K_5	K_6	K_7
K_8	K_9	K_{10}	K_{11}
K_{12}	K_{13}	K_{14}	K_{15}

Fig. 3.6 : Add Round Key

iii. Sub-Bytes: This is a non-linear byte replacement stage. In this step, it converts each byte of the state array into hexadecimal, divided into two equal parts. These parts are the rows and columns, mapped with a substitution box (S-Box) to generate new values for the final state array as shown in Fig. 3.7. S-box is a pre-calculated replacement table which holds 256 numbers (from 0 - 255) and their matching resulting value. Sub-Bytes is based on Galois Field Inverse operation GF (2^8) known to have excellent non-linearity properties. The use of Galois Field is to prevent attacks based on simple algebraic properties.

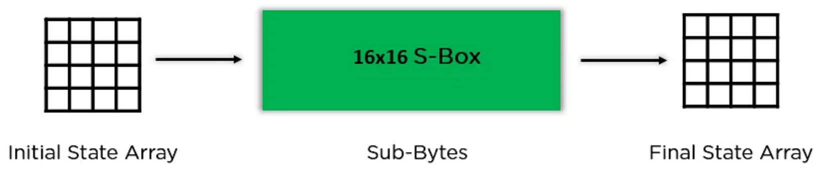


Fig. 3.7 : Sub-Bytes

iv. Shift Rows: It swaps the row elements among each other. It skips the first row. It shifts the elements in the second row, one position to the left. It also shifts the elements from the third row two consecutive positions to the left, and it shifts the last row three positions to the left (Fig. 3.8). The goal of this transformation is to scramble the byte order inside each 128-bit block.

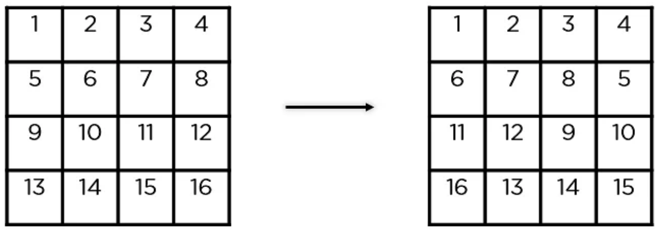


Fig. 3.8 : Shift Rows

v. Mix Columns: It multiplies a constant matrix with each column in the state array to get a new column for the subsequent state array. Once all the columns are multiplied with the same constant matrix, we get your state array for the next step. This particular step is not to be done in the last round (Fig. 3.9). Instead of multiplying dot product is performed and in place of addition XOR is done.

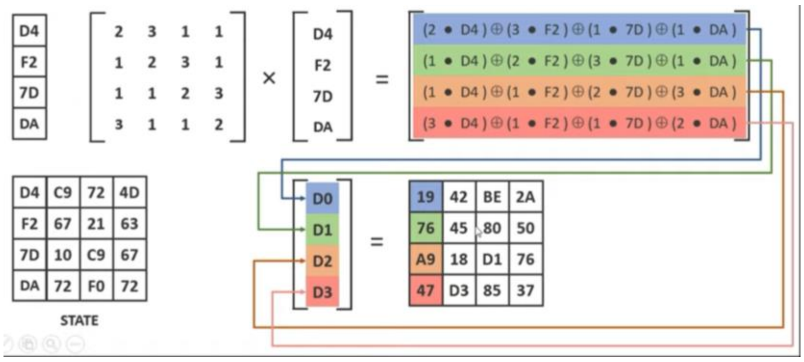


Fig. 3.9 : Mix Columns

vi. Add Round Key: In this step, the data stored in the state array is passed through an XOR function with a different key generated by key expansion as shown in Fig. 3.10 .

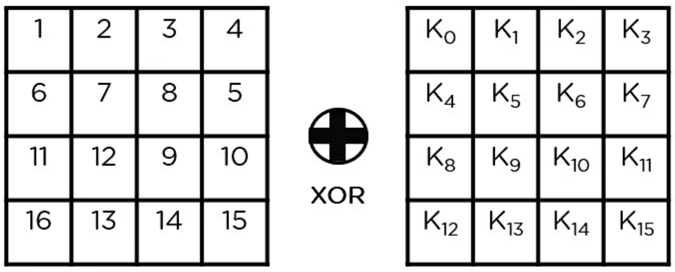


Fig. 3.10 : Add Round Key

AES involves taking the MI of each byte in GF(2^8) and bit scrambling. This is the only nonlinear step in mapping a plaintext block to a ciphertext block in AES. The MI byte substitution step in AES is meant to protect it against such cryptanalysis. The block ciphers whose S_Boxes were based on polynomial arithmetic in Galois fields could be vulnerable to a new attack referred to as the interpolation attack. The bit scrambling part of the S_Box in AES is meant to be a protection against the interpolation attack.

Fig. 3.11 shows the iterations that take place in AES Encryption. The number of iterations 'Nr' depends on the key size. A 128-bit key size dictates ten rounds, a 192-bit key size dictates 12 rounds, and a 256-bit key size has 14 rounds. AES uses a substitution-permutation network in a more general sense. Each round of processing in AES involves byte-level substitutions followed by word-level permutations.The encryption phase of AES can be broken into three phases: the initial round (the first round), the main rounds, and the final round.

- Initial Round
  - AddRoundKey
- Main Rounds
  - SubBytes
  - ShiftRows
  - MixColumns
  - AddRoundKey
- Final Round
  - SubBytes
  - ShiftRows
  - AddRoundKey

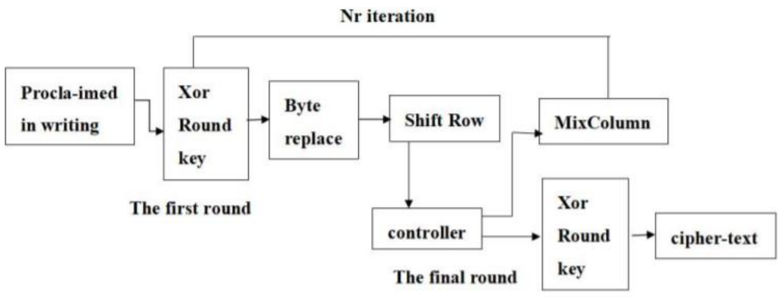


Fig. 3.11 : AES Encryption flowchart

3.2 Pseudo Code

3.2.1 RSA

Step 1: Take any 2 prime numbers as input and store it in p,q
Step 2: Compute n = p*q and print n
Step 3: Compute phi = (p-1)*(q-1) and print phi

```
Step 4: Initialise 2 variables val (used to check if e is prime or not) and cd (used to store gcd of $\varphi(n)$ and e) to 0
Step 5: Generate e, a random integer  between 1 and phi
Step 6: Compute val = isprime(e)
Step 7: Compute cd = gcd(e,phi)
Step 8: If cd is not equal to 1 or val is equal to 0 then repeat steps 5,6,7 else proceed to step 9
Step 9: Print e (e is computed in above steps such that $1<e<\varphi(n)$ and $\gcd(e,\varphi(n)) = 1$)
Step 10: Initialise val1 (used to store mod value of de and $\varphi(n)$) and d to 0
Step 11: Compute d = d+1
Step 12: Compute val1 = mod(d*e,phi)
Step 13: If val1 is not equal to 1 repeat steps 11,12 else proceed to step 14
Step 14: Print d (d is computed in above steps such that $de = 1 \bmod \varphi(n)$ )
Step 15: Print Public key {e,n} and Private key {d,n} as computed above
Step 16: Take a message to be encrypted as input and store it in array m
Step 17: Store the ascii values of all the character of m in array m1
Step 17: Apply encryption formula $c=m^e \bmod n$ for each number in array m1 in  and store it in array c1
Step 18: Print m1 which has ascii representation of message and c1 which has encrypted message
Step 19: Apply decryption formula $m = c^d \bmod n$ for each number in array c1 and store in array nm
Step 20: Convert each number in array nm to it's character representation and store it in array nm1
Step 21: Print nm - the ascii representation of decrypted message and nm1 - the decrypted message itself
```

3.2.2 Diffie Hellman and AES

Algorithm of main() function

```
Step 1: Set p to randomly generated prime number between 0 and 11 and print p
Step 3: Set g to one of the (randomly selected) generators of the mod p group and print g
Step 3: Set a to any randomly selected number between 0 and p and print a
Step 4: Compute $A= g^a \bmod p$
Step 5: Set b to any randomly selected number between 0 and p and print b
Step 6: Compute $B= g^b \bmod p$
Step 7: Compute $keyA= B^a \bmod p$ //shared key
Step 8: Compute $keyB= A^b \bmod p$ //shared key
Step 9: Print "Alice's secret key is " + keyA
Step 10: Print "Bob's secret key is " + keyB
Step 11: If keyA is 1
                initialize key to {12,14,67,4,34,65,37,23,17,98,45,35,56,10,21,19}
        else if keyA is 2
                initialize key to {1,4,6,4,38,64,33,22,11,99,55,39,65,15,23,18}
        else if keyA is 3
                initialize key to {24,28,63,8,68,30,44,46,34,95,90,70,56,20,42,38}
        else if keyA is 4
                initialize key to {4,23,69,81,50,33,41,66,35,57,9,7,6,29,27,3}
        else if keyA is 5
                initialize key to {87,82,76,4,43,56,73,32,71,89,54,53,65,1,12,91}
        else if keyA is 6
                initialize key to {25,14,62,40,49,52,78,31,73,81,51,36,26,10,22,78}
        else if keyA is 7
                initialize key to {64,94,52,8,86,21,46,6,42,9,4,30,51,15,22,13}
        else if keyA is 8
                initialize key to {29,47,67,40,36,65,34,23,15,98,45,35,12,30,44,26}
        else if keyA is 9
                initialize key to {28,14,69,4,32,66,37,23,7,98,5,35,56,19,27,10}
        else if keyA is 10
                initialize key to {12,15,16,13,14,11,17,10,7,9,6,5,4,3,2,1}
        else
                initialize key to {1,14,7,4,3,6,14,46,38,9,5,37,57,17,27,19}
Step 10: Print "the shared key is" + key
Step 11: Input message to be encrypted into message
Step 12: The length of the entered message is stored in the variable named 'originalLen'
Step 13: Initialize lenOfPaddedMessage to originalLen
Step 14:  If lenOfPaddedMessage mod 16 is not equal to 0
                lenOfPaddedMessage=(lenOfPaddedMessage/16 + 1)*16 (It is padded such that its
length is a multiple of 16
Step 15: For i= 0 to lenOfPaddedMessage:
                if i> originalLen
                        Then initialize paddedMessage[i] to 0
                else
                        Initialize paddedMessage[i] to message[i]
        End of for
Step 16: For i= 0 to lenOfPaddedMessage:
          Call the function AES_Encrypt(paddedMessage[i], key)
          End of for
Step 17: Print the encrypted message obtained after calling the AES_Encrypt function
          For i=0 to lenOfPaddedMessage:
                print hexadecimal form of paddedMessage[i]+ " "
          End of for
```

```
Algorithm of AES_Encrypt(message,key) function:
Step 1: State array is initialized to the message array(parameter of the function)
Step 2: numberOfRounds is initialized to 9(excluding the last round), since the total number of
rounds in the 128 bit key is 10.
Step 3: expandedKey array of size 176 is defined.
Step 4: KeyExpansion(key, expandedKey) is called.
Step 5: AddRoundKey(state,key) is called.
Step 6: SubBytes(state)
```

```
          ShiftRows(state)
          MixColumns(state)
          AddRoundKey(state,expandedKey+(16*(i+1)))  //Main Rounds
Step 7: Repeat Step 6 'numberOfRounds' times.
Step 8: SubBytes(state)
          ShiftRows(state)
          AddRoundKey(state,expandedKey+(16*(i+1))) //Final Round
```

Step 9: message array is now initialized to state array to reflect the transformations made to state array.

Algorithm of KeyExpansion(key, expandedKey) function:

```
Step1: For i=0 to 16:
                    initialize expanded key[i] with the original key[i].
Step2: Initialize the bytesGenerated to 16
Step3: Initialize the rconIteration to 1
Step4: while the bytesGenerated is less than 176:
                    For i=0 to 16:
                              initialize temp[i] to expandedKey[i+bytesGenerated-4]
                    check if the bytesGenerated are integral multiple of 16
                              call the KeyExpansionCore(temp, rconIteration) function
                                        perform the left rotation operation
                                                  initialize t to temp[0]
                                                  set temp[0] to temp[1], temp[1] to temp[2],
                                                      temp[2] to temp[3], temp[3] to t
                              substitute for each byte with the corresponding values from S-box:
                                        temp[0] to s_box[0], temp[1] to s_box[1]
                                        temp[2] to s_box[2], temp[0] to s_box[3]
                              XOR the first byte with the Rcon value corresponding to the rconIteration:
                                        set temp[0] to temp[0]^rcon[rconIteration]
                    For a=0 to 4
                              update expandedKey[a] to expandedKey[bytesGenerated-16]^temp[a]
                              increment the bytesGenerated
          end of while
```

Algorithm of AddRoundKey(state,roundkey) function:

```
Step 1: For i=0 to 16:
                    initialize state[i]^roundKey[i] to state[i]
```

Algorithm of SubBytes(state) function:

```
Step 1: For i=0 to 16:
                    initialize s_box[state[i]] to state[i]
```

Algorithm of ShiftRows(state) function:

```
Step 1: tmp array of size 16 is defined
Step 2: initialize tmp[0] to state[0]
        initialize tmp[1] to state[5]
        initialize tmp[2] to state[10]
        initialize tmp[3] to state[15]

        initialize tmp[4] to state[4]
        initialize tmp[5]  to state[9]
        initialize tmp[6] to state[14]
        initialize tmp[7] to state[3]

        initialize tmp[8] to state[8]
        initialize tmp[9] to state[13]
        initialize tmp[10] to state[2]
        initialize tmp[11] to state[7]

        initialize tmp[12] to state[12]
        initialize tmp[13] to state[1]
        initialize tmp[14] to state[6]
        initialize tmp[15] to state[11]
Step 3: for i=0 to 16:
                initialize state[i] to tmp[i]
```

Algorithm of MixColumns(state) function:

```
Step 1: tmp array of size 16 is defined
Step 2: initialize tmp[0] to mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3]
        initialize tmp[1] to state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3]
        initialize tmp[2] to state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]]
        initialize tmp[3] to mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]]

        initialize tmp[4] to mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7]
        initialize tmp[5] to state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7]
        initialize tmp[6] to state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]]
        initialize tmp[7] to mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]]

        initialize tmp[8] to mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11]
        initialize tmp[9] to state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11]
        initialize tmp[10] to  state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]]
        initialize tmp[11] to mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]]

        initialize tmp[12] to mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15]
        initialize tmp[13] to state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15]
        initialize tmp[14] to state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]]
        initialize tmp[15] to mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]
```

```
Step 3: for i=0 to 16:
                initialize state[i] to tmp[i]
```

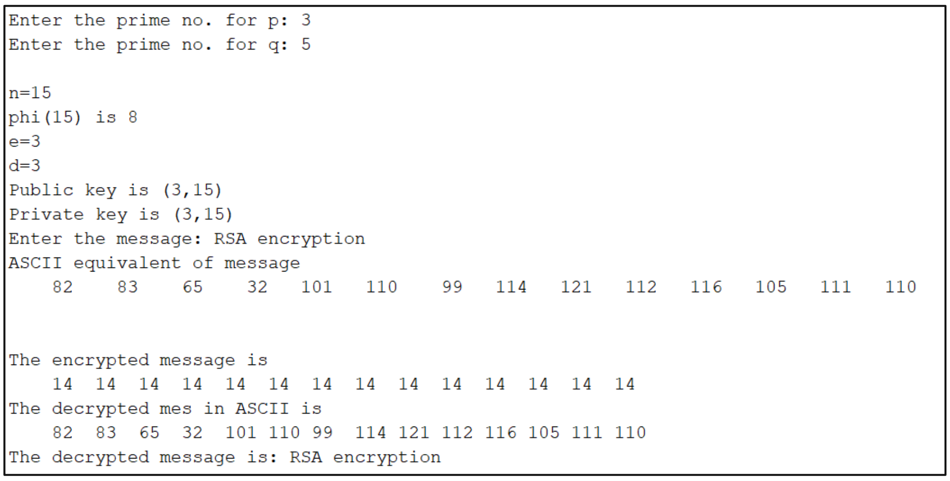
Chapter 4

Results and Snapshots

This section includes snapshots of the output obtained on running the 2 programs for implementation of RSA algorithm and implementation of AES and Diffie Hellman algorithms. It also gives an explanation for the results obtained in both the cases.

4.1 RSA

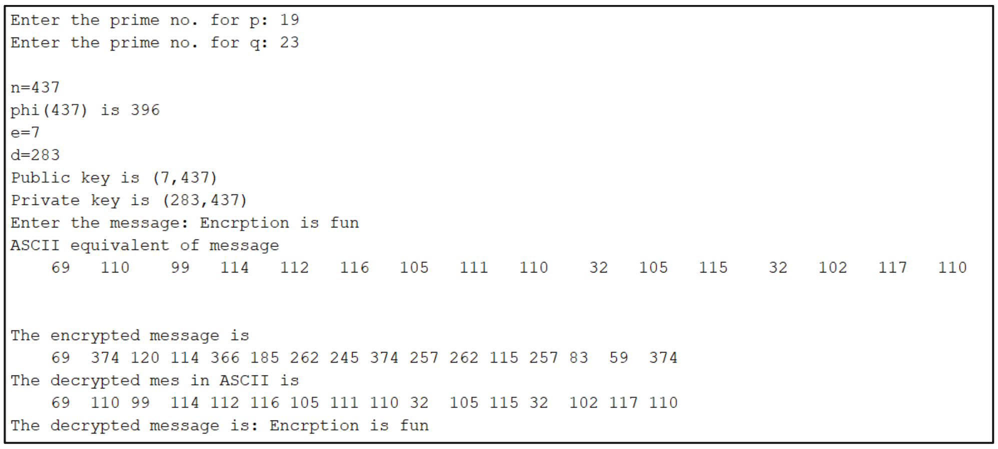
Fig.4.1 shows the output obtained on running the code for implementation of RSA encryption. In this case the input given for the 2 prime numbers are p = 3 and q = 5. The value of n = p*q = 3*5 = 15 is computed and displayed. Next the value of φ(n) = (p-1)(q-1) = 2*4 = 8 is computed and displayed. The value of e is computed and displayed as 3 such that 1< (e = 3) < (φ(n)=8) and gcd(e=3,φ(n)=8)=1. The value of d is computed and displayed as 3 such that de = 1 mod φ(n) => (3*3)%8 = 1. The public key {e,n} = {3,15} and the private key = {d,n} = {3,15} are printed. Next the message, in this case "RSA encryption" is taken as input. The ascii representation of the message is found and displayed. Next using the encryption formula $c=m^e$ mod n the message is encrypted and displayed. Using the decryption formula $m = c^d$ mod n the ascii representation of decrypted message is found and displayed. Finally the ascii is converted to character representation and the decrypted message is printed.



```
Enter the prime no. for p: 3
Enter the prime no. for q: 5

n=15
phi(15) is 8
e=3
d=3
Public key is (3,15)
Private key is (3,15)
Enter the message: RSA encryption
ASCII equivalent of message
    82    83    65    32   101   110    99   114   121   112   116   105   111   110


The encrypted message is
    14  14  14  14  14  14  14  14  14  14  14  14  14  14
The decrypted mes in ASCII is
    82  83  65  32  101 110 99  114 121 112 116 105 111 110
The decrypted message is: RSA encryption
```

Fig. 4.1: Example 1 for output of RSA implementation

Fig.4.2 shows another output obtained on running the code for implementation of RSA encryption again with different inputs.This time the input given for the 2 prime numbers are p = 19 and q = 23. The value of n = p*q = 19*23 = 437 is computed and displayed. Next the value of φ(n) = (p-1)(q-1) = 18*22 = 396 is computed and displayed. The value of e is computed and displayed as 7 such that 1< (e = 7) < (φ(n)=396) and gcd(e=7,φ(n)=396)=1. The value of d is computed and displayed as 283 such that de = 1 mod φ(n) => (283*7)%396 = 1. The public key {e,n} = {7,437} and the private key = {d,n} = {283,437} are printed. Next the message, in this case “Encryption is Fun” is taken as input. The ascii representation of the message is found and displayed. Next using the encryption formula $c=m^e$ mod n the message is encrypted and displayed. Using the decryption formula $m = c^d$ mod n the ascii representation of decrypted message is found and displayed. Finally the ascii is converted to character representation and the decrypted message is printed.



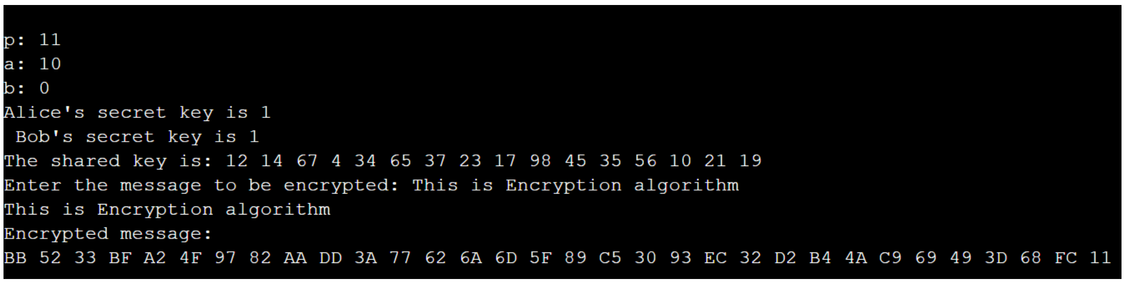
```
Enter the prime no. for p: 19
Enter the prime no. for q: 23

n=437
phi(437) is 396
e=7
d=283
Public key is (7,437)
Private key is (283,437)
Enter the message: Encrption is fun
ASCII equivalent of message
    69   110    99   114   112   116   105   111   110    32   105   115    32   102   117   110


The encrypted message is
    69  374 120 114 366 185 262 245 374 257 262 115 257 83  59  374
The decrypted mes in ASCII is
    69  110 99  114 112 116 105 111 110 32  105 115 32  102 117 110
The decrypted message is: Encrption is fun
```

Fig. 4.2 : Example 2 for output of RSA implementation

4.2 AES and Diffie Hellman

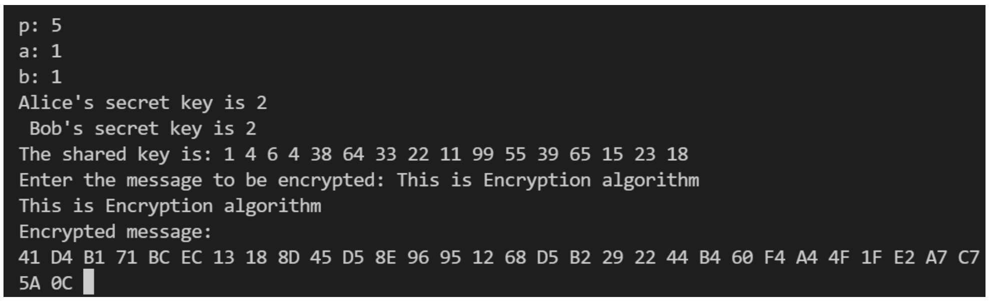
The output obtained from the implementation of Diffie Hellman and AES is shown in Fig 4.3 and Fig 4.4. The output displays the random prime number chosen for the Diffie Hellman encryption to get the shared key, p. Following this, the secret numbers chosen by the sender and receiver a and b are displayed. The value of the key obtained by the sender and receiver on performing the steps in the Diffie Hellman algorithm are printed. It is observed that these 2 are equal thereby verifying the Diffie Hellman algorithm.

Each possible shared key has been mapped to a unique 128 bit hexadecimal key which is passed to the AES algorithm for encryption of the message. It is observed that once the secret key is printed, the corresponding shared 128 bit key is found and printed. After this, the program takes in the message that must be encrypted as input and then prints the original message as well as the AES encrypted message in the hexadecimal format.



```
p: 11
a: 10
b: 0
Alice's secret key is 1
 Bob's secret key is 1
The shared key is: 12 14 67 4 34 65 37 23 17 98 45 35 56 10 21 19
Enter the message to be encrypted: This is Encryption algorithm
This is Encryption algorithm
Encrypted message:
BB 52 33 BF A2 4F 97 82 AA DD 3A 77 62 6A 6D 5F 89 C5 30 93 EC 32 D2 B4 4A C9 69 49 3D 68 FC 11
```

Fig. 4.3 : Example 1 for output of AES and Diffie Hellman implementation



```
p: 5
a: 1
b: 1
Alice's secret key is 2
 Bob's secret key is 2
The shared key is: 1 4 6 4 38 64 33 22 11 99 55 39 65 15 23 18
Enter the message to be encrypted: This is Encryption algorithm
This is Encryption algorithm
Encrypted message:
41 D4 B1 71 BC EC 13 18 8D 45 D5 8E 96 95 12 68 D5 B2 29 22 44 B4 60 F4 A4 4F 1F E2 A7 C7
5A 0C
```

Fig. 4.4 : Example 2 for output of AES and Diffie Hellman implementation

Chapter 5

Conclusion

Today's generation is digitalized in every manner possible. One of the core technologies of information security is cryptography. There has been rapid progress in cryptography, and cryptography occupies an increasingly important position.
Since there is a demand for high and real time encryption algorithms in more and more fields, they are widely used as a standard to authenticate routers and clients. They play an important role in securing website server authentication from both client and server end.

Fig. 5.1 shows that AES consumes lesser encryption time than RSA.

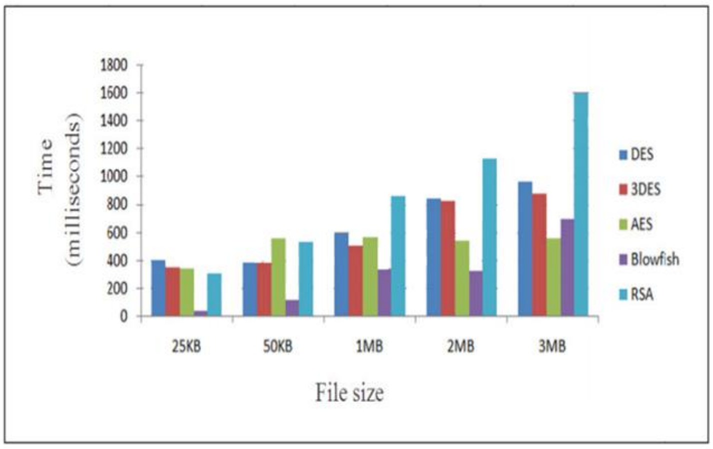


Fig. 5.1 : Comparison of time taken to encrypt in various encryption algorithms

Chapter 6

Bibliography

[1] Mehrdad Biglari1 , Ehsan Qasemi2 , Behnaz Pourmohseni2. Maestro: A High Performance AES Encryption/Decryption System. Published in: The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013) Date of Conference: 30-31 Oct. 2013 DOI: 10.1109/CADS.2013.6714255

[2] Mingying Chen; Hongling Wei; Hongyan Li. Architecture design and hardware implementation of AES encryption algorithm. Published in: 2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE) Date of Conference: 25-27 Dec. 2020 INSPEC Accession Number: 20692569 DOI: 10.1109/ICMCCE51767.2020.00353

[3] P. Joshi, M. Verma and P. R. Verma, "Secure authentication approach using Diffie-Hellman key exchange algorithm for WSN," 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2015, pp. 527-532, doi: 10.1109/ICCICCT.2015.7475336.

[4] U. Maurer and S. Wolf, "Diffie-Hellman, decision Diffie-Hellman, and discrete logarithms," Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No.98CH36252), 1998, pp. 327-, doi: 10.1109/ISIT.1998.708932.

[5] A. N. Borodzhieva, "Software implementation of a module for encryption and decryption using the RSA algorithm," 2016 XXV International Scientific Conference Electronics (ET), 2016, pp. 1-4, doi: 10.1109/ET.2016.7753464.

[6] Qixin Zhang, "An Overview and Analysis of Hybrid Encryption: The Combination of Symmetric Encryption and Asymmetric Encryption", 2021 2nd International Conference on Computing and Data Science (CDS), 2021, pp, 1-5, doi: 10.1109/CDS52072.2021.00111.

[7] M. Frunza and L. Scripcariu, "Improved RSA Encryption Algorithm for Increased Security of Wireless Networks," *2007 International Symposium on Signals, Circuits and Systems*, 2007, pp. 1-4, doi: 10.1109/ISSCS.2007.4292737.

[8] https://www.techiedelight.com/c-program-demonstrate-diffie-hellman-algorithm/
[9] https://brilliant.org/wiki/group-theory-introduction/
[10] https://www.youtube.com/watch?v=ESPT_36pUFc
[11] https://www.khanacademy.org/computing/computer-science/cryptography
[12]https://www.nku.edu/~christensen/the%20mathematics%20of%20the%20RSA%20cryptosystem.pdf

Appendix

Source Code

1. **RSA encryption - code written using MATLAB software**

```
clc;
disp('RSA algorithm');
p=input('Enter the prime no. for p: ');
q=input('Enter the prime no. for q: ');
n=p*q;
fprintf('\nn=%d',n);
phi=(p-1)*(q-1);
fprintf('\nphi(%d) is %d',n,phi);
val=0;
cd=0;
```

```
while(cd~=1||val==0)
n1=randi(n,1,1);
e=randi([2 n1],1,1);
 val=isprime(e);
 cd=gcd(e,phi);
end
fprintf("\ne=%d",e)
val1=0;
d=0;
while(val1~=1)
d=d+1;
val1=mod(d*e,phi);
end
fprintf('\nd=%d',d);
fprintf('\nPublic key is (%d,%d)',e,n);
fprintf('\nPrivate key is (%d,%d)',d,n);
m=input('\nEnter the message: ','s');
m1=m-0;
disp('ASCII equivalent of message ');
disp(m1);
over=length(m1);
 o=1;
 while(o<=over)
    m=m1(o);
    diff=0;
    if(m>n)
        diff=m-n+1;
    end
    m=m-diff;
qm=dec2bin(e);
len=length(qm);
c=1;
xz=1;
while(xz<=len)
    if(qm(xz)=='1')
      c=mod(mod((c^2),n)*m,n);
    elseif(qm(xz)=='0')
        c=(mod(c^2,n));
    end
    xz=xz+1;
end
c1(o)=c;
qm1=dec2bin(d);
len1=length(qm1);
nm=1;
xy=1;
while(xy<=len1)
    if(qm1(xy)=='1')
       nm=mod(mod((nm^2),n)*c,n);
    elseif(qm1(xy)=='0')
```

```
        nm=(mod(nm^2,n));
    end
     xy=xy+1;
end
nm=nm+diff;
nm1(o)=char(nm);
o=o+1;
 end
o=1;
fprintf('\nThe encrypted message is \n');
while(o<=over)
   fprintf('\t%d',c1(o));
   o=o+1;
end
o=1;
fprintf('\nThe decrypted mes in ASCII is \n');
while(o<=over)
fprintf('\t%d',nm1(o));
o=o+1;
end
fprintf('\nThe decrypted message is: ');
disp(nm1);
fprintf('\n');
```

2. **AES and Diffie Hellman encryption - code written in C++**

```
#include <iostream>
#include <conio.h>
#include <string.h>
#include <time.h>
#include <math.h>
using namespace std;

unsigned char s_box[256] = {
 /* 0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F  */
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab,
0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72,
0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31,
0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2,
0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f,
0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58,
0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f,
0xa8,
   0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3,
0xd2,
   0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19,
0x73,
   0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b,
0xdb,
   0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4,
0x79,
   0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae,
0x08,
   0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b,
0x8a,
   0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d,
0x9e,
   0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28,
0xdf,
   0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb,
0x16
};

unsigned char mul2[] = {
   0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,
   0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,
   0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,
   0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,
   0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,
   0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,
   0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,
   0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,
   0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,
   0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,
   0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,
   0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,
   0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,
   0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,
   0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,
   0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5
};

unsigned char mul3[] = {
   0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,
   0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,
   0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,
   0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,
   0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,
   0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,
   0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,
   0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,
   0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,
   0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,
   0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,
   0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,
   0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,
   0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,
   0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,
   0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a
};

unsigned char rcon[256] = {
   0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
0x9a,
   0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91,
0x39,
   0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,
0x3a,
   0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c,
0xd8,
   0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,
0xef,
   0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66,
0xcc,
   0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
0x1b,
   0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4,
0xb3,
   0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a,
0x94,
   0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10,
0x20,
   0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97,
0x35,
   0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2,
0x9f,
   0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,
0x04,
   0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc,
0x63,
   0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3,
0xbd,
   0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb,
0x8d
};

void KeyExpansionCore(unsigned char* in, unsigned char i){
   /*unsigned int *q=(unsigned int*)in;
   *q=(*q>>8)| ((*q &0xff)<<24);*/
   unsigned char t=in[0];
   in[0]=in[1];
   in[1]=in[2];
   in[2]=in[3];
   in[3]=t;
   in[0]=s_box[in[0]]; in[1]=s_box[in[1]];
   in[2]=s_box[in[2]]; in[3]=s_box[in[3]];

   in[0]^=rcon[i];
}
void KeyExpansion(unsigned char *inputKey, unsigned char *expandedKeys){
   for(int i=0; i<16;i++)
   {
       expandedKeys[i]=inputKey[i];
   }
   int bytesGenerated=16;
   int rconIteration=1;
   unsigned char temp[4];

   while (bytesGenerated<176)
   {
       for(int i=0;i<4;i++)
           temp[i]=expandedKeys[i+bytesGenerated-4];
       if(bytesGenerated % 16==0)
           KeyExpansionCore(temp,rconIteration++);
       for(unsigned char a=0;a<4;a++)
       {
           expandedKeys[bytesGenerated]=expandedKeys[bytesGenerated-16]^temp[a];
           bytesGenerated++;
       }
   }
}

void SubBytes(unsigned char* state){
   for(int i=0;i<16;i++){
       state[i]=s_box[state[i]];
   }
}
void ShiftRows(unsigned char* state){
   unsigned char tmp[16];
   tmp[0]=state[0];
   tmp[1]=state[5];
   tmp[2]=state[10];
   tmp[3]=state[15];

   tmp[4]=state[4];
   tmp[5]=state[9];
   tmp[6]=state[14];
   tmp[7]=state[3];

   tmp[8]=state[8];
   tmp[9]=state[13];
   tmp[10]=state[2];
   tmp[11]=state[7];
   tmp[12]=state[12];
   tmp[13]=state[1];
   tmp[14]=state[6];
   tmp[15]=state[11];
   for(int i=0;i<16;i++){
       state[i]=tmp[i];
   }
}

void MixColumns(unsigned char* state){

   unsigned char tmp[16];
   tmp[0] = (unsigned char)(mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3]);
   tmp[1] = (unsigned char)(state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3]);
   tmp[2] = (unsigned char)(state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]]);
   tmp[3] = (unsigned char)(mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]]);

   tmp[4] = (unsigned char)(mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7]);
   tmp[5] = (unsigned char)(state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7]);
   tmp[6] = (unsigned char)(state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]]);
   tmp[7] = (unsigned char)(mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]]);

   tmp[8] = (unsigned char)(mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11]);
   tmp[9] = (unsigned char)(state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11]);
   tmp[10] = (unsigned char)(state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]]);
   tmp[11] = (unsigned char)(mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]]);

   tmp[12] = (unsigned char)(mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15]);
   tmp[13] = (unsigned char)(state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15]);
   tmp[14] = (unsigned char)(state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]]);
   tmp[15] = (unsigned char)(mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]);

   for(int i=0; i<16; i++)
       state[i]=tmp[i];
}

void AddRoundKey(unsigned char* state,unsigned char* roundKey){
   for(int i=0;i<16;i++){
       state[i]^=roundKey[i];
   }
}

void AES_Encrypt(unsigned char* message,unsigned char* key){

   unsigned char state[16];
   for(int i=0;i<16;i++){
       state[i]=message[i];
   }
   int numberOfRounds=9;
   unsigned char expandedKey[176];
   KeyExpansion(key,expandedKey);

   AddRoundKey(state,key);
   for(int i=0;i<numberOfRounds;i++){
       SubBytes(state);
       ShiftRows(state);
       MixColumns(state);
       AddRoundKey(state,expandedKey+(16*(i+1)));
   }
   //Final Round
   SubBytes(state);
   ShiftRows(state);
   AddRoundKey(state,expandedKey+160);

   for(int i=0;i<16;i++){
       message[i]=state[i];
   }
}

void PrintHex(unsigned char x){
   if(x/16<10) cout<<(char)((x/16)+'0');
   if(x/16>=10) cout<<(char)((x/16-10)+'A');

   if(x%16<10) cout<<(char)((x%16)+'0');
   if(x%16>=10) cout<<(char)((x%16-10)+'A');
}

// Function to compute `a^m mod n`
int compute(int a, int m, int n)
{
   int r;
   int y = 1;

   while (m > 0)
   {
       r = m % 2;

       // fast exponention
       if (r == 1) {
           y = (y*a) % n;
       }
       a = a*a % n;
       m = m / 2;
   }

   return y;
}
```

```
int generate(){
    srand(time(NULL));
   int primeN=rand(), flag=0,i;
   do{
       primeN=(rand()%11)+1;
   for(i=2;i<primeN/2;i++)
   {
       if(primeN%i==0)
       break;
   }
   if(i==primeN/2)
   flag=1;
   }while(flag!=1);
   //cout<<primeN<<endl;
   return primeN;
}

int gcd(int a, int b)
{
    int q, r;
    while (b > 0)
    {
        q = a / b;
        r = a - q * b;
        a = b;
        b = r;
    }
    return a;
}

int gens(int a)
{
    char ch, op;
    int k;
    int mode = 0;
    int val;
    int ord[50], g[50], H[50];
    bool cyclic = false;
    int elements[100];
    int order;
    ch='*';
    mode = 1;
    op='*';
        if (mode)
        {
            //printf("G=<Z%d*,%c>\n", a, op);
            k = 0;

            for (int i = 0; i < a; i++)
            {
```

```
        if (gcd(a, i) == 1) {        // co-prime
            elements[k++] = i;
        }
    }

    elements[k] = '\0';
    mode = 0;
}
else {
    //printf("G=<Z%d,%c>\n", a, op);
    op='*';
    for (int i = 0; i < a; i++) {
        elements[i] = i;
    }
    k = a;
}

/*printf("G = {");
for (int i = 0; i < k; i++)
{
    printf("%d", elements[i]);
    (i < k-1) ? printf(",") : printf("}");
}*/

order = k;
//printf("\nThe order of the group is %d.\n\n", order);

for (int i = 0; i < k; i++)
{
    int temp = 1;
    while (1)
    {
        if (op == '*')
        {
            val = (int)(pow(elements[i], temp));
            if (val % a == 1)
            {
                ord[i] = temp;
                //printf("order(%d) = %d\n", elements[i], temp);
                break;
            }
        }

        temp++;
    }
}

int x = 0, count = 0;

for (int i = 0; i < k; i++)
```

```
        {
           if (ord[i] == order)
           {
              g[x++] = elements[i];
              cyclic = true;
              count++;
           }
        }

        if (cyclic)
        {
           //printf("A group is a cyclic group with %d generators", count);
           srand(time(NULL));
           return g[rand()%count];
        }
        else {
           cout<<"The group is not a cyclic group."<<endl;
           exit(0);
        }

        printf("\n\n————————————————————————————————————————————
————————————————————\n\n");
}

int main()
{
   srand(time(NULL));
   int p = generate(); // modulus
   printf("\np: %d",p);
   int g = gens(p);       // base

   int a, b;    // `a` – Alice's secret key, `b` – Bob's secret key.
   int A, B;    // `A` – Alice's public key, `B` – Bob's public key

   // choose a secret integer for Alice's private key (only known to Alice)
   a = rand() % p;        // or, use `rand()`
   printf("\na: %d",a);

   // Calculate Alice's public key (Alice will send `A` to Bob)
   A = compute(g, a, p);

   // choose a secret integer for Bob's private key (only known to Bob)
   b = rand() % p;       // or, use `rand()`
   printf("\nb: %d\n",b);

   // Calculate Bob's public key (Bob will send `B` to Alice)
   B = compute(g, b, p);

   // Alice and Bob Exchange their public key `A` and `B` with each other
// Find secret key
int keyA = compute(B, a, p);
int keyB = compute(A, b, p);
unsigned char key[16];
cout<<"Alice's secret key is "<<keyA<<endl;
cout<<" Bob's secret key is "<<keyB<<endl;
int i;
unsigned char a1[]={12,14,67,4,34,65,37,23,17,98,45,35,56,10,21,19};
unsigned char a2[]={1,4,6,4,38,64,33,22,11,99,55,39,65,15,23,18};
unsigned char a3[]={24,28,63,8,68,30,44,46,34,95,90,70,56,20,42,38};
unsigned char a4[]={4,23,69,81,50,33,41,66,35,57,9,7,6,29,27,3};
unsigned char a5[]={87,82,76,4,43,56,73,32,71,89,54,53,65,1,12,91};
unsigned char a6[]={25,14,62,40,49,52,78,31,73,81,51,36,26,10,22,78};
unsigned char a7[]={64,94,52,8,86,21,46,6,42,9,4,30,51,15,22,13};
unsigned char a8[]={29,47,67,40,36,65,34,23,15,98,45,35,12,30,44,26};
unsigned char a9[]={28,14,69,4,32,66,37,23,7,98,5,35,56,19,27,10};
unsigned char a10[]={12,15,16,13,14,11,17,10,7,9,6,5,4,3,2,1};
unsigned char a11[]={1,14,7,4,3,6,14,46,38,9,5,37,57,17,27,19};
switch(keyA){
   case 1:
   for(i=0;i<16;i++)
   key[i]=a1[i];
   break;
   case 2:
   for(i=0;i<16;i++)
   key[i]=a2[i];
   break;
   case 3:
   for(i=0;i<16;i++)
   key[i]=a3[i];
   break;
   case 4:
   for(i=0;i<16;i++)
   key[i]=a4[i];
   break;
   case 5:
   for(i=0;i<16;i++)
   key[i]=a5[i];
   break;
   case 6:
   for(i=0;i<16;i++)
   key[i]=a6[i];
   break;
   case 7:
   for(i=0;i<16;i++)
   key[i]=a7[i];
   break;
   case 8:
   for(i=0;i<16;i++)
   key[i]=a8[i];
```

```
    break;
    case 9:
    for(i=0;i<16;i++)
    key[i]=a9[i];
    break;
    case 10:
    for(i=0;i<16;i++)
    key[i]=a10[i];
    break;
    case 11:
    for(i=0;i<16;i++)
    key[i]=a11[i];
    break;
}
cout<<"The shared key is: ";
for(i=0;i<16;i++){
    printf("%d ", key[i]);
}
printf("\n");
//unsigned char message[]="This is a message we will encrypt with AES!";
unsigned char message[100];
i=0;
printf("Enter the message to be encrypted: ");
while((message[i]=getchar())!='\n'){
    i++;
}
message[i]='\0';
//scanf("%s",message);
//unsigned char message[]="This is AES Encryption";
cout<<message;
//unsigned char key[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
//unsigned char key[16]={11,22,33,44,55,66,77,88,99,10,31,27,34,43,59,62};
int originalLen= strlen((const char*)message);
int lenOfPaddedMessage=originalLen;

if(lenOfPaddedMessage%16!=0){
    lenOfPaddedMessage=(lenOfPaddedMessage/16 + 1)*16;
}

unsigned char* paddedMessage = new unsigned char[lenOfPaddedMessage];
for(int i=0;i<lenOfPaddedMessage;i++){
    if(i>=originalLen){
        paddedMessage[i]=0;
    }
    else{
        paddedMessage[i]=message[i];
    }
}
for(int i=0;i<lenOfPaddedMessage;i+=16){
    AES_Encrypt(paddedMessage+i,key);
    }

    cout<<"\nEncrypted message: "<<endl;
    for(int i=0;i<lenOfPaddedMessage;i++){
        PrintHex(paddedMessage[i]);
        cout<<" ";
    }
    getch();
    delete[] paddedMessage;
    return 0;
}
```