### **Defence Research & Development Organisation**

Internship Project: Cryptography

### 

### 

### **SSL:**

SSL stands for Secure Sockets Layer. It’s a technology that helps create a secure and encrypted connection between a web browser (like Chrome or Firefox) and a web server (where a website lives). This ensures that any data sent between you and the website is private and cannot be read or changed by anyone else.

SSL is what makes websites show https:// at the start of their addresses instead of just http:// — the “s” means the connection is secure.

### **HTTPS:**

### HTTPS stands for HyperText Transfer Protocol Secure. It’s an extension of the regular HTTP, which is the protocol your browser uses to load websites. The difference is that HTTPS adds a layer of security using encryption, so the data exchanged between your browser and the website is private and protected from hackers.

### HTTPS uses SSL/TLS (Secure Sockets Layer / Transport Layer Security) to encrypt the communication between your device and the website server. This means:

* When you visit a website starting with https://, your browser first sets up a secure connection using SSL/TLS.
* All the information you send or receive — like passwords, credit card numbers, messages — is scrambled (encrypted) so nobody else can read it if they intercept the data.

### 

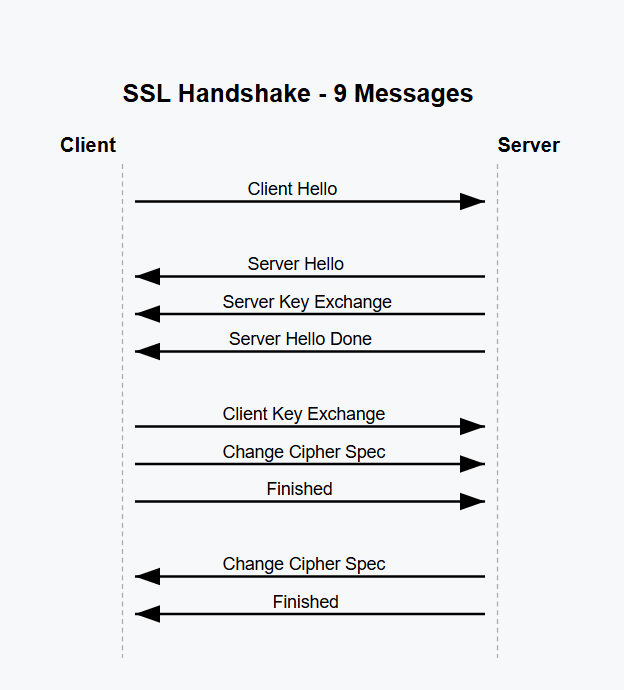
### **Importance of https:**

1. **Data Privacy:** It keeps your information safe from attackers who might try to steal it.
2. **Data Integrity:** It ensures that the data you receive or send is not tampered with or changed by anyone.
3. **Authentication:** It confirms that the website you’re visiting is genuine, not a fake or a scam site.

**Establishes Encrypted Communication:**

When we connect to a website using SSL, several steps happen behind the scenes to create a safe, encrypted link:

1. **ClientHello** The client starts the handshake by sending a ClientHello message with its supported SSL/TLS version, list of cipher suites, compression methods, a random number, and an optional session ID.
2. **ServerHello** The server responds with a ServerHello message, selecting the SSL/TLS version, cipher suite, compression method, and providing its own random number and session ID.
3. **ServerKeyExchange** If needed (e.g., in ephemeral key exchange like DHE/ECDHE), the server sends its public key or key exchange parameters to help the client establish a shared secret.
4. **ServerHelloDone** This message indicates the server has finished its part of the handshake and is waiting for the client’s response.
5. **ClientKeyExchange** The client responds with a ClientKeyExchange message, sending the pre-master secret encrypted with the server’s public key (or using key agreement parameters).
6. **Client ChangeCipherSpec** The client tells the server it is switching to encrypted mode using the agreed cipher suite and session key.
7. **Client Finished** The client sends a Finished message encrypted with the session key. It includes a hash of all previous handshake messages for integrity verification.
8. **Server ChangeCipherSpec** The server also switches to encrypted communication using the same session key.
9. **Server Finished** The server sends its own encrypted Finished message to confirm the handshake was successful and the session is secure.



**Symmetric Key:**Symmetric key cryptography is a method of encryption where the same key is used to both encrypt and decrypt the message. This means that both the sender and the receiver must have access to the same secret key and must keep it private.

### **Working of symmetric key:**

1. The sender uses the key to convert the original message (called plaintext) into an unreadable format (called ciphertext).
2. The receiver then uses the same key to change the ciphertext back into the original message.

### **Key Features:**

* **Efficiency**Symmetric key cryptography is highly efficient, especially when dealing with large volumes of data. It requires fewer resources and processes data faster than asymmetric methods, making it suitable for real-time applications.
* **Simplicity**This method uses just one secret key for both encryption and decryption. The simplicity of having only one key makes it easier to implement and manage in many systems.
* **Security**The security of symmetric cryptography depends entirely on how well the key is protected. If the key is kept secret and shared securely between the sender and receiver, the communication remains safe.
* **Speed**Algorithms like AES (Advanced Encryption Standard) and DES (Data Encryption Standard) are optimized for speed. They are designed to encrypt and decrypt data quickly, making them effective for time-sensitive applications.
* **Symmetry**Since the same key is used on both ends, both parties must have access to the same key before communication starts. This requires a secure way to share the key, which can be challenging, especially over long distances or open networks.

**Asymmetric Key:**

Asymmetric Key Cryptography, also known as public-key cryptography, is a type of encryption that uses two different keys instead of one. These keys are called the public key and the private key.

* The public key is shared openly and used to encrypt (lock) the information.
* The private key is kept secret and used to decrypt (unlock) the information.

Because the keys are different, you don’t need to share the private key with anyone. This makes communication safer, especially over the internet or other insecure channels.

This method is very useful for things like:

* Digital signatures: proving who sent a message.
* Secure key exchanges: safely sharing secret information.

Asymmetric cryptography forms the backbone of many secure communication protocols used today.

### **Key Features:**

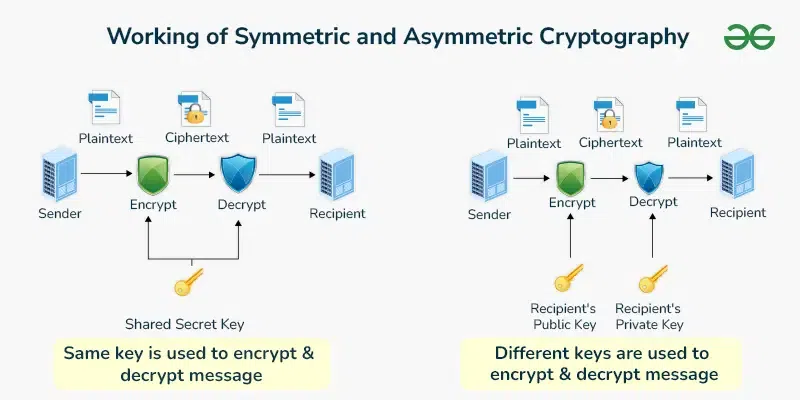
**Key Pair**: It uses two keys—a public key and a private key—to lock (encrypt) and unlock (decrypt) data. This makes it more secure than using just one key.

**Confidentiality**: The public key can be shared with anyone, but the private key is kept secret by the owner. This keeps the information safe.

**Digital Signatures**: It can create digital signatures, which help prove that a message is really from the sender and hasn’t been changed.

**Key Distribution**: Only the public key needs to be shared with others. The private key stays secret, so it’s harder for attackers to steal the secret key.

**Resource Intensive**: It takes more time and computing power compared to other methods (like symmetric cryptography), so it’s not ideal for encrypting very large amounts of data.



Source: www.geeksforgeeks.org

**Data Encryption Standard:**

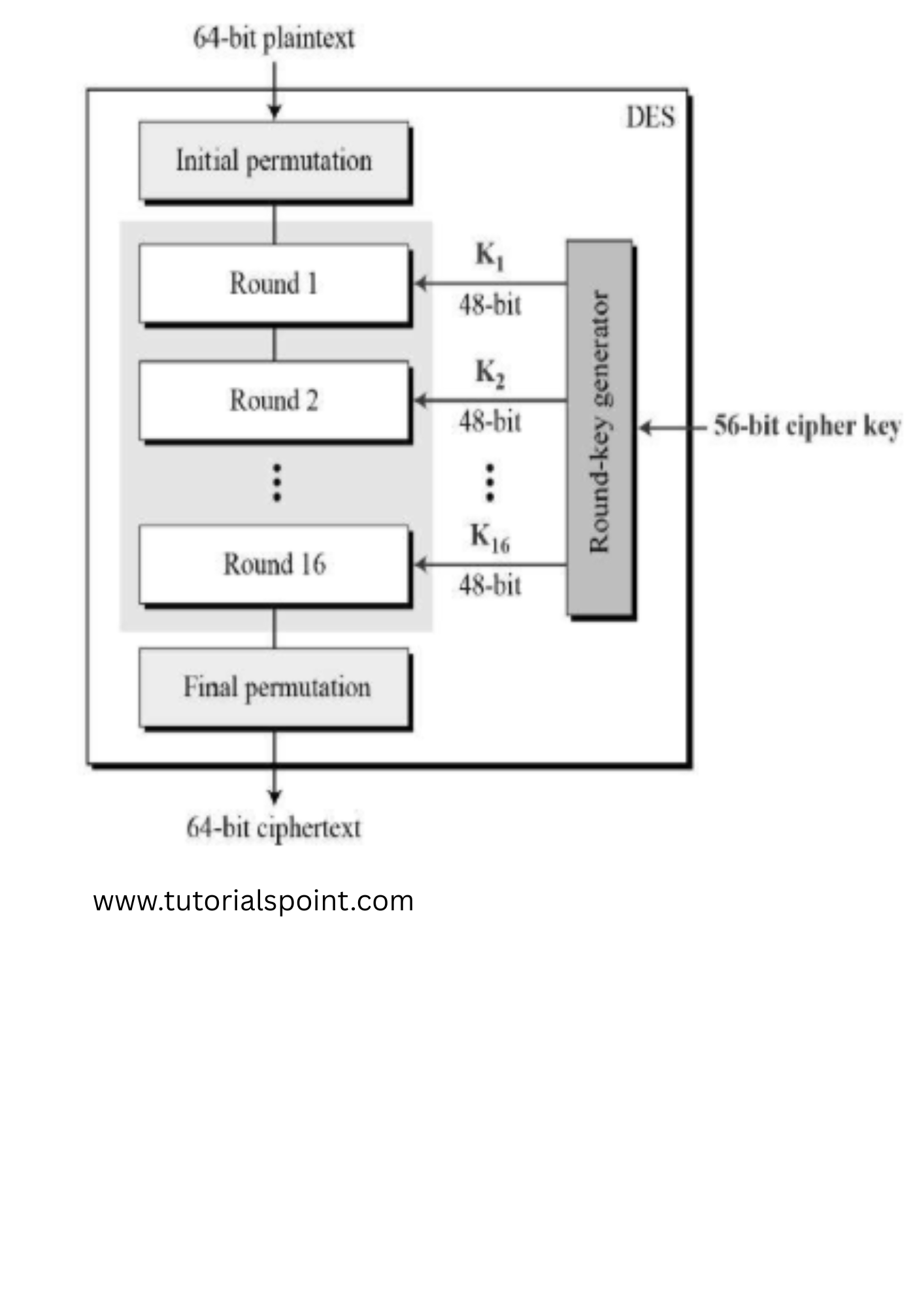
DES (Data Encryption Standard) is a type of symmetric-key encryption — this means it uses the same key for both encrypting and decrypting data. It was developed and published by NIST (National Institute of Standards and Technology) and was widely used for securing data in the past.

### **Key Features of DES:**

1. **Symmetric Key Cipher**:  
    Same key is used for both encryption and decryption.
2. **Block Cipher**:  
    It works on blocks of data, not individual bits or bytes. In DES, each block is **64 bits** (8 bytes) long.
3. **Feistel Structure**:  
    DES is based on something called the **Feistel Cipher**, which divides the data block into two halves and processes them through **16 rounds** of complex transformations (mixing and substitution operations). Each round increases the security.
4. **Key Size**:  
    The key used is **64 bits** long, but only **56 bits** are actually used in the encryption process. The other 8 bits are used for error checking or are just ignored.

**Working of DES:**

1. **Input**: 64-bit block of plain text.
2. **Key**: 56-bit effective key is used.
3. **Rounds**: The data goes through **16 rounds** of encryption, involving shifting, substitution, permutation, and mixing.
4. **Output**: A 64-bit block of encrypted data (ciphertext).



**Advanced Encryption Standard:**

AES (Advanced Encryption Standard) is a symmetric-key encryption algorithm used to secure data. AES is different from DES in the way it processes data:

* It is not a Feistel cipher (like DES).
* Instead, AES uses a method called a Substitution-Permutation Network.

This means encryption is done using a mix of:

* **Substitution**: Replacing data using fixed patterns (like a lookup table).
* **Permutation**: Rearranging the data (like shuffling).

### **Working of AES:**

1. **Block Size:** AES works with 128-bit blocks of data.
   * These 128 bits are grouped into 16 bytes.
   * The bytes are arranged into a 4x4 matrix (4 rows and 4 columns) for processing.
2. **Byte-Level Operations:** Unlike DES, which works on bits, AES does all its processing on bytes, making it more efficient on modern hardware.

AES goes through multiple rounds of encryption steps. The number of rounds depends on the key size:

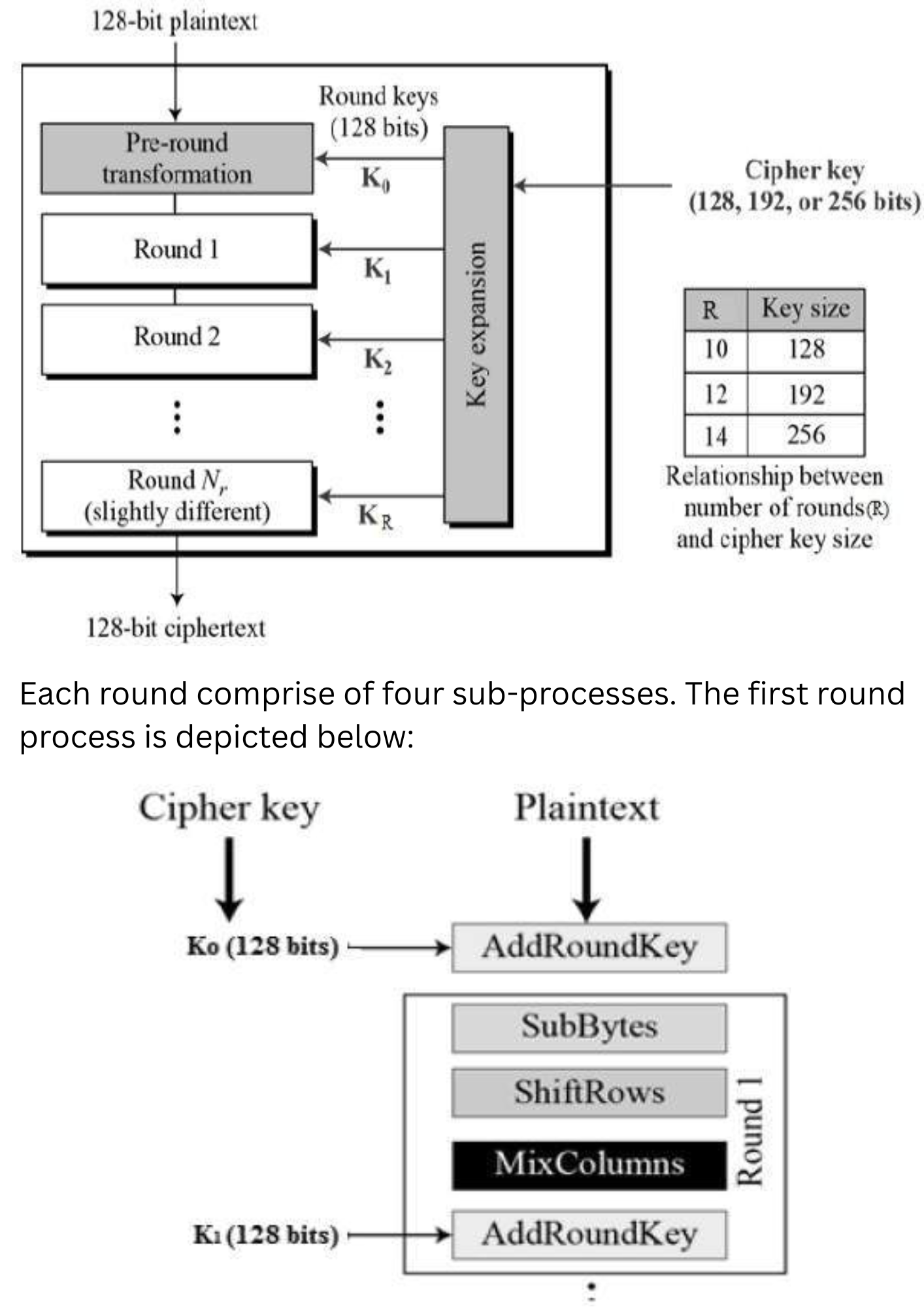
* 10 rounds for a 128-bit key
* 12 rounds for a 192-bit key
* 14 rounds for a 256-bit key

Each round involves several steps, like:

* SubBytes (substitution)
* ShiftRows (row shifting)
* MixColumns (column mixing)
* AddRoundKey (adding a unique round key)

### **Key Expansion:**

* AES creates a different key for each round.
* These round keys (also 128-bit each) are generated from the original AES key using a process called Key Expansion.



In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES have been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of future-proofing against progress in the ability to perform exhaustive key searches.

**Ensuring Data Integrity with Hash Codes:**

A hash value is a fixed-length numerical representation that uniquely corresponds to a specific set of data. Since it condenses large data into a smaller, manageable format, hash values are commonly used in digital signatures for efficiency. Instead of signing an entire dataset, it's faster and more practical to sign its hash. Hashing also plays a crucial role in verifying data integrity, especially when information is transmitted over untrusted networks. By comparing the hash of the received data with the original hash, one can detect any tampering or alterations.

**Key Aspects:**

* **File Integrity Checking**: Our expertise in File Integrity Checking guarantees the integrity of your data. We employ advanced hashing mechanisms to verify file integrity, preventing unauthorized alterations or tampering.
* **Hashing Mechanism Expertise:** Utilizing robust hashing algorithms, we create unique hash values for your data. These values act as digital signatures, ensuring the authenticity and reliability of your files.
* **Automatic Hash Verification:** Our system offers Automatic Hash Verification, allowing seamless checking of active elements against original hash values. This automated process ensures data integrity without manual intervention.
* **Data Integrity Monitoring:** Through Data Integrity Monitoring, we provide continuous surveillance of your files, promptly identifying any discrepancies or unauthorized changes, allowing for immediate corrective actions.
* **File Hash Validation:** Experience precise File Hash Validation, ensuring that files remain unaltered. We employ industry-standard algorithms to generate and validate file hashes, providing a robust defense against data manipulation.

**OBJECTIVE:**

To develop a client which can send and receive messages/files. If two clients are installed on different computers(for testing you can use the same computer but with different ports) then they can communicate to each other by sending messages/files. This can be written in code in Python or any other programming language. Further implement the following functionalities (similar to SSL/TLS Protocol):

1. Encryption and decryption scenario(using standard algorithm AES)
2. Hash (SHA) based integrity check for messages and files.
3. Key exchange mechanism using RSA algorithm
4. Auto discovery of active clients in the network (more oriented towards network programming)

### **Week 1:**

### **Peer-to-Peer Communication and AES-style Encryption**

**Objective:**To implement a peer-to-peer communication system in Python that supports:

* Encrypted message exchange using a symmetric key algorithm (XOR-based AES alternative)
* Secure file transfer
* Communication between two clients on the same computer using different ports

**Tools and Technologies Used:**

* Python 3.x
* socket (for communication)
* threading (for concurrent listening and sending)

**Implementation:**Two clients are set up using different ports on the same machine. Each user enters a shared secret password which is used to derive a symmetric key. The message/file content is then encrypted using a custom XOR-based encryption algorithm (mimicking AES logic in a simplified way). This ensures that the data transferred over the network is not visible in plaintext.

* The encryption function XORs each byte of the data with the password-derived key.
* The same key is used for decryption on the receiver's side.

**Outcome:** The system successfully allowed **secure encrypted communication** between peers.  
 Only clients who knew the shared password were able to **decrypt and read the messages or files**, validating the confidentiality of the communication.

### **Week 2:**

### **Integrity Check Using Custom Hash Function**

**Objective:** To implement an integrity verification mechanism for both messages and files by introducing a custom hash-based check that ensures the data has not been tampered with during transmission.

**Implementation:**

1. Before sending:
   * Compute the hash of the data.
   * Append the hash to the original data.
   * Encrypt the combined content using the XOR-based method.
2. After receiving:
   * Decrypt the message/file.
   * Separate and extract the hash.
   * Recompute the hash on the received content.
   * Compare both hashes:
     + If they match → Accept and display/save content.
     + If they don’t → Display integrity error and discard the data.

**Outcome:**This week’s implementation ensured that any tampering or data corruption during transmission is detected on the receiver side.  
 Users are warned with a clear error message if the hash does not match, which adds a layer of data integrity verification in addition to encryption.

**Code Snippets:**

1. **Custom Hash Function-**

def simple\_hash(data: bytes) -> bytes:

h = [0] \* 16

for i, byte in enumerate(data):

h[i % 16] ^= byte

h[i % 16] = (h[i % 16] + 31) % 256

return bytes(h)

1. **XOR-Based Encryption-**

def simple\_xor\_encrypt(data: bytes, key: bytes) -> bytes:

return bytes([b ^ key[i % len(key)] for i, b in enumerate(data)])

## **Encryption with Hash-**

def encrypt\_with\_hash(data: bytes, key: bytes) -> bytes:

checksum = simple\_hash(data)

combined = data + checksum

return simple\_xor\_encrypt(combined, key)

## **Decryption with Integrity Check-**

def decrypt\_with\_hash(data: bytes, key: bytes) -> bytes:

decrypted = simple\_xor\_encrypt(data, key)

content, recv\_hash = decrypted[:-16], decrypted[-16:]

if simple\_hash(content) != recv\_hash:

raise ValueError("Integrity check failed: Custom hash mismatch.")

return content

1. **Sending a Message-**

def send\_message(sock, key, target\_port):

msg = input("Enter message: ").encode()

enc\_msg = encrypt\_with\_hash(msg, key)

sock.sendto(b"msg" + enc\_msg, ("localhost", target\_port))

print("[SENT] Encrypted message with hash sent.")

1. **Sending a File-**

def send\_file(sock, key, target\_port):

filepath = input("Enter file path: ").strip()

with open(filepath, "rb") as f:

file\_data = f.read()

enc\_data = encrypt\_with\_hash(file\_data, key)

sock.sendto(b"file" + enc\_data, ("localhost", target\_port))

print(f"[SENT] File '{filepath}' sent with integrity hash.")

## **Receiving Messages and Files-**

def handle\_peer(sock, key):

while True:

data, addr = sock.recvfrom(65536)

if data.startswith(b"msg"):

decrypted = decrypt\_with\_hash(data[3:], key)

print(f"[MESSAGE from {addr}] {decrypted.decode()}")

elif data.startswith(b"file"):

decrypted = decrypt\_with\_hash(data[4:], key)

filename = f"received\_file\_from\_{addr[1]}.bin"

with open(filename, "wb") as f:

f.write(decrypted)

print(f"[FILE RECEIVED] Saved as {filename}")

## **Main Function-**

def main():

password = input("Enter shared secret password: ")

key = derive\_key(password)

my\_port = int(input("Enter your listening port: "))

peer\_port = int(input("Enter target peer's port: "))

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind(("localhost", my\_port))

print(f"[LISTENING] on port {my\_port}...")

threading.Thread(target=handle\_peer, args=(sock, key), daemon=True).start()

while True:

cmd = input("Enter command ('msg', 'file', 'exit'): ").strip()

if cmd == "msg":

send\_message(sock, key, peer\_port)

elif cmd == "file":

send\_file(sock, key, peer\_port)

elif cmd == "exit":

print("[INFO] Exiting...")

break

else:

print("[ERROR] Invalid command.")

**REFERENCES:**

1. *Cryptography and Network Security - Principles and Practice (Stallings William)*
2. *SSL and TLS Essential*
3. [*What is Cryptography | why cryptography? Introduction to Cryptography*](https://youtu.be/9X1rSWLFhLY?si=PuxKTod1wUC1aMHa)
4. *https://www.tutorialspoint.com/*

**GITHUB LINK:**

<https://github.com/akshita123454/DRDO_Cryptography_Project>