A PROJECT ASSIGNMENT REPORT ON

Cryptography and it's application in Range EOTS

Submitted by:

Rohan Mahapatra

(Regd no. - 2202070091)

Submitted to:

Dr. Bandan Kumar Bhoi

(Assistant Professor)

DEPARTMENT OF

ELECTRONICS AND TELECOMMUNICATION ENGINEERING

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, BURLA, SAMBALPUR – 768018

Contents

Introduction

Electro-Optical Targeting System (EOTS)

The **Electro-Optical Tracking System (EOTS)** is a critical component used for the **detection, tracking, and recording** of high-speed objects such as missiles, aircraft, or projectiles during flight trials. It uses a combination of optical sensors and electronic systems to observe and analyse the trajectory and performance of flight vehicles in real time.

Key features of EOTS include:

- **High-resolution optical cameras** (daylight and thermal/IR)
- **Precision tracking mechanisms** with auto-tracking capabilities
- **Long-range zoom lenses** for detailed observation
- **Real-time data acquisition and recording systems**
- **Integration with radar and telemetry systems**

Components crucial to EOTS

Component	Function
FLIR (Forward-Looking Infrared)	Thermal imaging for night vision and target identification.
IRST (Infrared Search and Track)	Passive detection of heat-emitting targets (like aircraft) at long range.
Laser Rangefinder/Designator	Measures distance and guides laser-guided weapons with pinpoint accuracy.
Optical Sensors	Provide high-definition visual imagery and magnification of targets.
Image Processing Software	Assists in real-time object recognition, stabilization, and threat analysis.

Range EOTS at ITR, DRDO

At the **Integrated Test Range (ITR), DRDO** in Chandipur, EOTS forms the backbone of **Range Instrumentation Systems**. The **Range EOTS** refers to the strategically deployed electro-optical systems across the missile test range, which are specifically used to track missiles, UAVs, and other test vehicles during flight trials.

Functions and Importance of Range EOTS:

- **Tracking Flight Vehicles:** Provides real-time visual tracking data to complement radar and telemetry.
- **Flight Analysis:** Assists in measuring velocity, acceleration, and orientation of missiles during different phases of flight.
- **Safety Monitoring:** Helps in range safety operations by ensuring vehicles follow designated trajectories.
- **Post-Test Evaluation:** Recorded video footage and data assist in analysing performance and verifying mission objectives.
- **High Precision:** Due to the use of IR and visible spectrum tracking, EOTS can track targets even under low visibility conditions (night or cloudy weather).

Purpose and Role of Cryptography in Test Range Scenario

In missile test ranges like ITR, cryptography ensures the security and integrity of sensitive data such as telemetry, command signals, and video feeds. It protects against unauthorized access, data tampering, and espionage by encrypting real-time transmissions and stored information. Cryptography also verifies the authenticity of users and systems, ensuring that only authorized personnel can access critical test data. This makes it an essential component for maintaining mission confidentiality and national security.

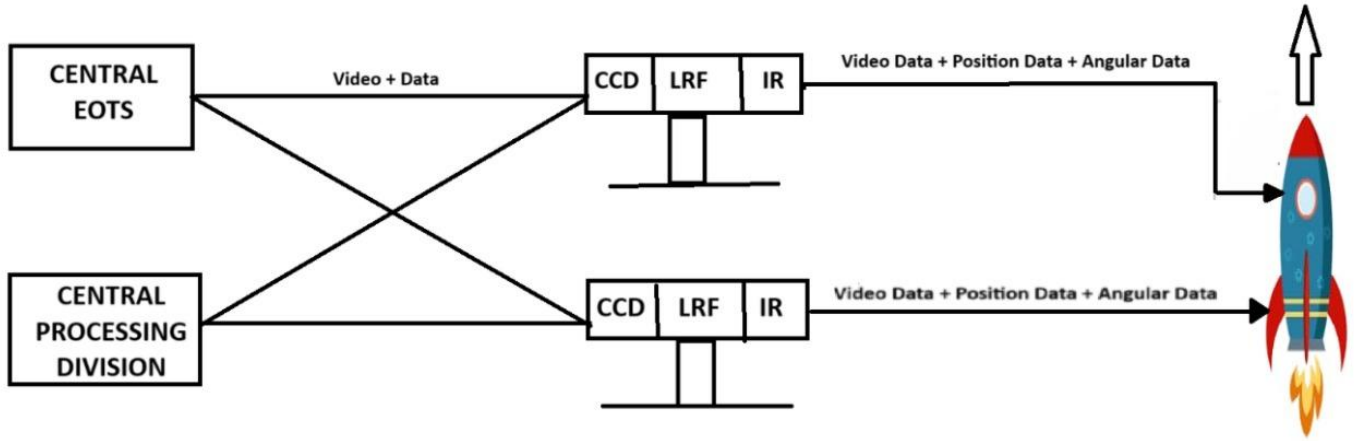


Fig: Range EOTS Operation Scenario

CRYPTOGRAPHY

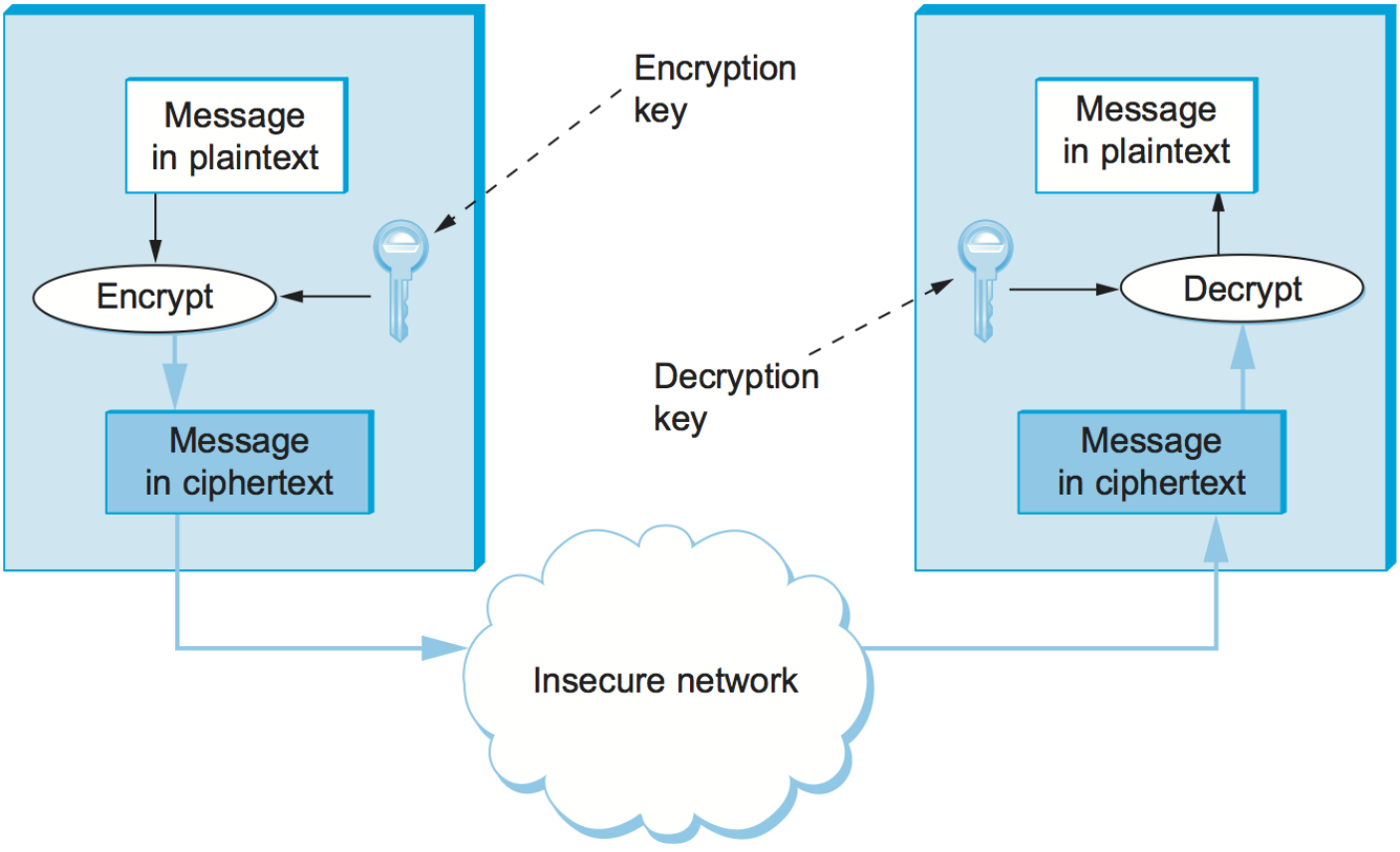
What is Cryptography?

Cryptography is the science of securing information by transforming it into a form that unauthorized users cannot understand. It enables secure communication in the presence of malicious third parties, commonly known as adversaries.

Derived from the Greek words *kryptos* (hidden) and *graphein* (to write), cryptography essentially means “hidden writing.”[1]

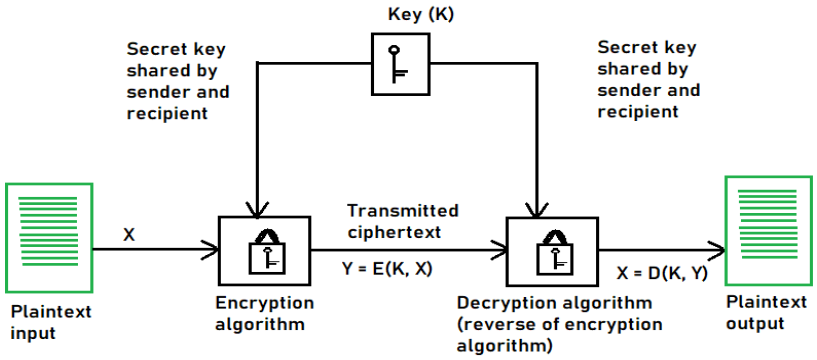
Cryptography, though vital to modern electronic communication, has ancient origins. The earliest known use dates back to around 2000 B.C. in Egypt, where non-standard hieroglyphics were used to conceal messages. Since then, many civilizations with written language have employed some form of secret writing. Notable historical examples include the scytale cipher used in ancient Sparta and the Caesar cipher developed in ancient Rome—both early methods of protecting sensitive information.

While **Cryptography** is the science of secret writing aimed at concealing the meaning of a message, **Cryptanalysis** is the science — and often the art — of breaking cryptographic systems and uncovering hidden information.



Symmetric Cryptography

Symmetric cryptography, also known as secret key cryptography, refers to a method where both parties share the same secret key for encryption and decryption. It is best suited for bulk encryption due to its speed and efficiency compared to asymmetric cryptography.



Types of Symmetric Key Cryptography:

1. Stream Ciphers

2. Block Ciphers

Stream Ciphers:

The encryption process begins with the stream cipher's algorithm generating a pseudo-random keystream made up of the encryption key and the unique randomly generated number known as the nonce. The result is a random stream of bits corresponding to the length of the ordinary plaintext. Then, the ordinary plaintext is also deciphered into single bits. [2]

Block Cipher:

The result of a block cipher is a sequence of blocks that are then encrypted with the key. The output is a sequence of blocks of encrypted data in a specific order. When the ciphertext travels to its endpoint, the receiver uses the same cryptographic key to decrypt the ciphertext blockchain to the plaintext message.

The most common block cipher algorithms are:

1.Advanced Encryption Standard (AES)[3]:

- It has support for three-length keys: 128 bits, 192 bits, or 256 bits, the most commonly used one is a 128-bit key.
- It includes secure communication, data encryption in storage devices, digital rights management (DRM), and so on.

2. Data Encryption Standard (DES):

- In DES, the 64-bit blocks of plaintext are encrypted using a 56-bit key.
- This weakness caused by the small key size led to the development of a more secure algorithm, called AES.

3. Triple Data Encryption Algorithm (Triple DES):

- The development of the Triple DES, also called Triple-DES or TDEA, was triggered by the weak security resulting from the small key size in the DES.

Asymmetric Cryptography

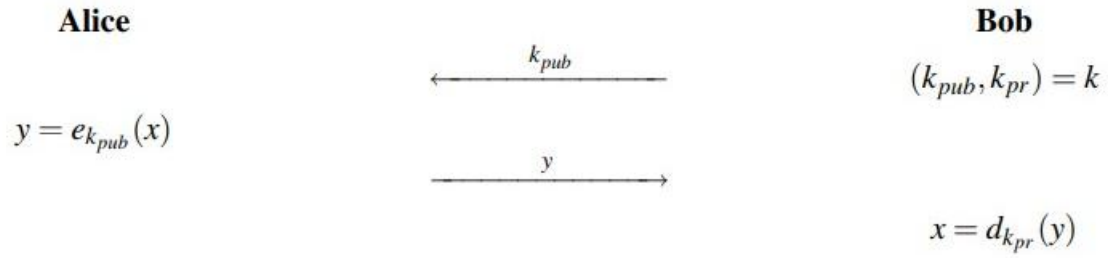
Definition

Asymmetric cryptography, also known as **public-key cryptography**, uses a **pair of keys** — a **public key** and a **private key** — to encrypt and decrypt data. The public key is shared openly and used for encryption, while the private key is kept secret and used for decryption. This eliminates the need to share a single secret key and significantly enhances security, especially for digital communications over open networks. Asymmetric cryptography is widely used in secure email, digital signatures, and online transactions. Common algorithms include **RSA (Rivest–Shamir–Adleman)** and **Elliptic Curve Cryptography (ECC)[6]**.

It is widely used in:

- 🔐 Secure communication
- 📝 Digital signatures
- 🌐 SSL/TLS for secure websites
- 📱 End-to-end encryption in messaging apps
- 📄 Verifying documents and software updates

Here's An Example of Alice and Bob using Asymmetric Cryptography



Bob generates a key pair:

- Public key (kpub) – shared with everyone
- Private key (kpr) – kept secret

Alice wants to send a secret message to Bob:

- She encrypts the message using Bob's public key (kpub)

Bob receives the encrypted message:

- He decrypts it using his private key (kpr)

RSA

In 1977, Ronald Rivest, Adi Shamir and Leonard Adleman proposed a scheme which became the most widely used asymmetric cryptographic scheme called RSA. **RSA** is a **public-key cryptosystem** that is used for **secure data transmission**, especially in applications like secure emails, digital signatures, and online transactions. It is based on the **mathematical difficulty of factoring large prime numbers** — a problem that is easy to verify but extremely hard to reverse.

RSA uses **two keys**:

- A **public key** for encryption (shared openly)
- A **private key** for decryption (kept secret)

Anyone can use the public key to encrypt data, but only the holder of the private key can decrypt it. Before the application of RSA, some prerequisite knowledge on Fermat's(little) Theorem and Euler's Theorem are required[7].

Fermat's (little) Theorem

Fermat's Little Theorem is a fundamental result in number theory. It states:

If p is a prime number and a is an integer such that a is not divisible by p, then:

Let a be an integer and p be a prime, then:

$$a^p \equiv a \pmod{p}.$$

Fermat's Little Theorem is fundamental to public-key cryptography, particularly:

- RSA algorithm (used in modular arithmetic and modular inverses)
- Fast modular exponentiation
- Finding large prime numbers
- Modular inverse calculation (e.g., computing private keys)

Modulo Multiplicative Inverse

The **modulo multiplicative inverse** of an integer **a modulo n** is an integer **x** such that:

$$a \times x \equiv 1 \pmod{n}$$

Euler's Theorem

Euler's Theorem is a generalization of **Fermat's Little Theorem**. It states:

$$\textit{Let } a \textit{ and } m \textit{ be integers with } \gcd(a,m) = 1, \textit{ then:}$$

$$a^{\Phi(m)} \equiv 1 \pmod{m}.$$

Where:

- ϕ(n) is Euler's totient function, which counts the number of integers less than n that are coprime to n.

It is used in **RSA** to compute the **modular inverse** of the public exponent **e**, which helps generate the **private key d**:

$$d \equiv e^{-1} \mod \phi(n)$$

Relationship with Fermat's Theorem

Fermat's Little Theorem is a **special case** of Euler's Theorem when **n** is a **prime number**:

$$a^{p-1} \equiv 1 \mod p \quad \text{(Fermat)}$$

$$a^{\phi(n)} \equiv 1 \mod n \quad \text{(Euler)}$$

ALGORITHM

1. Key Generation

The process of creating a matching pair of public and private keys for secure communication.

1. Choose two large primes p and q.
2. Compute $n = p \cdot q$.
3. Compute $\Phi(n) = (p-1)(q-1)$.
4. Select the public exponent $e \in \{1, 2, \ldots, \Phi(n)-1\}$ such that

$$\gcd(e, \Phi(n)) = 1.$$

5. Compute the private key d such that

$$d \cdot e \equiv \bmod \Phi(n)$$

🔓 **Public Key: (e, n) – shared with everyone**

🔒 **Private Key: (d, n) – kept secret**

2. Encryption

The process of converting a plaintext message into an unreadable ciphertext using the recipient's public key.

1. Convert the plaintext message m into an integer such that:

$$0 < m < n$$

2. Use the public key (e, n) to encrypt:

$$c = m^e \mod n$$

Where:

c is the **ciphertext**

3. Decryption

The process of converting the ciphertext back into the original plaintext using the private key.

$$m = c^d \mod n$$

Where:

c = the encrypted message (ciphertext)

d = private exponent

n = modulus

m = original plaintext message

Implementation of RSA using Python

1. Key Generation

```python
import random, math, json

def str2ascii(st):
    return [ord(c) for c in st]

def is_prime(n):
    if n <= 1: return False
    if n <= 3: return True
    if n % 2 == 0 or n % 3 == 0: return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0: return False
        i += 6
    return True

def generate_prime(bits):
    while True:
        candidate = random.getrandbits(bits)
        candidate |= (1 << bits - 1) | 1  # Ensure it's of proper length and odd
        if is_prime(candidate): return candidate

def generate_keys(bits):
    p = generate_prime(bits)
    q = generate_prime(bits)
    while p == q:
        q = generate_prime(bits)
    n = p * q
    phi = (p - 1) * (q - 1)
    e = random.randrange(3, phi, 2)
    while math.gcd(e, phi) != 1:
        e = random.randrange(3, phi, 2)
    d = pow(e, -1, phi)
    return p, q, e, d, n
```

2. Encryption

```
def encrypt(ascii_list, e, n):
    return [pow(m, e, n) for m in ascii_list]

# --- MAIN LOGIC ---
bit_length = int(input("🔢 Enter bit length for primes (e.g., 16, 32, 64): "))
message = input("✉️  Enter message to encrypt: ")
ascii_msg = str2ascii(message)

p, q, e, d, n = generate_keys(bit_length)
cipher = encrypt(ascii_msg, e, n)

# Save to file
with open("rsa_encrypted_data.json", "w") as f:
    json.dump({
        "ciphertext": cipher,
        "public_key": {"e": e, "n": n},
        "private_key": {"d": d, "n": n},
        "primes": {"p": p, "q": q},
        "original_ascii": ascii_msg
    }, f, indent=2)

# --- Display Info ---
print("\n🔑 Keys Generated:")
print(f"  p = {p}")
print(f"  q = {q}")
print(f"  n = {n}")
print(f"  e (public exponent) = {e}")
print(f"  d (private exponent) = {d}")

print(f"\n🔐 Public Key: (e={e}, n={n})")
print(f"🔓 Private Key: (d={d}, n={n})")

print(f"\n✉️  Original Message: '{message}'")
print(f"📄 ASCII Encoding: {ascii_msg}")
print(f"🔒 Encrypted Ciphertext: {cipher}")
print(f"\n✅ Data saved to 'rsa_encrypted_data.json'")
```

Output

🔢 Enter bit length for primes (e.g., 16, 32, 64): 18

✉️ Enter message to encrypt: NEWPROJECT

🔑 Keys Generated:

p = 164299

q = 235177

n = 38639345923

e (public exponent) = 25962231703

d (private exponent) = 34470784807

🔐 Public Key: (e=25962231703, n=38639345923)

🔐 Private Key: (d=34470784807, n=38639345923)

📨 Original Message: 'NEWPROJECT'

📄 ASCII Encoding: [78, 69, 87, 80, 82, 79, 74, 69, 67, 84]

🔒 Encrypted Ciphertext: [28992306408, 13413407240, 17344499723, 4526239047, 35332784591, 15496763047, 18893364312, 13413407240, 33727538030, 19625484893]

✅ Data saved to 'rsa_encrypted_data.json'

3. Decryption

```python
def ascii2str(ascii_list):
    """Converts a list of ASCII values back into a string."""
    return ''.join(chr(code) for code in ascii_list)

def endecrypt_message(m, key, n):
    """Performs RSA modular exponentiation (used for both encryption/decryption)."""
    result = 1
    m = m % n
    if m == 0:
        return 0
    while key > 0:
        if key % 2 == 1:
            result = (result * m) % n
        key = key >> 1
        m = (m * m) % n
    return result

# --- INPUT SECTION ---
# Input encrypted message
encrypted_str = input("Enter encrypted message (comma-separated integers): ")
encrypted = [int(x.strip()) for x in encrypted_str.split(",")]

# Input private key
d = int(input("Enter private key exponent d: "))
n = int(input("Enter modulus n: "))

# --- DECRYPTION ---
decrypted_ascii = [endecrypt_message(char, d, n) for char in encrypted]
decrypted_message = ascii2str(decrypted_ascii)

# --- OUTPUT ---
print("\n✅ Decryption Successful!")
print(f"Decrypted ASCII: {decrypted_ascii}")
print(f"Original Message: {decrypted_message}")
```

OUTPUT

Enter encrypted message (comma-separated integers): 28992306408, 13413407240, 17344499723, 4526239047, 35332784591, 15496763047, 18893364312, 13413407240, 33727538030, 19625484893

Enter private key exponent d: 34470784807

Enter modulus n: 38639345923

✅ Decryption Successful!

Decrypted ASCII: [78, 69, 87, 80, 82, 79, 74, 69, 67, 84]

Original Message: NEWPROJECT

Comparison between Symmetric and Asymmetric Cryptography

Symmetric cryptography uses the same secret key for both encryption and decryption, making it very fast and efficient for large data. However, sharing the key securely is a challenge. Examples include AES, DES, and Blowfish. In the DES example, after multiple rounds of substitutions and permutations using the secret key, the output was a 64-bit ciphertext (6F6571825C78B1AB), which was then successfully decrypted back into the plaintext (HELLO_W!). The intermediate rounds showed how the data was split into left and right halves and mixed using the key, demonstrating that symmetric encryption is highly effective at producing encrypted blocks and recovering the original message with minimal computational overhead.

Asymmetric cryptography uses a public key to encrypt and a private key to decrypt, so the private key never needs to be shared. This provides better security for key exchange and is often used for digital signatures and SSL/TLS. Examples include RSA, ECC, and DSA. In the RSA example, we used 18-bit primes for demonstration, and the output was a list of large integers (e.g. [28992306408, 13413407240, ...]). The private exponent was also a large number, and decryption successfully recovered the ASCII representation of the message (NEWPROJECT) after substantial computation using modular exponentiation. These outputs highlight the greater mathematical complexity and computational cost of public-key encryption.

Together, both symmetric and asymmetric cryptography yielded distinct types of outputs that highlight their different purposes and efficiencies in securing Range EOTS data. Symmetric encryption produces compact, block-wise encrypted data quickly and efficiently, making it ideal for continuous telemetry, video, or large data streams. In contrast, asymmetric encryption generates larger numeric outputs and is slower, making it more practical for securely exchanging keys or signing critical control messages. By combining these approaches, Range EOTS can establish a secret key securely using RSA, then use DES or another symmetric algorithm to protect large volumes of data at high speed — ensuring robust and practical end-to-end security for all operations.

Conclusion

Cryptography plays a **critical role** in safeguarding sensitive data in **Range EOTS** at test facilities like ITR, DRDO. Both **symmetric** and **asymmetric** encryption techniques contribute to protecting communications, telemetry, video feeds, and control signals. **Symmetric cryptography** enables **fast and efficient encryption** of large volumes of real-time data—ensuring that video streams, sensor readings, and test-range data remain confidential and intact. Meanwhile, **asymmetric cryptography** provides a robust mechanism for **secure key exchange, authentication, and access control**, ensuring that only authorized personnel can decrypt data or send valid commands to range instruments. Together, these methods guarantee the **integrity, confidentiality, and authenticity** of all test-range information, strengthens the security and operational reliability of Range EOTS systems in high-stakes defence environments.

In modern systems, both methods are often used together in a **hybrid approach**. Asymmetric encryption is used to exchange a symmetric session key, and then symmetric encryption takes over to handle the data transfer efficiently. This combination leverages the strengths of both techniques, ensuring both **security and performance**.

As technology advances, especially with the rise of quantum computing and interconnected devices, cryptography will continue to evolve — but its core purpose remains the same: to protect information in a world that increasingly depends on it.

Reference

[1] Christof Paar and Jan Pelzl, *Understanding Cryptography* – A Textbook for Students and Practitioners, First Edition, Springer, pp. 55-75, 2010

[2] William Stallings, *Cryptography and Network Security*

[3] Ganesh Ramani, Niraj Kumar, B K Das, *Comprehensive Multimedia Encryption System Generalization and Comparison*

[4] Alfred Menezes, Paul van Oorschot, and Scott Vanstone, *Handbook of Applied Cryptography*

[5] Bruce Schneier*, Applied Cryptography – Protocols, Algorithms, and Source Code in C,* 2nd Edition, Wiley, pp. 120–145, 1996

[6] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno, *Cryptography Engineering – Design Principles and Practical Applications*, 1st Edition, Wiley, pp. 60–95, 2010

[7] Jean-Philippe Aumasson, Serious Cryptography – *A Practical Introduction to Modern Encryption*, 1st Edition, No Starch Press, pp. 200–235, 2017

[8] Symmetric Key Cryptography - GeeksforGeeks