Experiment – 06

6.1 Aim: To implement data encryption at rest and in transit using cryptographic techniques.

6.2 Course Outcome (CO):

Students will be able to apply cryptographic techniques to ensure data confidentiality during storage (at rest) and transmission (in transit).

6.3 Lab Objective:

To demonstrate how encryption protects data both while stored on disk and while being transmitted over a network.

6.4 Requirements:

- Operating System: Kali Linux or Windows with Node.js installed
- Programming Language: JavaScript (Node.js)
- Libraries: Built-in crypto module in Node.js
- Text Editor or IDE (e.g., VS Code)

6.5 Theory:

Encryption at Rest refers to encrypting data when it is stored on a physical medium (e.g., disk). This protects against unauthorized access if storage is compromised.

Encryption in Transit refers to encrypting data while it is being transferred over a network (e.g., HTTP, FTP, etc.). This protects against eavesdropping and man-in-the-middle attacks.

Public Key Cryptography (RSA) is used here, where:

- A public key encrypts the data (for transit).
- A private key decrypts the data (at rest or upon reception).

6.6 Procedure:

1. Create a new JavaScript file encryption.js.
2. Import the crypto module.
3. Generate RSA key pairs (public and private).
4. **Simulate encryption in transit** by encrypting a message using the public key.
5. **Simulate encryption at rest** by decrypting the message using the private key (as if stored securely).

6. Display the decrypted message to validate the success.

6.7 Output Screenshots:

```
encryption.js  Settings
C: > Users > lab401 > Desktop > encryption.js > ...
1
2  const crypto = require('crypto');
3
4  const { publicKey, privateKey } = crypto.generateKeyPairSync('rsa', {
5    modulusLength: 2048,
6  });
7
8  const message = "This is confidential data.";
9        const encryptedData: Buffer<ArrayBufferLike>
10 const encryptedData = crypto.publicEncrypt(
11   publicKey,
12   Buffer.from(message)
13 );
14
15 console.log("Encrypted Data (in transit):");
16 console.log(encryptedData.toString("base64"));
17
18 const decryptedData = crypto.privateDecrypt(
19   privateKey,
20   encryptedData
21 );
22
23 console.log("\nDecrypted Data (at rest):");
24 console.log(decryptedData.toString());
25
```

```
PS C:\Users\lab401\Desktop> node encryption.js
>>
Encrypted Data (in transit):
pC7icJ3J/+kENAvLj2sd4cNlBfuPedQOlzbkW77v1eFBF8M2ueCaSKInsSu3r3/L8DsvsCnUo9zRRG2dUy17Ps0wGLPATlHlBvG5G/f1+k4h7dXu4nBkCxPlI+HF+mH9KRo8raUHoVZOPmUTYnOJTD4GdqwXUNejlYLCN1p+NnOawRgUXcd
NtYm5xAAV3sLfRoTIk0Ch1GjLNcnplOuT0YJcJ1UtLDcuUeDEf2JpQU0qG6pZ9N7awMPLa45o/upsu5v6TaxHNkEQFK6/ScdUw8a2Gsk3TmO1GyFex0BanPwuNiImDZeEkMdtx1UNg0KrRe2ZCQTsiqd0bEVJGfYhyA==

Decrypted Data (at rest):
This is confidential data.
PS C:\Users\lab401\Desktop>
```

6.8 Conclusion:

This experiment successfully demonstrated the implementation of data encryption both at rest and in transit using RSA cryptographic techniques in Node.js. By encrypting data with a public key during transmission and decrypting it with a private key upon reception or storage, the confidentiality and security of sensitive information were effectively ensured. This highlights the importance of using cryptography to protect data against unauthorized access and cyber threats.