



LORDS INSTITUTE OF ENGINEERING AND TECHNOLOGY
A UGC Autonomous Institution | Approved by AICTE | Recognized by Govt. Of Telangana
Accredited by NBA & NAAC 'A' Grade | Included of 2F of UGC



Department of Computer Science And Engineering

Academic Year: 2026-2027

Student Roll Number:

160923733152

Student Name:

MOHAMMED UFRAAN

Year:

04

Semester:

07

Branch & Section:

CSE-A2
(C)

Course Name:

CRYPTOGRAPHY AND NETWORK SECURITY

Course Code:

U23CS704

Assignment Number:

01

Date of Assignment:

02/08/26

Due date:

15/08/26

Date of Submission:

Marks Obtained:

10

Remarks:

Signature of Faculty with date

Mohammed Ufraan
160923733152



LORDS INSTITUTE OF ENGINEERING AND TECHNOLOGY

(UGC Autonomous)

Approved by AICTE | Affiliated to Osmania University | Estd.2003 | Accredited by NBA & 'A' grade by NAAC

Department of Computer Science Engineering

Academic Year 2026-27

Date: 02-08-2026

B.E. VII-SEMESTER Section CSE -4A2

Course Name: Cryptography and Network Security

Course Code: U23CS704

Assignment – 1

Max.Marks:10

Bloom's Taxonomy Levels (TL)

BTL1 –Remember	BTL2 Understand	BTL3 –Apply
BTL4-Analyze	BTL5 –Evaluate	BTL6 –Create

Answer the following questions:

Q. No.	Question	Marks	Course Outcome	Taxonomy Level
1	Analyze the working of the RC4 stream cipher and discuss the reasons for its security vulnerabilities.	5	CO2	BTL4
2	Analyze the Diffie-Hellman Key Exchange process and explain how a shared secret is established	5	CO2	BTL4

Note:

1. Answers to the assignment questions is to be written on A4 size papers.
2. Each page of the assignment papers should contain full roll number of the student
3. Start writing answers of every question from new page.
4. The hard copy of the assignment is to be submitted on or before 15-08-2026

FACULTY NAME: Ayesha Parveen

Q1.) Analyse the working of the RC4 stream cipher and discuss the reasons for its security vulnerabilities.

Ans: RC4 (Rivest Cipher 4) is a symmetric stream cipher designed by Ronald Rivest in 1987. It was widely used in protocols like SSL/TLS and WEP, but has since been deprecated due to significant security vulnerabilities. RC4 operates on individual bytes of data, making it fast but cryptographically weak by modern standards.

RC4 Architecture:

RC4 consists of two main components:

- (i) Key Scheduling Algorithm (KSA): initialises the internal state using the secret key.
- (ii) Pseudo-Random Generation Algorithm (PRGA): generates keystream bytes used for encryption.

Internal State:

- * RC4 maintains a permutation S of all 256 possible bytes: $S[0], S[1], \dots, S[255]$
- * Two pointers: i (initialised to 0) and j (initialised to 0).
- * Initial state $S[x] = x$ for $x \in [0, 255]$.

WORKING OF RC4:

Step (i): Key Scheduling Algorithm (KSA):

The KSA takes a variable length key (typically 40-256 bits) and permutes the state array:


```
for i = 0 to 255:  
    S[i] = i
```

```
j = 0
```

```
for i = 0 to 255 :
```

```
    j = (j + S[i] + key[i mod key-length]) mod 256  
    swap(S[i], S[j])
```

Purpose: incorporate key material into the permutation to make it dependent on the secret key.

Time Complexity: $O(256) = O(1)$.

Step (2): Pseudo-Random Generation Algorithm.
The PRGA generates the keystream used for encryption:

```
i = 0
```

```
j = 0
```

loop (for each byte of plaintext):

```
    i = (i + 1) mod 256
```

```
    j = (j + S[i]) mod 256
```

```
    swap(S[i], S[j])
```

```
    K = S[(S[i] + S[j]) mod 256]
```

```
    ciphertext-byte = plaintext-byte XOR K
```

Encryption Formula:

$$C_t = P_t \oplus K_t$$

where C_t is ciphertext byte, P_t is plaintext byte, K_t is keystream byte, and $\oplus$ denotes XOR operation.

Example : RC4 Encryption:

Given:

Secret key: KEY (3 bytes)

Plaintext: HELLO

After KSA:

State array is permuted based on key material

PRGA generates keystream:

e.g. [247, 108, 54, 142, 89]

Encryption:

Character	ASCII	Keystream byte	Cipher text
H	72	247	$72 \oplus 247 = 175$
E	69	108	$69 \oplus 108 = 41$
L	76	54	$76 \oplus 54 = 126$
L	76	142	$76 \oplus 142 = 218$
O	79	89	$79 \oplus 89 = 22$

Ciphertext:

[175, 41, 126, 218, 22]

Security Vulnerabilities of RC4:

- (i) Keystream bias in early bytes:
 - * the first few bytes of the keystream exhibit statistical bias and are not uniformly distributed.

- * an attacker can exploit this bias by :
 - collecting many ciphertexts encrypted with same key.
 - statistical analysis reveals patterns in early keystream bytes.
 - recovering plaintext bytes with higher probability than random guessing.
- * mitigation: discard the first 256 bytes of keystream (SSL 3.0 approach), but this only partially mitigates the issue.

(ii) Related-key Attacks:

- * if two keys are related (differ in only a few bits), the resulting keystreams exhibit correlations.
- * keys "KEY" and "KEZ" produce correlated keystreams that can be exploited to recover key.
- * in WEP protocol, RC4 used weak key scheduling where related keys were common, making it vulnerable to key recovery attacks.

(iii) TKIP and WEP flaws:

- * WEP (Wired Equivalent Privacy):
 - used RC4 with 24-bit initialisation vector (IV).
 - IV repeated after 2^{24} packets (≈ 16 million packets).
 - attackers can break WEP in minutes.
- * TKIP (Temporal Key Integrity Protocol):
 - attempted fix for WEP by adding key mixing.
 - still vulnerable to attacks.

(iv) Key Reuse Problem:

- * stream ciphers require unique keys or nonces for each encryption.

$$C_1 = P_1 \oplus K, \quad C_2 = P_2 \oplus K$$

- * if attacker obtains both cipher texts:

$$C_1 \oplus C_2 = P_1 \oplus P_2$$

- * recovery: known plaintext attacks or statistical analysis can recover both P_1 and P_2 .

Cryptography Implications:

- * Authentication Weakness: RC4 does not provide authentication. An attacker can:

- modify ciphertext bits arbitrarily.

- corresponding plaintext bits flip without detection.

- no message authentication code (MAC) built-in.

- * Forward Secrecy: if a single key is compromised, all past and future cipher-texts encrypted with that key are revealed.

Conclusion:

RC4, despite its speed and simplicity, is fundamentally broken due to its biased key-stream output, vulnerability to key reuse, and statistical weaknesses. The deprecation of RC4 reflects the cryptographic community's consensus that security must not be sacrificed for speed alone.

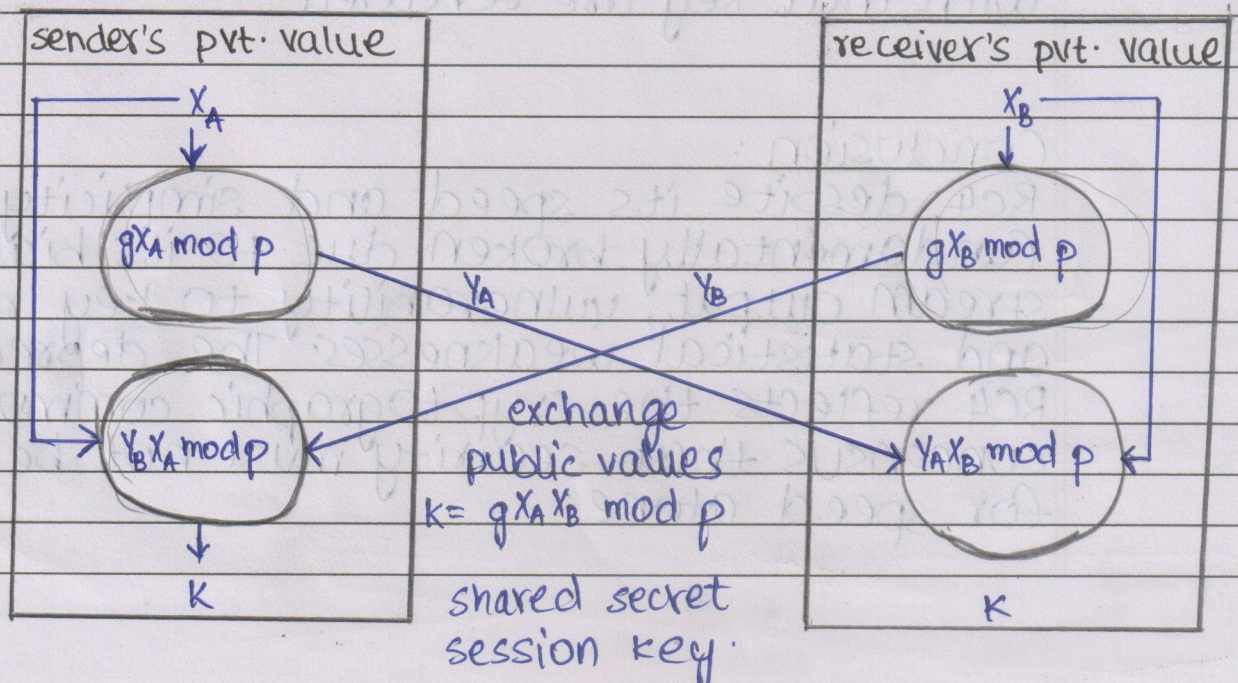
02.) Analyse the Diffie-Hellman Key Exchange process and explain how a shared secret is established.

Ans: The Diffie-Hellman (DH) Key Exchange, proposed by Whitfield Diffie and Martin Hellman in 1976, is a fundamental cryptographic protocol that enables two parties to establish a shared secret over an insecure channel without prior communication. This shared secret can subsequently be used as a key for symmetric encryption.

Scenario:

Alice and Bob wish to communicate securely but:

- * they have never met before.
- * all communication channels are monitored by an eavesdropper.
- * they cannot pre-share a secret key.
- * they need a symmetric encryption key for their conversation.



MATHEMATICAL FOUNDATION:

- The security of Diffie-Hellman relies on the computational difficulty of the Discrete Logarithm Problem:
- Given:
 - * a prime p (typically 2048 bits or larger)
 - * a generator g (primitive root modulo p).
 - * $y = g^x \text{ mod } p$.
- Find:
the exponent x such that $g^x \equiv y \pmod{p}$.

DIFFIE-HELLMAN PROTOCOL:

Step 01:

- * Public Parameters Agreement.
- * Alice and Bob publicly agree on two parameters: p (a large prime number) and g (a generator).
- * Requirement: $g^x \text{ mod } p$ generates all non-zero elements of $\mathbb{Z}_p$ as x varies from 1 to $p-1$.
- * Transmission: both (p, g) can be public knowledge or pre-agreed.

Step 02

- * Alice's secret key generation.
- * Alice generates a random number:
 $a \in \{2, 3, \dots, p-2\}$.
- * Constraint: a must be kept secret and never transmitted.

Step 03

- * Alice computes Public Key:
 $A = g^a \text{ mod } p$.
- * Transmission: Alice sends A to Bob over the channel.

Step 04

- * Bob generates his own random number:
 $b \in \{2, 3, \dots, p-2\}$.
- * Constraint: b must be kept secret and never transmitted.

Step 05

- * Bob computes his public key:
 $B = g^b \mod p$.
- * Transmission: Bob sends B to Alice over the insecure channel.

Step 06

- * Shared secret computation (Alice).
- * Alice receives B and computes the shared secret:
 $K_{\text{Alice}} = B^a \mod p = (g^b)^a \mod p = g^{ab} \mod p$.

Step 07

- * Shared secret computation (Bob).
- * Bob receives A and computes the shared secret:
 $K_{\text{Bob}} = A^b \mod p = (g^a)^b \mod p = g^{ab} \mod p$.

Verification:

Both arrive at same secret

$$K_{\text{Alice}} = B^a \mod p = (g^b)^a \mod p = g^{ab} \mod p$$

$$K_{\text{Bob}} = A^b \mod p = (g^a)^b \mod p = g^{ab} \mod p$$

Therefore:

$$K_{\text{Alice}} = K_{\text{Bob}} = K = g^{ab} \mod p$$

Both parties have successfully established the same shared secret K without ever transmitting it directly.

Numerical Example:

Given parameters:

prime $p = 23$

generator $g = 5$

Step 1

Alice chooses secret
 $a = 6$ (secret).

Step 2

Alice computes public key

$$A = g^a \mod p = 5^6 \mod 23$$

$$A = 15625 \mod 23 = 8$$

Alice sends $A = 8$ to Bob.

Step 3

Bob chooses secret

$b = 15$ (kept secret).

Step 4

Bob computes public key

$$B = g^b \mod p = 5^{15} \mod 23$$

$$B = 30517578125 \mod 23 = 19$$

Bob sends $B = 19$ to Alice.

Step 5

Alice computes shared secret

$$K_{\text{Alice}} = B^a \mod p = 19^6 \mod 23$$

$$K_{\text{Alice}} = 47045881 \mod 23 = 2$$

Step 6

Bob computes shared secret

$$K_{\text{bob}} = A^b \bmod p = 8^{15} \bmod 23$$

$$K_{\text{bob}} = 35184372088832 \bmod 23 = 2$$

Result: Both Alice and Bob have established $K=2$ as their shared secret.

Advantages of Diffie Hellman:

- * if long-term keys are compromised, past session keys remain secure (when used in ephemeral DH mode).
- * multiple parties can establish pairwise shared secrets.
- * works with no prior agreement or trust establishment.
- * based on well-studied number theory.

Vulnerabilities:

- * Man in the Middle attack.
- * Weak Prime Selection.
- * Small subgroup attack.

Modern Applications:

- * TLS 1.3
- * SSH
- * IPsec
- * Elliptic Curve DH (ECDH)

Conclusion :

- * The Diffie-Hellman key exchange revolutionised cryptography by solving the key distribution problem without pre-shared secrets.
- * By leveraging the computational difficulty of the discrete logarithm problem, two parties can establish a shared secret over an insecure channel.
- * The principle of DH - using public information and secret exponents to derive a shared secret - remains central to contemporary cryptographic protocols and continues to ensure secure communication across the internet.