Answers to the Q. NO-1

What shor's algorithm does:

Shor's algorithm: Shor's algorithm is a quantum algorithm that finds the prime factors of Integers and computes discrete logarithms in polynomial time on a fault tolerent concretely.

For RSA, security relies on the classical hardness large factoring integers $N = pq$. shor's algorithm factors N in time polynomial in the numbers of bits on N, so large enough, error corrected quantum computer running shor can recover the private key from the public RSA.

For ECC, security relies on the hardness of the elliptic-curve discrete logarithm problem. Shor's algorithm also solve discrete logarithms on elliptic curves in polynomial time, breaking ECC.

Consequence for current digital infrastructure

- public key systems broken: RSA based signatures / encryption an ECC based TLS, SSH key, code signing, and many authentication systems would be breakable - exposing private keys and allowing decryption or impersonation.
- Retroactive decryption: Adversaries can record encrypted traffic today and decrypt it later.

- PKI and identity systems at risk: Certificate authorities, code-signing keys and software update mechanisms would need replacement / migration.

- Blockchains and crypto currencies: Private keys for wallets (often ECC) could be derived from public addresses - enabling effect unless migrated to quantum-safe schemes.

- Operational disruption: Massive re-keying (TLS, VPNs, IOT devices, embedded systems) would be required; many legacy systems may be hard or impossible to update quickly.

Answer to the Q. No 2

QKD is a method to distribute symmentric secret keys between two parties using Quantum states.

- Information-theoretic detection of eavesdropping

Any measurement by an eaves-dropper disturbs Quantum states in detectable way. after sifting and error checking, participants can bound the eavesdropper's information, or abort.

- Generates symmentric keys, not direct encryption.

QKD produces a shared secret key which is then used in classical symmentric encryption.

• Requires quantum channel + authenticated classical channel:

You need a quantum channel to send qubits and an authenticated classical channel for post-processing. Authentication itself requires a short pre-shared key or a classical public-key signature.

How QKD differs from classical public-key encryption:

• Security model: QKD can provide information-theoretic security assumptions whereas classical public-key cryptography (RSA/ECC) provides computational security.

• Use case: QKD distributes symmetric keys; public-key systems enable key exchange, digital signatures and more flexible authentication without special channel.

• Infrastructure and scale: QKD typically requires point-to-point quantum infrastructure, making global-scale deployment harder than purely classical PKI. Recent research into quantum repeaters may change the range/scale story but practical wide-area QKD networks are still constrained and more expensive.

• Authentication requirement:

QKD cannot by itself provide authentication. it needs an authenticated classical channel.

• post quantum alternative complementarity:

QKD is one approach to secure key distribution in a quantum era. An orthologonal approach is post-quantum cryptography.

Ans to the Q.NO-3

Traditional number theoretic crypto (RSA, ECC):

- Security based on factoring (RSA) or discrete log (ECC) - problems efficiently solved by shor's algorithm on a large quantum computers.

- Many schemes are well-understood, small key sizes (ECC in particular) and fast.

Lattice based cryptography:

- security assumption: hardness of problems like learning with Errors (LWE) or short Integer solution (SIS), these are believed to be hard for both classical and quantum computers.

• worst case to average - case reductions:

Some lattice problems have reductions showing that breaking typical instanse is as hard as solving certain worst-case lattice problems — a strong theoretical advantage.

• performance and parameters:

lattice schemes generally have larger key/ciphertexts than ECC but are often quite practical. They also support advanced features such as efficient homophic encryption, signatures and key exchange.

• Quantum resistance :- Lattice problems have no known polynomial-time quantum algorithms analogous to Shor's. Current evidence points to them being resistant to quantum attacks.

• Maturity and implementation :- Lattice-based cryptography is advancing rapidly - NIST post-quantum selection included lattice schemes. Implementation care is needed.

Lattice-based schemes are the leading practical candidate for quantum-resistant cryptography, whereas number-theoretic approaches (RSA/ECC) are vulnerable to quantum algorithms like Shor's.

Answere to the Q.No-4

```
import time

def custom_prng(seed, count=10):
    rand_nums = []
    value = int(time.time_ns()) ^ seed
    for _ in range(count):
        value = (1103515245 * value + 12345)
                & 0x7fffffff
        rand_nums.append((value, value /
                          0x7fffffff))
    return rand_nums

seed = 20250810
for i, (num, frac) in enumerate(
        custom_prng(seed), 1):
    print(f"{i}: {num} {frac:.10f}")
```

Sample output

1: 2122216093 0.9875421542

2: 767758892 0.3577817086

3: 803183113 0.3739225884

4: 122684714 0.5711527169

5: 5555 80715 0.2588709219

6: 758022336 0.3531886759

7: 3184 70805 0.1180374669

8: 1819851 70 0.0848170861

9: 1013955823 0.1869412351

10: 1602189811 0.1462029325

Answere to the Q.No-5

Algorithm :

- To find all primes less than N.

1. Create a boolean array is_prime [0..N-1] initialized to True, except set is_prime[0] and is_prime[1] to false.

2. For each integer p from 2 to $\sqrt{N-1}$:
   - if is_prime[p] is True, mark every multiple kp (starting at p^2) as False.

3. Remaining True entries are primes.

all smallare multiples.

p.m for m < p. were already marked by smaller primes m.

primes less than 50 (found by sieve):

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

Time complexity comparision:

- Seive of Eratotheres runs in $O(N \log \log N)$ time and uses $O(N)$ memory. The dominant cost is marking multiples, the harmonic-sum analysis yields the log log N factor.

- Trial division costs roughly $O(N \pi N / \log N)$ in naive terms or more precisely $O(\sum_{k \le N} \sqrt{k} \log k)$ - asymptocally much worse than the sieve for large N.

Answer to the Q-No - 6

Korselt's criterion :

An odd composite integer n is a Carmichael number iff:

1. n is square-free
2. for every prime p dividing n, $(p-1)$ divides $(n-1)$

---

① $n = 561$

- Factorization : $561 = 3 \cdot 11 \cdot 17$
- check divisibility:
  - $3-1 = 2$ Does 2 | 560 ? Yes
  - $11-1 = 10$. Does 10 | 560 ? Yes ($560/10 = 56$)
  - $17-11 = 16$. Does 16 | 560 ? Yes ($560/16 = 35$)

$\rightarrow$ 561 is a Carmichael number.

(2) $n = 1105$

Factorization: $1105 = 5 \cdot 13 \cdot 17$

- check:
  - $5-1 = 4$, $1104 \mid 4 = 276$ → divides.
  - $13-1 = 12$, $1104 \mid 12 = 92$ → divides
  - $17-1 = 16$, $1104 \mid 16 = 69$ → divides

→ 1105 is a carcmichael number.

(3) $n = 1729$.

- Factorization: $1729 = 7 \cdot 13 \cdot 19$
- check:
  - $7-1 = 6$, $1728 \mid 6 = 288$ → divides
  - $13-1 = 12$, $1728 \mid 12 = 144$ → divides
  - $19-1 = 18$, $1728 \mid 18 = 96$ - divides

- 1729 is a carcmichael number.

Answer to the Q.No-7

(a) Is the set Z_{11} with $(+, \cdot)$ a ring?

Yes, Z_{11} (integers modulo 11) is a ring under addition and multiplication modulo 11. In fact, since 11 is prime, Z_{11} is a field:

- $(Z_{11}, +)$ is an abelian group.
- Multiplication is associative and distributive over addition.
- Every nonzero element has a multiplicative inverse (because 11 is prime).

So it satisfies the ring axioms.

(b) Are $(Z_{37}, +)$ and $(Z_{35}, \times)$ Abelian groups?

- $(Z_{37}, +)$: Yes. The set $\{0, 1, \ldots, 36\}$ with addition mod 37 is a cyclic abelian group of order 37 (generated by 1). Because 37 is prime is not required for additive group, $(Z_n, +)$ is always a finite cyclic abelian group for any n.

- $(Z_{35}, \times)$: No, as stated. The set Z_{35} with multiplication mod 35 is not a group because many elements are not invertible (0, 5, 7). The multiplicative group of units $Z_{35}^{\times}$ (elements coprime to 35) is an abelian group, but the whole ring Z_{35} under multiplication is not.

Answers to the Q. No-8

We want $-52 \bmod 31$. Compute

$$-52 + 2 \cdot 31 = -52 + 62 = 10$$

So, $-52 \equiv 10 \pmod{31}$. The remainder in the standard range $0, \ldots 30$ is 10.

Answers to the Q. No-9

Solve $7x \equiv 1 \pmod{26}$. Run extended Euclid:

$$26 = 3 \cdot 7 + 5$$
$$7 = 1 \cdot 5 + 2$$
$$5 = 2 \cdot 2 + 1$$
$$2 = 2 \cdot 1 + 0$$

Back-substitute to express 1:

$$1 = 5 - 2 \cdot 2 = 5 - 2(7 - 1 \cdot 5) = 3 \cdot 5 - 2 \cdot 7$$
$$= 3(26 - 3 \cdot 7) - 2 \cdot 7 = 3 \cdot 26 - 11 \cdot 7$$

So,

$-11 \cdot 7 \equiv 1 \pmod{26}$, hence

$x \equiv -11 \equiv 15 \pmod{26}$

Inverse $= 15$. (check: $7 \cdot 15 = 105 \equiv 1 \pmod{26}$.

Answer to the Q. No - 10

Compute product: $-8 \times 5 = -40$

Reduce modulo 17:

$-40 + 3 \cdot 17 = -40 + 51 = 11$

So, $(-8 \times 5) \bmod 17 = 11$.

multiple normally, then reduce to the canonical representative by adding / subtracting multiples of modulus.

alternative convert negative multiplicand

$-8 \equiv 9 \pmod{17}$ then $9 \cdot 5 = 45 \equiv 11 \pmod{17}$

Answer to the Q. No - 11

Bezout's theorem:

For integers a, b not both zero, there exists integers x, y such that

$$ax + by = \gcd(a, b)$$

The Extended Euclidean algorithm computes such x, y. In particular, if $\gcd(a, m) = 1$, then $ax \equiv 1 \pmod{m}$

For the x given by Bezout; thus x is the multiplicative inverse of a mod m

Find inverse of 97 modulo 385:

compute gcd and run extended Euclid:

$$385 = 3 \cdot 97 + 94$$
$$97 = 1 \cdot 94 + 3$$
$$94 = 31 \cdot 3 + 1$$
$$3 = 3 \cdot 1 + 0$$

Back substitute to express 1:

$$\begin{aligned} 1 &= 94 - 31 \cdot 3 \\ &= 94 - 31(97 - 94) \\ &= 32 \cdot 94 - 31 \cdot 97 \\ &= 32(385 - 3 \cdot 97) - 31 \cdot 97 \\ &= 32 \cdot 385 - 127 \cdot 97. \end{aligned}$$

So, $-127 \cdot 97 \equiv 1 \pmod{38}$ Thus the inverse is

$$-127 \equiv 385 - 127 = 258.$$

Inverse $= 258$ (check: $97 \cdot 258 \bmod 385 = 1$)

Answer to the Q. NO-12

Brief proof idea: Extended Euclidean algorithm gives linear integer combination, equal to gcd(a,b) : if gcd(a,b)=1 then it gives $ax+by=1$ hence $ax \equiv 1 \pmod{b}$, so x is the inverse.

compute inverse 43 mod 240:

Euclid:

$$240 = 5 \cdot 43 + 25$$
$$43 = 1 \cdot 25 + 18$$
$$25 = 1 \cdot 18 + 7$$
$$18 = 2 \cdot 7 + 4$$
$$7 = 1 \cdot 4 + 3$$
$$4 = 1 \cdot 3 + 1$$
$$3 = 3 \cdot 1 + 0$$

Back-substitute to express 1

$1 = 67 \cdot 13 - 12 \cdot 210$

hence, $67 \cdot 13 \equiv 1 \pmod{210}$

so, $z = 67$. (check: $13 \cdot 67 = 871 = 12 \cdot 210 + 1$)

Answer to the Q. NO-13

Fermat's little Theorem (statement):

If p is prime and a is not divisible by p, then

$$a^{p-1} \equiv 1 \pmod{p}.$$

It provides a probabilistic primality test, if for some a (coprime to n), $a^{n-1} \not\equiv 1 \pmod{n}$, then n is composite. But passing the test for many a does not guarantee primality (carmichae

numbers are composites that pass $a^{n-1} \equiv 1$ for all a (co-prime to n).

Is 561 prime by test?

561 is composite ($561 = 3 \cdot 11 \cdot 17$). but it is a Carmichael number, many bases a coprime to 561 satisfy

$a^{560} \equiv 1 \pmod{561}$. So Fermat test alone can fail - 561 would appear prime to this test for all coprime bases.

compute $5^{123} \bmod 175$. Since $175 = 25 \cdot 7$ use CRT: compute mod 25 and mod 7 and combine.

- modulo 25: 5^{123} is divisible by 25 for any exponent ≥ 2. So $5^{123} \equiv 0 \pmod{25}$

- modulo 7: $\varphi(7) = 6$; since $\gcd(5,7) = 1$, reduce exponent modulo 6:

$123 \equiv 3 \pmod{6}$. So $5^{123} \equiv 5^3 = 125 \equiv 6 \pmod{7}$

Now solve

$x \equiv 0 \pmod{25}$, $x \equiv 6 \pmod{7}$.

Let $x = 25k$ then $25k \equiv 6 \pmod{7}$.

since $25 \equiv 4 \pmod{7}$

$4k \equiv 6 \pmod{7} \Rightarrow k \equiv 4^{-1} \cdot 6 \pmod{7}$

Inverse of 4 mod 7 is 2, so

$k \equiv 2 \cdot 6 = 12 \equiv 5 \pmod{7}$.

Take $k = 5 \Rightarrow x = 125$.

So, $5^{123} \bmod 175 = 125$.

Answere to the Q. No-11.

CRT (statement): If $m_1, \ldots, m_k$ are pairwise coprime positive integers and you have congruencies

$$x \equiv a_i \pmod{m_i} \quad (i = 1, \ldots, k),$$

then there is a unique solution modulo $M = m_1 \cdots m_k$. Moreover a solution can be constructed via the CRT algorithm.

Solve:

$$x \equiv 2 \pmod 3, \quad x \equiv 3 \pmod 5, \quad x \equiv 2 \pmod 7$$

First solve the first two:

numbers $\equiv 2 \pmod 3$ that are $\equiv 3 \pmod 5$. By inspection $x = 8$ works since $8 \equiv 2 \pmod 3$ and $8 \equiv 3 \pmod 5$:

So $x \equiv 8 \pmod{15}$.

Now enforce.

$x \equiv 2 \pmod 7$. let $x = 8+15k$ we requires.

$8+15k \equiv 2 \pmod 7$.

$-7 \mid 15k \equiv -6 \equiv 1 \pmod 7$.

Since $15 \equiv 1 \pmod 7$. this gives

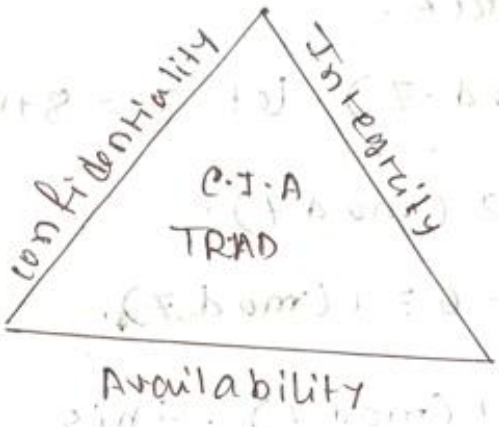
$k \equiv 1 \pmod 7$. choose $k=1 \rightarrow x = 8+15 = 23$.

So, one solution is

$x \equiv 23 \pmod{105}$.

Minimal positive solution: 23.

Answer to the Q.NO-15



C- Confidentiality: Ensure information is accesible only to authorized parties. Method: encryption (TLS, disk encryption), access control, least-privilege; Confidentiality prevents unauthorized disclosure.

I - Integrity: Ensure data is accurate and unmodified except by authorized parties. Methods:

cryptographic hases, digital signatures, message authentication codes (MACs); logging, Integrity prevents tampering and enables detection of unauthorized changes.

A - Availability: Ensure authorized users can access systems, and data when needed. Methods: redundancy, backups, DDoS protection, disaster recovery, load balancing. Availability prevents denial-of-service and ensures continuity.

Answer to the Q. No-16

Aspect	Steganography	Cryptography
Purpose	Hides the existence of the message so no one suspects communication is taking place.	Scrambles the message so it cannot be read without the key.
Visibility	Data is hidden inside another medium (image, audio, video, text) looks like ordinary content	The encrypted message is visible, but unreadable without decryption.
Security Goal	Secrecy of existence	Secrecy of content
Example	Hiding a text message inside an image file so it looks like a normal picture.	Using AES to encrypt a message before sending.

Common steganographing Techniques:

1. Least Significant Bit (LSB) Insertion:
   - Replace the least significant bits of pixel values in an image with message bits.
   - Example: change pixel 11001010 → 11001011

2. Masking and Filtering:
   - Hide data within significant parts of the image (like watermarking) without degrading quality.

3. Transform Domain Techniques:
   - Hide data in frequency components using transforms like Discrete cosine Transform (DCT) in JPEG.

4. Audio steganography:
   - LSB coding in audio samples or phase coding.

Answer to the Q.No-17

Attack Type	Method	Impact
Phishing	Social Engineering attack where fake emails, websites, or messages trick users into revealing sensitive information	Identity theft, Credential compromise, financial loss.
Malware	Malicious software installed on a system without user consent	Data theft, system damage, spying, ransomware lockouts.
DoS/DDoS	Overwhelms a target server or network with excessive requests, making it unavailable	service unavailability, financial loss, reputational damage.

Answer to the Q. No-18

GDPR Overview:

- European Union regulation (enforce in 2018)
- Focus on protecting personal data and privacy rights
- Applies to any organization processing data of EU citizens, regardless of location.

Key provisions:

1. Data Minimization: only collect necessary Data, reducing attack surface.
2. Consent Requirement: Users must agree before data collection.
3. Right to access & Erasure: Users can view and delete their personal data.

4. Data Breach Notification: Organizations must report breaches within 72 hours.

5. Privacy by Design & Default: Security must be integrated from the start.

Answer to the Q.No-19

DES overview:

- Block cipher: process data in 64-bit blocks.
- Key size: 56 bits (originally 64 bits with 8 parity bits removed).
- Number of rounds: 16

DES steps:

1. Initial permutation:
   - Rearranges the 64-bit plaintext bits accordingly to a fixed table.
   - Does not add security, but helps hardware implementation.

2. 16 Rounds of Feistel structure
   - Each round takes L (left 32 bits) and R (right 32 bits).
   - New values:
     - $L_i = R_{i-1}$
     - $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$
   - f-function:
     - Expansion (32 → 48 bits)
     - Key mixing (XOR with round key K_i)
     - substitution using S-boxes
     - permutation.

3. Final permutation:

- Inverse of IP
- Produces ciphertext

Answers to the Q. No - 20

Given:

- $R_0 = 0xF0F0F0F0$
- $K_1 = 0x0F0F0F0F$
- $L_0 = 0xAAAAAAAA$
- Function $f(R_0, K_1) = R_0 \oplus K_1$

Step 1: Compute $f(R_0, K_1)$

$R_0 = F0F0F0F0$ $K_1 = 0F0F0F0F$

XOR each hex byte:

$F0 \oplus 0F = FF$

$f(R_0, K_1) = FFFFFFFF$

Step 2 : compute R_1 :

$$R_1 = L_0 \oplus f(R_0, K_1)$$

L_0 = AAAAAAAA $f(R_0, K_1)$ = FFFFFFFF

XOR :

AA ⊕ FF = 55

R_1 = 55555555

Step - 3 : compute L_1 :

$L_1 = R_0$ = F0F0 F0F0

Final answer :

- L_1 = F0F0 F0F0
- R_1 = 55555555
- First round output : (L_1, R_1)

= (F0F0F0F0, 55555555)

Answere to the Q:NO-20

Given :

- $L_0 = 0XAAAAAAAA$
- $R_0 = 0X\ F0F0F0F0$
- Round key $K_1 = 0X\ 0F0F0F0F$

assumptions :

the round function $f(R_0, K_1)$ is simply the bitwise XOR $R_0 \oplus K_1$ expansion, S-boxes etc.

Step-1 - compute $f(R_0, K_1)$:

$$f(R_0, K_1) = R_0 \oplus K_1$$

Work hex-by-hex :

$F_0 \oplus 0F = FF$

$$f(R_0 K_1) = 0XFFFFFFFF$$

step-2 - compute R_1:

$R_1 = L_0 \oplus f(R_0, K_1) = 0xAAAAAAAAAA \oplus$

$0xFFFFFFFFFF = 0x5555555555$

(AD XOR FF = 55 for each byte)

step-3 - compute L_1:

In the Feistel step $L_1 = R_0$

$L_1 = 0xF0F0F0F0F0$

answer:

$L_1 = 0xF0F0F0F0$, $R_1 = 0x5555555555$

Answere to the Q.No-21

SubBytes :

- SubBytes replaces each byte b of the state by S[b], where S is the AES S-box.
- If b = 0xXY (hex), you can view the S-box as a table with row = X (high nibble) and column = Y (low nibble) Then S[b] = S[X][Y]

Given Input word:

[0x23, 0xA7, 0x4C, 0x19]

We perform subbytes on each byte:

1. Byte = 0x3
   - High nibble 2, low nibble 3.
   - standard AES S-box : S[0x23] = 0x26.

2. Byte = 0xA7

- High nibble A, low nibble 7.
- standard AES S-box: S[0xA7] = 0x5C

3. Byte = 0x4C

- High nibble 4, low nibble C.
- standard AES S-box: S[0x4C] = 0x29

4. Byte = 0x19

- High nibble 1, low nibble 9.
- standard AES S-box: S[0x19] = 0xD4

Final output word:

[0x26, 0x5C, 0x29, 0xD4]

Answer to the Q. No. 22

Given :

- Input word : [0x1A, 0x2B, 0x3C, 0x4D]
- Round key word : [0x55, 0x66, 0x77, 0x88]

AddroundKey = bitwise XOR of corresponding bytes.

1. 0x14 ⊕ 0x55 = 0x4F

(binary : 00011010 XOR 01010101 = 01001111 → 0x4F)

2. 0x2B ⊕ 0x66 = 0x4D

(00101011 XOR 01100110 = 01001101 → 0x4D)

3. 0x3C ⊕ 0x77 = 0x4B

(00111100 XOR 01110111 = 01001011 → 0x4B)

4. 0x4D ⊕ 0x88 = 0xC5

(01001101 XOR 10001000 = 11000101 → 0xC5)

Answer to the Q. No-23

MATRIX (AES MixColumns):

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Input column (given) = $[0x01, 0x02, 0x03, 0x04]^T$

Remainder: in AES GF(2^8) Arithmetic

- multiplication by 1: $1 \cdot x = x$
- multiplication by 2: $2 \cdot x$ = xtime(x) - Leftshift ; if high bit was 1 then XOR with 0x1B.
- multiplication by 3: $3 \cdot x = 2 \cdot x \oplus x$

compute each output byte = row . column (XOR of the products).

1. First row: $02 \cdot 0x01 \oplus 03 \cdot 0x02 \oplus 01$.

0x03 ⊕ 01 · 0x04

· 02 · 0x01 = 0x02

· 03 · 0x02 = (02 · 0x02) ⊕ 0x02 = 0x04 ⊕ 0x02

= 0x06

· 01 · 0x03 = 0x03

· 01 · 0x04 = 0x04

XOR : 0x02 ⊕ 0x06 ⊕ 0x06 ⊕ 0x03 ⊕ 0x01 = 0x04

01 · 0x01 = 0x01

02 · 0x02 = 0x04

03 · 0x03 = (02 · 0x03) ⊕ 0x03 · 02 · 0x03

= xtime (0x03) = 0x06 · 0x03 · 0x03 =

0x06 ⊕ 0x03 = 0x05

· 01 · 0x04 = 0x04

XOR : 0x01 ⊕ 0x04 ⊕ 0x05 ⊕

0x04 = 0x04

Fourth row:

$03 \cdot 0x01 \oplus 01 \cdot 0x02 \oplus 01 \cdot 0x03 \oplus 2 \cdot 0x04$

$\cdot 03 \cdot 0x01 = (02 \cdot 0x01) \oplus 0x01 = 0x02 \oplus 0x01$
$= 0x03$

$\cdot 01 \cdot 0x02 = 0x02$

$\cdot 01 \cdot 0x03 = 0x03$

$\cdot 02 \cdot 0x04 = xtime(0x04) = 0x08$

$X00 : 0x03 \oplus 0x02 \oplus 0x03 \oplus 0x08 = 0x0A$

Output form:

$[0x03, 0x01, 0x09, 0x0A]^T$

Binary 8-bits forms.

- $0X03 = 00000011$
- $0X01 = 00000100$
- $0X09 = 00001001$
- $0X0A = 00001010$

Answer to the Q.No-24

Operation flow:

Sender:

IV → E(K) → O1 → E(K) → O2

XOR

Plain P1 → C1

Send C1, C2 ⟶

Receiver:

(Receiver computes same O1 (O2))

XOR

Plain P2 → C2

Receive C1, C2

Decryption: $P_i = C_i \oplus O_{i-1}$

- Keystream is generated by repeatedly encrypting the previous output:

$$O_0 = IV;\ O_i = E_K(O_{i-1})$$

- Encryption: $C_i = P_i \oplus O_i$
- Decryption: $P_i = C_i \oplus O_{i-1}$

- Synchronization: both sender and receiver must start with the same IV and use the same key and block-cipher encryption function. Because keystream depends only on IV and key, sender and receiver produce identical keystream blocks and so remain synchonized.

- Error behaviour: bits errors in ciphertext only flip corresponding plaintext bits. Loss or misordering of ciphertext blocks breaks synchronization because OFB is synchronous (you must know which keystream block corresponds to which ciphertext block).

Answer to the Q. NO-25

CBC (Cipher Block Chaining):

- Encryption: $C_i = E_K(P_i \oplus C_{i-1})$ with $C_0 = IV$.
- Decryption: $P_i = D_K(C_i) \oplus C_{i-1}$

If a ciphertext block C_i is corrupted:

- P_i becomes essentially random because $D_K(C_i)$ is random like → entire block i corrupted.
- P_{i+1} will have bit-errors equal to the corrupted bits of C_i but D_{i+1} is not fully randomized.
- So a single corrupted block affects two plaintext blocks in decryption.

CFB (Cipher Feedback):

- CFB uses previous ciphertext (or a register with feedback) into the cipher to produce keystream segments.

- Error propagation depends on the segment size: a corrupt ciphertext byte/segment will cause errors for the segments until the corrupted bits exit the shift register. Typically propagation is limited, not indefinite.

Integrity implication: Always use authenticated encryption (AES-GCM or AES-CCM) or a MAC to detect corruption, because models alone do not guarantee integrity.

Answer to the Q.No-26

Best AES mode for encrypting large files with parallel processing is CTR.

CTR is best because:

- **Parallelizable:** Keystream blocks are $E_K(\text{nonce} \| \text{counter})$; each block's keystream can be computed independently, so both encryption and decryption scale horizontally (perfect for large files and multicore/distributed processing).
- **Random access:** We can decrypt any block independently.
- **No pattern leakage:** (unlike ECB) provided the nonce/counter is never reused.

- Low error propagation: Errors affect only corresponding plaintext bits.

Comparison:

- ECB: Parallel but insecure - identical plaintext blocks produce identical ciphertext blocks.
- CBC: Decryption can be parallelized but encryption is sequential. Also has the two-block error propagation behavior described above.
- CTR: gets the best mix of security and parallelism.

Answer to the Q. No-27

Given $M = 1$, $e = 5$, $n = 14$

Ciphertext $C = M^e \mod n = 1^5 \mod 14 = 1$

Decrypt with private $d = 11$:

$M' = c^d \mod n = 1^{11} \mod 14 = 1$

Trivial because $1^k = 1$.

Answer to the Q. No-28

Given message hash

$H(M) = 5$,

Private key $d = 3$, $n = 33$.

Signature $s = H(M)^d \mod n = 5^3 \mod 33 = 125 \mod 33 = 26$.

So, signature = 26.

Answere to the Q. No-29

Given :

prime $P=17$, base $g=3$,

Aleya private $a=4$

Badol private $b=5$

compute public keys :

- Aleya : $A = g^a \bmod p = 3^4 \bmod 17 = 81 \bmod 17 = 13$
- Badol : $B = g^b \bmod p = 3^5 \bmod 17 = 243 \bmod 17 = 5$

Shared secret = 13

$g=3$, $P=17$ (Publi

Aleya : $a=4 \rightarrow A = 3^4 \bmod 17 = 13 \longrightarrow A=13 \rightarrow$ Bad

Badol : $b=5 \rightarrow B = 3^5 \bmod 17 = 5 \longrightarrow B=5 \rightarrow$ Ale

shared secret :

$S = A^b \bmod p = B^a \bmod p = 13$

Answer to the Q.NO-30

Compute H("AB") and H("BA"):

- ASCII : A = 65 , B = 66
- Sum for "AB" : 65 + 66 = 131 → 131 mod 100 = 31
- Sum for "BA" : 66 + 65 = 131 → 131 mod 100 = 31

So, H("AB") = H("BA") = 31.

this implies:

- Two different messages produce the same hash → a collision.
- This function is not collision-resistant. It is trivially easy to find collisions. Good cryptographic hashes must make collisions extremely hard to find.

Answer to the Q.No-31

Given :

message = 15, Secret key = 7.

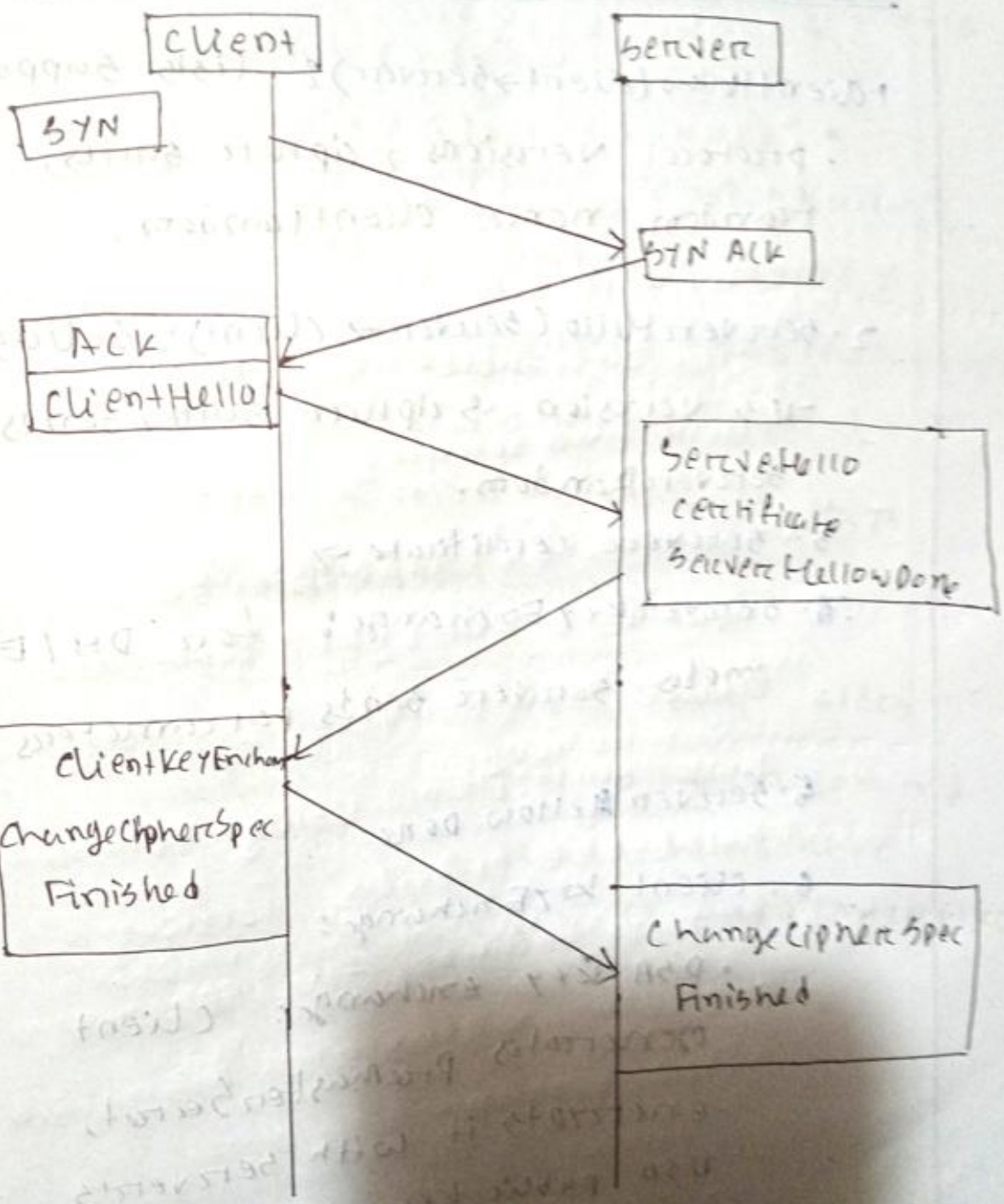
Compute MAC:

- MAC = (15 + 7) mod 17 = 22 mod 17 = 5

Attacker changes message ~~for~~ to 10 but doesn't know key - can they forge a correct MAC?

- Correct MAC for modified message 10 is (10+7) mod 17 = 17 mod 17 = 0.
- Without knowing the key, attacker cannot deterministically compute the proper MAC. However:
  - The MAC construction is too weak: additive, small modulus (17) → attacker can brute-force the key easily or guess a valid pair. Therefore forging is trivial in practice

Answer to the Q.No - 32



High-level handshake (TLS 1.2 typical flow):

1. ClientHello (client → server): lists supported protocol versions, cipher suites, random. nonce clientrandom.

2. ServerHello (server → client): selects TLS version & cipher suite, sends serverRandom.

3. Server certificate →

4. ServerKeyExchange: for DH / ECDH mode, server sends parameters.

5. ServerHelloDone

6. ClientKeyExchange:

- RSA key exchange: Client generates PreMasterSecret, encrypts it with server's RSA public key and sends it.

- (EC)DHE: Client sends its DH public value; the shared secret is computed via DH.

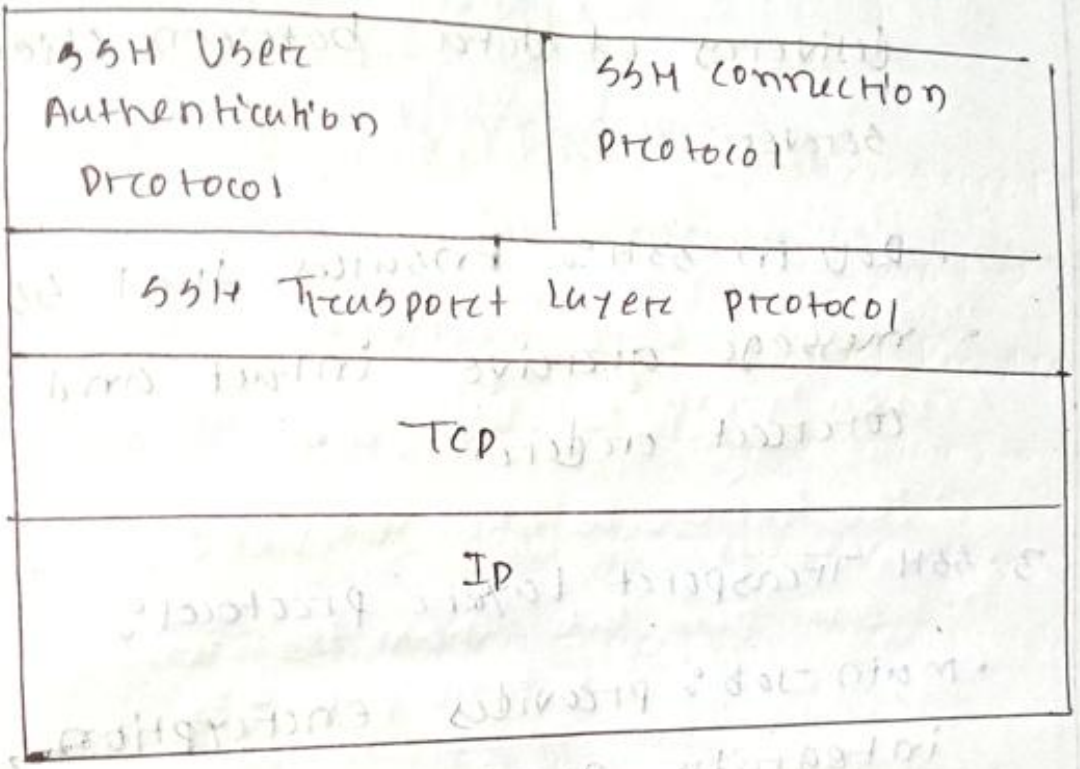
7. Both sides compute Master Secret:
both compute PreMaster Secret →
derive MasterSecret = PRF(PreMaster, Client Random, Server Random) →
derive session symmetric keys from master secret.

8. Change cipher spec: both sides signal switching to the negotiated cipher and verify handshake integrity with Finished messages.

How asymmetric cryptography establishing symmetric key:

- Asymmetric operations (RSA encryption of PreMaster, or ephemeral Diffie-Hellman / ECDHE exchange) securely agree on a short secret known only to client and server.
- That secret (PreMasterSecret) is input to a KDF (PRF) with nonces to derive symmetric key used for bulk encryption. Using ephemeral DH/ECDHE provides forward secrecy because the ephemeral private values are discarded after session.

Answer to the Q. No-33



1. IP Layer:

- Purpose: Handles addressing and routing of data packets across the network.
- Example: Decides where the packet should be delivered (destination IP address)

2. TCP Layer:

- Purpose: Provides reliable, ordered delivery of data between client and server.
- Role in SSH: Ensures that SSH message arrive intact and in the correct order.

3. SSH Transport Layer protocol:

- Main Job: Provides encryption, integrity protection and server authentication.
- Functions:
  - Negotiates cryptographic algorithms
  - Exchanges keys
  - Encrypts all data after the session starts.
  - Authenticates the server to prevent impersonation.

Example: Ensures all data is unreadable to attackers.

4. SSH User authentication:

- Main job: Authenticates the client user to the server.
- Methods:
  - password-based login
  - public key authentication.
  - Multifactor authentication.

Example: verifying you are the legitimate user before allowing access.

5. SSH connection protocol:

- Main job: Manages multiple logical channels over a single SSH connection.

- Functions:
  - Opening terminal sessions (shell).
  - File transfer (SCP, SFTP).
  - Port forwarding / tunneling.
- Example: You can run commands and transfer files securely over the same SSH connection.

Answer to the Q. No - 35

General short Weierstrass form (over prime field F-p):

$y^3 = x^3 + ax + b$ where $a, b \in F_p$

and the curve must be non-singular.

discriminant $4a^3 + 27b^3 \neq 0 \pmod p$

For binary fields (characteristic 2) a different equivalent form is used.

Why elliptic curves are used in cryptography

- The points on the curve form an abelian group under a well defined addition operation.
- The Elliptic curve Discrete logarithm problem (ECDLP) - given points P and $Q = kP$, find k - is believed to be hard. No sub-exponential algorithm is known for properly chosen curves. best generic attacks (pollard's rho) take roughly $O(\sqrt{n})$ time where n is the group order.
- Using elliptic curves yields smaller key sizes for comparable security to RSA or finite-field D.H.

Answer to the Q.No-36

ECC achieve the same level of security as RSA with a smaller key size?

Elliptic - curve cryptography (ECC) gains security form the elliptic curve discrete logarithm problem (ECDLP) which - for well-chosen curves - is much harder per bit than the integer factorization or classical discrete log problems that RSA/DH rely on. concretely:

- For RSA, the best known general attacks have sub-exponential complexity in the key size. To reach a given security level. (that means 128-bit security).

need a very large modulus (≈ 3072 bits.)

- For ECC, the best general attacks are exponential in the size of the group order (e.g., Pollard's rho ~ $O(\sqrt{n})$). As a result a 256-bit elliptic-curve key gives roughly the same (≈ 128-bit) security as a 3072-bit RSA modulus.

Answer to the Q. No-37

Does $P = (3, 6)$ lie on the curve $y^2 = x^3 + 2x + 3 \pmod{97}$?

compute both sides modulo 97:

- Left side: $y^2 = 6^2 = 36$.
- Right side: $x^3 + 2x + 3 = 3^3 + 2 \cdot 3 + 3$
  $= 27 + 6 + 3$
  $= 36$

Modulo 97 both are 36. Therefore $P(3, 6)$ does lie on the curve.

Answer to the p. No-38

Elgamal encryption with public key:

$p=23$, $g=5$, $h=8$! message $m=16$, random $k=6$.

Elgamal ciphertext $=(c_1, c_2)$

$=(g^k \bmod p, m \cdot h^k \bmod p)$

compute step by step modulo 23.

1) $g^k = 5^6$

- $5^2 = 25 \equiv 2 \pmod{23}$
- $5^4 = (5^2)^2 \equiv 2^2 = 4 \pmod{23}$
- $5^6 = 5^4 \cdot 5^2 \equiv 4 \cdot 2 = 8 \pmod{23}$.

So $c_1 = 8$

2) $h^k = 8^6$

- $8^2 = 64 \equiv 18 \pmod{23}$ (Since $64 - 46 = 18$)

- $8^4 = (8^2)^2 \equiv 18^2 = 324 \equiv 2 \pmod{23}$

  (since $324 - 23 \cdot 14 = 324 - 322 = 2$)

- $8^6 = 8^4 \cdot 8^2 \equiv 2 \cdot 18 = 36 \equiv 13 \pmod{23}$

  (since $36 - 23 = 13$)

So, $h_b \equiv 13$.

3) $c_2 = m \cdot h_k \bmod 23 = 10 \cdot 13 = 130 \equiv 15 \pmod{23}$ (since $130 - 23 \cdot 5 = 130 - 115 = 15$)

Final Elgamal ciphertext:

$(c_1, c_2) = (8, 15)$

Answer to the Q. No-39

lightweight cryptography important for securing IoT devices, because:

IoT devices often have severe limits on cpu, memory, power and sometimes bandwidth. Light Weight cryptography provides cryptographic strength but with reduced computational on constrained devices without draining battery or causing latency

Example Algorithm: PRESET or SPECK/SIMON (from NSA). These ciphers use small state/round structure and require less memory and less computation than standard AES implementations, making them suitable

for tiny sensors, RFID tags and low-power MCUs.

Answer to the Q. No - 40

Three common IoT-specific attacks and mitigations:

1. Firmware hijacking / malicious firmware:
   - What: Attackers replace device firmware with malicious code (backdoors, botnet agents).
   - Mitigations: signed firmware and verified boot (secure boot), encrypted firmware updates, strong code-signing key management, and restricting update channels.

2) Physical tampering / side-channel attacks:

- What: Attackers with physical access extract secrets or alter hardware.
- Mitigations: Tamper-resistant packaging, secure elements, disabling debug interfaces in production, and hardware/firmware that zeroes keys on tamper detection.

3) Botnets / credentials based (compromise e.g. Mirai style):

- What: Devices with default/weak credentials get infected and used in distributed attacks.
- Mitigations: Enforce unique default credentials or forced password change at first boot, implement strong authentication, network segmentation, rate limiting and anomaly detection plus timely patching.