③ What are the main differences between lattice-based cryptography and traditional number-theoretic approaches like RSA, particularly in the context of quantum resistence?

Ans: lattice-based cryptography uses hard lattice problems like the shortest Vector Problem (SVP) or learning with Errors (LWE), which remain difficult even for quantum computers.

Differences from RSA:

- RSA is based on integer factorization, while lattice-based schemes use high-dimensional geometry problems.
- Shor's algorithm can break RSA, but lattice-based cryptography is believed to be quantum-resistant.
- Lattice methods often have faster operations but require large keys.

Its quantum resistance makes it a strong candidate for post-quantum cryptography

(5) Explain the sieve of Eratosthenes algorithm and use it to find all prime numbers less than 50. How does its time complexity compare to trial division?

Ans: The Sieve of Eratosthenes is an ancient, efficient algorithm for finding all prime numbers less than a given number n.

Steps:

1. Create a list of boolean values from 2 to n-1, initialy marked as True (meaning all are assumed to be prim)
2. Start with the first prime number p = 2.
3. Mark all multiples of p (starting from p* p) as False (i.e. not prime).
4. Find the next True number in the list and repeat step 3.
5. Continue until p* p > n.
6. All numbers still marked True are prime,

Example: Prime less than 50

Here's a python implementation:

python
copy Edit

```
def sieve_of_eratosthenes(n):
is_prime = [True] * n
is_prime [0:2] = [False, False] #0 and 1 are not prime
for p in range(2, int(n**0.5)+1):
```

Que-23: Show how Mix columns uses the fixed matrix over GF(2^8) to transform column [0x01, 0x02, 0x03, 0x04]

Matrix:

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

Compute each output byte as gf-mul/xor (multiplication by 0x02/0x03 done in Rijndael field with reduction poly 0x11B)

Step results (hex):

- Row 1 = 02.0x01 ⊕ 03.0x02 ⊕ 01.0x03 ⊕ 01.0x04
  = 0x02 ⊕ 0x06 ⊕ 0x03 ⊕ 0x04 = 0x03
- Row 2 = 0x01 ⊕ 0x04 ⊕ 0x05 ⊕ 0x04 = 0x04
- Row 3 = 0x01 ⊕ 0x02 ⊕ 0x06 ⊕ 0x0C = 0x09
- Row 4 = 0x03 ⊕ 0x02 ⊕ 0x03 ⊕ 0x08 = 0x0A

Output column : [0x03, 0x04, 0x09, 0x0A]

- Server Hello Done.
- ClientKey Exchange (premaster via RSA or client DH value)
- (optional) client certificate + certificate verify
- Both compute master secret from premaster and derive session keys.
- Client sends ChangeCipherSpec and finished
- Secure appplication traffic begins under negotiated symmetric keys.

(35) General form of elliptic curve equation over a finite field and why used in cryptography.

Ans: - Over a prime field F_p: $y^2 = x^3 + ax + b \pmod{P}$

with discriminant $4a^3 + 27b^2 \neq (\bmod P)$ to avoid singularities.

Points on the curve plus a point at infinity form an abelian group (point addition / doubling). Why used: The Elliptic curve Discrete logarithm Problem (ECDLP) – given P and $Q = kP$, find k is believed hard. Ecc gives strong security per bit, enabling smaller keys and faster operations than equivalent RSA schemes.

Qus 32: Explain TLS handshake steps and how symmetric keys are established using asymmetric cryptography.

Ans —

- Client Hello: Client sends supported versions chiphersuites. Client random Rc.
- Server Hello: server picks ciphersuits. send server Random Rs.
- Server Certificate: server proves identify by sending certificate containing its public key.
- (optional) Server key Exchange: for ephemeral DH/ECDH server sends params signed by server key.
- Client key Exchange: either client encrypts peramaster secret with server public key (RSA key-exchange) or sends client DH public value (DHE/ECDHE).
- Both compute premaster → master secret
  master = PRF (premaster Rc, Rs)

Qus-27: RSA example: Given message M=1 public key e=5, n=19. Encrypt and decrypt with private d=11.

Ans: Encrypt : $c = M^e \bmod n = 1^5 \bmod 19 = 1$

Decrypt : $m = c^d \bmod n = 1^{11} \bmod 19 = 1$

Ciphertext = 1 (Note: n=19 is not secure-small composite - this is purely illustrative)

Qus-28: RSA signature: Given H(M)=5 private key d=3, n=33. Generate signature.

Ans:

signature $S = H(M)^d \bmod n = 5^3 \bmod 33$

$= 125 \bmod 33 = 26$

signature = 26

Ques-29: Diffie-Hellman example: P=17 g=3, a=4 (Aleya), b=5 (Badol). Compute public keys and shared secret.

Ans:-

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

Q. 20:

In the DES algorithm

Given,

- $R_0 = 0xF0F0F0F0$
- round key $K_1 = 0x0F0F0F0F$
- first round function $f(R_0, K_1)$ assumed to be bitwise XOR (simplified)
- $L_0 = 0xAAAAAAAA$
- $L_1 = R_0$ and $R_1 = L_0 \oplus f(R_0, K_1)$

Compute:

$$f(R_0, K_1) = R_0 \oplus K_1 = 0xF0F0F0F0 \oplus 0x0F0F0F0F$$
$$= 0xFFFFFFFF$$

So,

- $L_1 = R_0 = 0xF0F0F0F0$
- $R_1 = L_0 \oplus f(R_0; K_1) = 0xAAAAAAAA \oplus 0xFFFFFFFF$
$$= 0x55555555$$

Ans: $f(R_0, K_1) = 0xFFFFFFFF$

$L_1 = 0xF0F0F0F0$

$R_1 = 0x55555555$

(9) Develop a Python-based PRNG that uses the current system time and a custom seed value. Write a complete program and corresponding output.

Ans: Below is a complete Python program that implements a Pseudo-Random Number Generator (PRNG) using:

- The current system time (in nanoseconds) for randomness.
- A custom user-provided seed value to ensure repeatability when needed.
- A simple Linear Congruential Generator (LCG) algorithm to generate pseudo-random numbers.

```
· import time
class CustomPRNG:
  def_init_ (self, seed = None):
  # Use time in a nanaseconds as a default seed
   if non provided if seed is None:
    seed = int (time.time_ns()) # system time for
     entroopy
  self.seed = seed
  self.modulus = 2**32
  self.a = 1664525
  self.c = 1013904223
  self.state = seed
  print (f"[INFO] PRNG initialized with seed: {self.seed}")
```

2) Discuss the role of quantum key distribution (QKD) in future cryptographic systems. How does it differ from classical public-key encryption?

Ans: Quantum Key. Distribution (QKD) enables two parties to share encrytion keys securely using quantum mechanics. Any eavesdropping disturbs the quantum states, revealing the intrusion. For example, the BB84 protocol uses polarized photons to detect interception.

Difference from classical public-key encryption: QKD's security is based on the laws of physics, while classical methods (RSA, ECC) rely on hard mathematical problems that quantum computers can solve. QKD offers information-theoretic security and can be combined with post-quantum cryptography to protect data in the quantum era.

· Shared secret $K = B^a \bmod p = 5^a \bmod 17$
$= 625 \bmod 17$
$= 13$ (or $A^b \bmod p$ yields same)

public keys : Aleya $= 13$, Badol $= 5$
shared Secret $= 13$

Ques - 30 : Hash $H(x)$ = (sum ASCII chars) mod 100. Compute H ("A") and H("BA") what does this imply about collision resistance?

Ans :- • ASCII ('A') = 65, ASCII (B) = 66
Sum = 66+65
= 131

• H("A") = 131 mod 100 = 31
• H ("BA") = 66+65 = 131 mod 100 = 31

Both produce same hash (collision).
Implication : The function is not collision resistance many different inputs map to the same short output. A cryptographic hash must make finding collisions computationally infeasible.

```
def next(self):
  # linear Congruential Generator (LCG) formula
  self.state = (self.a * self.state + self.c) % self.modulus
  return self.state
def random (self):
  # Return float in [0, 1)
  return self.next() / self.modulus

# --- Example Usage ---
if __name__ == "__main__":
  # Use system time + custom seed
  custom_seed = int(time.time_ns()) ^ 123456789 # Add extra entropy
  prng = CustomPRNG(seed = custom_seed)
  print("\n--- Generating 5 pseudo_random numbers---")
  for i in range (5):
    print(f" Random # {i+1}: {prng.random()}")
```

Sample output:

```
[INFO] PRNG initialized with seed: 269829578116701295
--- Generating 5 pseudo_random numbers...
Random #1: 0.11248907709219 3561
Random #2: 0.7902038199826181
Random #3: 0.78889302099786096
Random #4: 0.5362985825855732
Random #5: 0.0935393800 9221852
```

How it works

- LCG Formula $X_{+} = (aX + c) \bmod m$ is a standard PRNG algorithm used in early libraries
- Entropy: Time in nanoseconds ensures different output in each run.
- Custom seed: Allows reproducibility if desired.

Qus-38: Given public key ($P=23, g=5, h=8$) and message $m=10$, compute the El Gamal ciphertext using random $K=6$.

Ans:

ElGamal encryption (mod P)

- $C_1 = g^k \bmod P$
- $C_2 = m \cdot h^k \bmod P$

Compute:

- $C_1 = 5^6 \bmod 23 = 15625 \bmod 23 = 8.$
- $h^k = 8^6 \bmod 23 =$ compute $=$ (gives) ?

$\rightarrow 8^2 = 64 \equiv 18,\ 8^4 \equiv 18^2 = 324 \equiv 2,$

then $8^2 \equiv 2 \cdot 18 \equiv 36 \equiv 13.$ so, $h^6 \equiv 13.$

- $C_2 = 10 \cdot 13 \bmod 23 = 130 \bmod 23 = 15.$
- $\therefore$ ciphertext : $(C_1, C_2) = (8, 15).$

Qus-39:

Explain why lightweight cryptography matters for IOT, and give one eample of a lightweight algorithm used in IOT.

Ans:

Why it matters:

IOT devices often have severe constraints - low CPU, little RAM, limited energy, small storage. So full-scale conventional crypto (large-block ciphers, heavy-weight authenticated protocols) may be too slow, power-hungry, Qus-7: Determine whether the following are valid algebraic structures and justify your answer:

- Is the set $\mathbb{Z}$ with operations $+$ and $\times$ a ring

Yes, $\mathbb{Z}$ is a commutative ring with identify.

Are the sets $(\mathbb{R}, +)$ and $(\mathbb{Z}/n\mathbb{Z}, +)$ Abelian group?

Yes, both are Abelian groups:

* $\mathbb{R}$ under addition: associative, identify 0, inverse $-a$, commutative.
* $\mathbb{Z}/n\mathbb{Z}$ under addition mod n: same properties hold

Qus-8: What is the remainder when $73 \times (-14)$ is reduced modulo 15?

Ans: $73 \times (-14) = -1022$

$-1022 \bmod 15 =$ remainder when divided by 15

$-1022 \equiv 8 \pmod{15}$

Answer: 8

or memory-intensive. Lightweight cryptography provides secure primitives tailored to those constraints: lower computational cost, smaller code size, and lower memory use while still offering adequate security for the devices threat model.

Example algorithm:
Ascon (an authenticated encryption and hashing family) — winner in the NIST light weight cryptography project — is designed for constrained devices and offers AEAD functionality. ~~devices and offers AEAD fo~~ with small footprint and good performance on low-end microcontrollers.

Qus-40: List and breifly explain three common IOT-specific attaks and mitigation strategies.

Ans:

1. Firmware hijacking / malicious firmware updates.
   - What: Attacker replaces legitimate firmware with malicious firmware (backdoors, botnet clients).
   - Mitigations: secure boot, digitally signed firmware updates, code-signature verificatio strick update channels, rollback protection, and regular signed update audits.

Qus-33

Explain SSH layered architecture (protocol stack) and roles of each layer.

Ans: • Transport layer (SSH-TRANSPORT): establish encrypted, authenticated channel, negotiates algorithms, provides confidentiality integrity optional compression.

• User Authentication layer (SSH-USERAUTH) authenticates the user over the secure transport (password, public-key, host-based).

• Connection Layer (SSH-CONNECTION): multiplexes channels (shells, exec, port forwarding, subsystems like SFTP) over the authenticated transport.

Each layer builds on the lower: transport secures, userauth authenticates, connection carries application sessions.

Qus-34: Explain the steps involved in the TLS handshake process.

Ans: (concise stepwise recap)

- ClientHello (version, ciphers)
- ServerHello (chosen ciphers)
- Server sends certificate (and ServerKey Exchange if needed).

Qus. 25: Which AES modes cause error propagation during decryption? Illustrate with CBC and CFB and explain integrity impact.

Ans:

· CBC: A single-bit error in ciphertext block C_i causes P_i (after decryption of C_i) to be completely garbled (because $D(C_i)$ is wrong). The subsequent plaintext block P_{i+1} will have a single bit flip at the same bit positions (because $P_{i+1} = D(C_{i+1}) \oplus C_i$). So, corruption affects two plaintext blocks (one fully garbled, the next with bit flips). CBC is malleable: attackers can flip bits in ciphertext to cause predictable changes in decrypted plaintext → integrity is not guaranteed without authentication.

· CFB: Error propagation depends on segment size; a bit error in C_i will directly flip corresponding bits in decrypted P_i, and because of feedback, a limited number subsequent bits/blocks may be affected until the error shifts out of the feedback

2. Physical tampering / device compromise
. What: Attacker with physical accon extracts keys. modifies hardware or installs implants.
· Mitigations: Use tamper-evident / tamper-ressistant enclosure, secure elements or TPMs to protect keys, disable debug ports in production hardware-based key storage (root of trust) and device attestation.
3. Botnets / mass compromis
· What: Default / weak credentials or vulnerable services are exploited at scale to enslave devices into DDOS botnets.
· Mitigations: Enforce strong unique credentials disable unused services / ports, network segmentation, rate-limiting automated patching use of device identify + certificate-based authentication and monitoring / IDS to detect unusual out bound traffic.
Additional general strategies: Least privilege encrypted communications (TLS with proper nonce handling) periodic security audits. Supply-chain security and incident response planning.

Q-21: Use the partial AES s-box to perform sub Bytes on [0X23, 0XA7, 0x4C, 0x19]

Ans: lookups use high nibble = row
low nibble = column

- 0X23 → row 2, col 3 = 0x D9
- 0XA7 → row A, col 7 = 0X63
- 0X4C → row 4, col C = 0X2E
- 0X19 → row 1, col 9 = 0XCb

Resulting output: [0XD4, 0X63, 0X2E, 0x 66]

Q-22: In AES, apply the Add Round key step only. Given Input word [0X1A, 0X2B, 0x3C, 0X4D] and round key [0x55, 0X66, 0X77, 0x88] compute the XOR.

Ans: bytewise XOR:

- $0X1A \oplus 0X55 = 0X4F$
- $0X2B \oplus 0X66 = 0X4D$
- $0X3C \oplus 0X77 = 0X4B$
- $0X4D \oplus 0X88 = 0XC5$

output word: [0X4F, 0X4D, 0X4B, 0XC5]

(1) How does shor's algorithm threaten the security of RSA and Elliptic Curve Cryptography (ECC), and what are the potential consequences for current digital infrastructure?

<u>Ans:</u> Shor's algorithm is a quantum algorithm that can factor large integers and solve discrete logarithms in polynomial time, tasks that are extremely slow on classical computers. This directly threatens RSA, which relies on the hardness of integer factorization, and ECC, which relies on the elliptic curve discrete logarithm problem. If large-scale quantum computers become practical, they could break both RSA and ECC, allowing attackers to recover private keys, decrypt secure communications, and forge digital signatures. This would endanger online banking secure government data, cryptocurrency systems, and the overall public key infrastructure (PK that underpins modern digital security. As a result, there is a pressing need to transition to post-quantum cryptography before such computers arrive.

24: Describe AES_OFB mode and how it ensure synchronization.

Ans: How it works: OFB (output feedback) turns the block cipher into a synchronous stream cipher. Starting from an IV:

$O0 = E(K, IV, O1 = E(K, O0), O2 = (K, O1)$...

Ciphertext $C_i = P_i \oplus O_i$. ~~Decryp~~ Decryption uses the same O_i to recover P_i.

· Synchronization: Both sides must share the same IV and key and process blocks in the same order. OFB keystream depends only on IV and key (not plaintext/ciphertext) so as long as sender and receiver use the same IV and no blocks are lost/inserted keystreams align and decryption succeeds. Loss or reordering of blocks breaks sync until a new IV or explicit resync is used.

register. So, CFB has limited error shifts out of the feedback register. So, CFB has limited error propagation.
Integrity implication: Neither mode provides integrity by it shelf, both need MAC/AE (HMAC or AES-GCM) to detect tampering.

Ques-26: Which AES mode for encrypting large files with parallel processing: ECB, CBC, or CTR? Justify

Recommendation: CTR (counter) mode

Why:
- Keystream blocks are E(K, nonce || counter_i) So, each block encryption/decryption is independent and fully parallelizable.
- No block-pattern leakage like ECB (which is insecure)
- No sequential dependency like CBC (which prevents parallel encryption)
- Error effect is local (no long propagation).

Caveat: use unique/counter per key to avoid reuse.

Qus-31: Mac = (message + secret key) mod 17.

Given message = 15, key = 7. compute MAC.
If attacker changes message to 10 without key, can they forge MAC?

Ans — MAC = (15 + 7) mod 17 = 22 mod 17 = 5

- For message 10, correct MAC would be
  (10 + 7) mod 17 = 17 mod 17 = 0.

Attacker without key cannot compute the correct MAC unless they guess the key, or brute-force (small modulus makes brute force trivial). Thus conceptually secure if key secret and large enough but this construction is insecure in practice (too simple, no cryptographic strength). Use HMAC or other secure MACs.

Qus-36: How does ECC achieve some level of security as RSA with much smaller keys?

Ans: · Different hard Problem: RSA security rests on integer factorization (sub-exponential GNFS algorithms), while ECC security rests on ECDLP (best general attacks like Pollard's rho run in roughly $O(\sqrt{n})$ exponential time)

· Consequence: for equivalent security ECC needs much smaller key sizes (256-bit ECC ≈ 3072-bit RSA).

· Practical benefits: smaller keys/certificate, faster key ~~gey~~ generation/signing, lower storage/communication overhead - useful for constrained devices.

Qus-37:

Given the elliptic curve $y^2 = x^3 + 2x + 3 \pmod{97}$ determine whether the point $P = (3,6)$ lies on the curve.

Ans:

Compute both sides modulo 97:

· Left: $y^2 = 6^2 = 36 \pmod{97}$

· Right: $x^3 + 2x + 3 = 3^3 + 2.3 + 3$

$= 27 + 6 + 3$

$= 36 \pmod{97}$

Since $36 \equiv 36 \pmod{97}$, the equality holds

Yes, $P = (3,6)$ lies on the curve.

```
if is_prime[P]:
  for i in range (p* P, n, P):
    is_prime[i] = False
primes = [i for i, prime in enumerate(is_prime)
    if prime] ~~not~~
 return primes
# Find all prime numbers less than 50
 primes_below_50 = sieve_of_eratosthenes(50)
 print("prime numbers less than 50:",
       primes_below_50)
```

Output :

less
copy Edit
prime numbers Less than 50: [2, 3, 5, 7, 11, 13
17, 19, 23, 29, 31, 37, 41, 43, 47]

Time complexity:

- Sieve: $O(n \log \log n)$ – much faster for larg n.
- Trial division : $O(n\sqrt{n})$ – slower, especially when n is large.

6) State and explain the necessary and sufficient conditions for a composite number to be a Carmicheal number. Then verify whether the number $n = 561$ and $n = 1105$ and $n = 1729$ are Carmichael number?

Ans:— Carmichael Number Condition (Korselt's Criterion):

A composition n is a Carmichael number iff:

1. n is a square-free.
2. For every prime $p \mid n$, $p-1 \mid n-1$.

Check $n = 561$:

$561 = 3.11.17$ (square-free), $n-1 = 560$.

$2 \mid 560, 10 \mid 560, 16 \mid 560 \Rightarrow$ Carmichael.

Check $n = 1105$:

$1105 = 5.13.17, n-1 = 1109$

$4 \mid 1109, 12 \mid 1109, 16 \mid 1104 \Rightarrow$ Carmichael.

Check $n = 1729$:

$1729 = 7.13.19, n-1 = 1728$.

$6 \mid 1728, 12 \mid 1728, 18 \mid 1728 \Rightarrow$ Carmichael

Conclusion: 561, 1105, and 1729 all satisfy the conditions $\Rightarrow$ All are Carmichael numbers.