(200+Q in interview)

No.

Date

PART-3 (Intro to Cryptography)

1.) What is Cryptography ??

⇒ Cryptography is the science of securing data through encoding techiques. It is done to maintain CIA traid and NAAA.

- **Types of Cryptography :-**

① **Symmetric Cryptography :-**

→ Uses a single key for encryption and decryption.

✔ Fast, efficient for large data.

✖ But key distribution is challenging.

- Direction → 2 way
- Speed → Fast, Key used → Same key for E/D.

- **Key Algorithms :-**

① AES :- 128, 192, or 256-bit keys; widely used (eg → SSL/TLS).

② DES :- 56 bit key, outdated.

② **Asymmetric Cryptography :-**

⊙ Uses a public - private key pair.

✔ Secure key exchange, supports digital signature

✖ Slower than symmetric.

• DSA → Signing only, • ECC → faster, smaller key

③ Hashing is irreversible. It have fixed length output. Used for passwords, digital signature

• MD5 - 128 bit, broken,
SHA-1 - 160 bit, broken, SHA-256 - Secure
→ bcrypt → Salting + adpative hashing
(for passwords).

Properties :- ① Deterministic, ② Irreversible
③ Collision - resistant.,
④ Avalanche effect.

⊙ Comparison Table [Encryption v/s Hashing v/s Encoding]

Feature	Encryption	Hashing	Encoding
Reversible	Yes	No	Yes
Key Used	Yes	No	No
Purpose	Confidentiality	Integrity	format readability
Example	AES, RSA	SHA-256	Base 64

PTO.

Use-case :- Password storage, Data integrity checks, Digital signatures.

Security :- Ensures integrity, not confidentiality, vulnerable to collisions if weak (eg) MD5

eg:- Input : "Hello" Output : SHA-256 hash (fixed, eg "a591a6...").

Cryptographic Protocols :-

→ SSL/TLS : Secures web communication
→ IPSec : Secure IP communications (VPNs, tunneling).
→ PGP/GPG : Encrypts emails and files
→ Kerberos : Authentication protocol for networked environments.

Key Managment :-

- Key generation :- Use cryptographically secure random number generators.
- Key distribution :- Securely share keys (eg → Diffie-Hellman for symmetric keys)

Common Cryptographic Defense Mechanisms :-

① Strong Algorithms :- Use robust, well tested algo like AES or RSA/ECC

Purpose → Prevent unauthorized access to sensitive data.

eg: AES-256 for encrypting files, RSA for secure key exchange

Defense against → Brute force, cryptanalysis attacks

---

② Secure Key Management :- Proper generation, distribution, storage, and rotation of cryptographic keys.

Purpose → Ensures key are not compromised maintaining encryption security.

Defense → Key theft, MITM.

③ Hashing for Integrity :- Use strong hash function (SHA-256) to generate fixed length digests of data.

Purpose → Verifies data has not been altered.

Example → Hashing a file to verify its integrity during download.

Defense → Data tampering, collision attacks.

④ Digital Signatures → Use asymmetric cryptography (RSA) to sign data, ensuring authenticity & non-repudiation.

Purpose → Proves the sender's identity and that the message wasn't altered.

eg) Signing software updates to verify they come from a trusted source.

Defense → Forgery, MITM attacks

⑤ Secure Protocols :- Implement protocols like TLS/SSL, IPsec, or SSH to secure communication channels.

Purpose → Protects data in transit across networks.

Eg) TLS in HTTPS for secure web browsing.

Defense → MITM attacks, Eavesdropping.

⑥ Salting for Password Hashing :- Add a unique, random string to password before hashing

Purpose → prevents precomputed attacks.

eg) ~~String~~ storing passwords with bycrypt and unique salts.

Defense → Rainbow table attacks, brute force

⑦ Padding and Initialization Vectors (IVs)

Use proper padding (eg → PKCS#5), and random IVs in encryption to ensure secure cipher text.

Purpose → Prevents pattern in Ciphertext that could aid cryptanalysis.

eg) AES-CBC mode uses an IV to randomize encryption

(8) Side-Channel Attack Mitigation
Protect against attack exploiting physical or environmental leaks.
Purpose :- Prevents attackers from inferring keys or data.
eg> Using constant-time algorithms in cryptographic libraries.
Defense ⇒ Timing attacks, differential Power analysis.

(9) Post-Quantum Cryptography :-
Adopt algorithms resistant to quantum computing attacks.
Purpose → Future-proofs systems against quantum threats like Shor's algo
eg> NIST's ongoing standardization of Post-quantum algo
Defense ⇒ Quantum based attacks

(10) Regular Audits & updates
⇒ defense against outdated or broken algo.

• TYPES OF ① Symmetric Encryption
② Asymmetric Encryption.
③ Hashing.

① Symmetric Encryption Algo's :-

(a) AES (Advanced Encryption Standard) :-
→ Block Cipher, 128-bit block size
→ Key size ⇒ 128, 192, 256 bits
→ Rounds : 10(128), 12(192), 14(256)
→ Security ⇒ Very Strong, resistant to all known attacks.
→ Structure :- ⇒ SubBytes → ShiftRows → MixColumns → AddRoundKey.
→ Used ⇒ in VPN, TLS, WiFi(WPA2), mobile encryption.

(b) DES (Data Encryption Standard) :-
→ Block Cipher, 64-bit block size, 56 bit key, Rounds ⇒ 16
→ Weakness → Brute forced

(c) 3DES (Triple DES) :-
- Applies DES 3 times (EDE or EEE)
- Key Size : 112 or 168 bits
- Block Size : 64 bit
- Status ⇒ Slow, MitM exp.

④ Blowfish :- Block Cipher, 64 bit block, Key size ⇒ 32-448 bit
Rounds : 16
→ Fast, but block size is small → Vulnerable to birthday attacks.
Used in Older SSH, VPN software.

⑤ ChaCha 20 : Stream Cipher, 256 bit key.
• Nonce ⇒ 96 bit, Used in : TLS 1.3, Google services.
⇒ Alternative to AES, better on mobile and IOT devices.

---

② Asymmetric Encryption :-

① RSA (Rivest - Shamir - Adleman)
• Key Sizes : 1024, 2048, 4096 bit
• Used for : Key exchange, digital signatures, encryption.
• Math base : Large prime factorization
• Vulnerable : Quantum attacks.
• Used in : TLS, PGP.

② ECC (Elliptic Curve Cryptography)
→ Shorter keys, same security (eg) 256 bit ECC ≈ 3072-bit RSA).
→ faster, lower CPU usage.
→ used in : TLS, Bitcoin, Signal
→ Popular Curves : secp256k1 (Bitcoin), Curve 25519.

③ DSA (Digital Signature Algorithm)
⇒ Used only for signature, not encryption. Key size = 1024-3072 bits
→ Based on : Discrete log problem,
→ Replaced by : ECDSA

④ Diffie Hellman :- Key Exchange protocol, not used for encryption directly
• Shared secret over insecure channel.
Problem Base : Discrete logarithm
Vulnerable to : MiTM

⑤ ElGamal :- Asymmetric encryption system & Probabilistic encryption.
• Based on - Diffie - Hellman
• Not widely use due to large ciphertext sizes

- Hashing Algo :-

① MD5 :- Output 128 bit hash,
not, but broken,
Don't use in production
used in : check sums

② SHA-1 :- Secure hash algo 1

output : 160 bit,
weak - known collisions
Deprecated by :- NIST
Avoid in security system.

③ SHA-256 :- Output : 256-bit
very secure, used in TLS,
Bitcoin, digital signatures
→ Recommended for most secure
applications

④ bcrypt :→ Hashing algo for passwords
→ uses salting + key
stretching
→ slow by design
→ output → 192 bit
→ Automatic salting included in output.

⑤ Scrypt :-
- Memory - hard - resists GPU attacks
- Better for password hashing
than bcrypt
→ used in : cryptocurrency wallets,
secure password storage

⑥ PBKDF2 :-

- Key stretching using HMAC (SHA-1/SHA-2)
- customizable iteration
→ NIST - approved for password
hashing

⊙ Digital Signatures :-

→ Cryptographic mechanism to verify the authenticity + integrity of digital data.

→ Provides : ① Authentication : Private key of Sender.
② Integrity : Hash of original message.

③ Non-repudiation :- Signature proves sender's intent.

▸ Working Process :

1.) Message Digest Generation → Hash the original message using SHA-256, etc.
→ Output : Fixed-length digest (fingerprint).

2.) Signing :- ⓐ Encrypt the digest using Sender's private key.
ⓑ Output : ~~Fixed-length~~ This become the digital signature.

3.) Send :-
→ Message + Digital Signature → Sent to receiver.

④ Verification by Receiver :-

ⓐ Hash the received message again
ⓑ Decrypt the signature using Sender's Public key.
ⓒ Compare both hashes → if match ⇒ Valid

▸ Algorithms :

Common algo includes RSA, which uses a pair of large prime number for encryption; DSA, a U.S. government std; and ECDSA, which offers similar security with smaller key sizes.

▸ Security :-

Security hinges on the confidentiality of the private key. If compromised, the signature can be forged. Secure key exchange is critical, often managed by PKI. Quantum computing poses a future threat, as it could potentially break current algo like RSA, DSA, necessitating post-quantum cryptography development.

P10.

PKI [Public Key Infrastructure]

- **Purpose** → PKI is a framework that manages public key encryption and digital certificates to enable secure communication over untrusted networks. It ensures that public keys are authentic and associated with correct entities, facilitating secure data exchange, authentication, and confidentiality.

- **Components :**
  - **Certificate Authority (CA) :-** A trusted entity that issues digital certificates, verifying the identity of the certificate holder.
  - **Registration Authority (RA) :-** Act as a verifier, checking identity details before a CA issues a certificate.
  - **Digital Certificate :** Electronic documents that bind a public key to an individual, organization, or device, signed by a CA.
  - **Key Pairs :-** Each entity has a public key and private key, used together for encryption and decryption.

- **Applications :-** PKI is widely used in SSL/TLS for securing websites (HTTPS), VPNs for private network access, and encrypted email systems like S/MIME. It also supports digital signatures and secure software distributions.

- **Security :** PKI's effectiveness depends on the trustworthiness of CAs; a compromised CA can issue fraudulent certificates. Key management is crucial, with challenges in distribution and revocation. Regular updates and audits of CA practices are essential to maintain security.

PKI (Key management, Key generation, Storage, lifecycle and Security consideration).

- **Key Management :-** ① **purpose** :→ Ensures the secure handlings of cryptographic keys throughout their lifecycle, critical for maintaining the integrity and security of PKI system.

⊙ Key Generations :

① Process → Keys are generated using secure random number generators :
For eg) RSA keys require large prime numbers, while ECC uses elliptic curve parameters.

② Standards → Follows guidelines like NIST SP 800-133 for secure key generation to prevent predictability or weakness.

③ Importance :→ Weak generation can lead to key compromise.

⊙ Key Storage :-

① Methods :- Keys are stored in secure locations such as Hardware security Modules (HSMs), trusted platform modules (TPMs) or encrypted files. Private keys must be kept confidential, often protected with passwords or biometric authentications.

▸ Best Practices :- Use strong encryption for stored keys and avoid unencrypted storage or vulnerable

▸ Challenges : Balancing accessibility for authorized use with protection against unauthorized access.

▸ Lifecycle (Creation, Distribution, Revocation) :→

• Creation → Involves generating key pairs during initialization, often tied to certificate issuance by a CA. Ensures keys meet required strength (eg → 2048 bit RSA).

• Distribution :→ Securely shares public keys via certificates, while private keys remain with the owner. Uses secure channels (eg TLS) or physical media to prevent interception.

• Revocation :→ Occurs when keys are compromised or no longer needed. Managed via Certificate Revocation Lists (CRLs) or Online ↓ Status Protocol (OCSP), where the CA invalidates the certificate, ensuring systems no longer trust the key.

- Security Considerations :- Proper key managment prevents unauthorized access, ensures key usability and mitigates risks from lost or expired keys. Regular rotation and secure backup are recommended practices.

SSL / TLS :

Definition :→ • SSL (Secure Sockets Layer): Outdated cryptographic protocol for securing communication over the internet.

• TLS → (Transport Layer Security) Modern, secure replacement for SSL; encrypts data, ensures integrity and authentication

Purpose → Provides CIA in data transmission
- Used in HTTPS, Email (SMTP, IMAP), VPNs, and VoIP.

- Comparison between SSL & TLS :-

T-10.

Feature	SSL	TLS
Version	SSL 2.0, 3.0	TLS 1.0 - 1.3
Status	Deprecated	TLS 1.2 / 1.3 in use
Security	Poor	Very strong
Handshake	More round trips	Fewer in TLS 1.3.

- TLS - Handshake (TLS 1.2 Simplified) :-

1. Client Hello : Sends supported TLS version, cipher suites, random value.
2. Server Hello : Sends selected cipher, certificate, random value.
3. Certificate Validation : Shared secret generated via RSA/DH/ECDHCE.
4. Key Exchange : Shared secret generated via RSA / DH / ECDHCE.
3. Certificate Validation :- Client verifies server cert via CA.
5. Session keys generated
6. Finished Messages encrypted with session key.

- Components :

- Certificates :- Digital certificates (eg → X.509) issued by CAs containing public keys and entity details.

- Cipher suites → Combination of algorithms for encryption, key changes and hashing.

- Protocols :- Defines the handshake and data transfer rules.

⊙ Applications :→
→ Secures websites (HTTPS), email (IMAPS), VPNs & online transactions
→ widely used in e-commerce, banking & Cloud services.
→ Facilitates secure file transfers for sensitive data exchange.

⊙ Security Features :-

- CIA is provided

Key Managment
- Uses PKI for certificate issuance and renewal.

- Session keys are generated per connection and discarded after use.

- Revocation handled via CRLs or OCSP if certificates are compromised.

- Common Attacks on SSL/TLS :-

① BEAST (Browser Exploit Against SSL/TLS)
→ Exploits weaknesses in CBC mode of TLS 1.0, allowing data decryption.

② Heartbleed :- A 2014 bug in OpenSSL exposing memory, including private keys.

③ MiTM

④ DROWN (Decrypting RSA with Obsolete and Weakened Encryption) :→
Uses SSLv2 to break TLS connections

PTO

▸ Digital Certificate & Certificate Authority (CA)

① **Digital Certificate** :- An electronic document that binds a public key to an identity, issued by a trusted entity, verifying authenticity.

▸ **Purpose** : ① **Authentication** ⇒ Verify the identity of users or device.
② **Integrity** ⇒ Ensures no data alteration.
③ **Encryption** :⇒ Supports secure data transmission (eg → via TLS).

▸ Components of a Digital Certificate :
① **Public key** : Used for encryption / verification.
② **Subject (Owner Identity)** :- Details of the certificate holders.
③ **Issuer (CA - Identity)** :- The Certificate Authority that issued it.
④ **Serial Number** :- Unique identifier for the certificate.
⑤ **Validity Period** : Start and end dates of certificate usability.
⑥ **Signature (CA's Digital Signature)** : Ensures Certificate authenticity.

Certificate Authority [CA] :-

Definition :- A trusted organization that issues and manages digital certificates.

- Role :- Verifies identities, issues certificates, and revokes compromised ones.

  Eg: DigiCert, Sectigo.

→ CAs are the backbone of PKI's, issuing and validating certificates.

Types →

① Self-signed → Created by the owner, not trusted by default.

② CA-signed → issued by a trusted CA, widely accepted.

Validation Methods :-

- CRL (Certificate Revocation List) :- A list of revoked certificates published by CA.

② OCSP : Real-time check of certificate status.

- Use in HTTPS/TLS :-

→ Browser-Server Trust Establishment :- Browser verifies CA-signed certificate to trust server.

→ Role in TLS Handshake : Certificate provides public key for key exchange and authentication.

- Common Attacks / Weakness :-

① Certificate Spoofing :- Fake certificates mimic legitimate ones to deceive users.

② CA Compromise :- Attack breaches a CA to issue fraudulent certificate.

③ Weak key attacks : Exploits poorly generated or short keys (Eg → < 1024-bit RSA)

④ Expired Certificate Misuse :

Attackers use outdated but valid-looking certificates.

- Real-life Example - HTTPS Padlock in Browsers
  - The padlock icon indicates a secure HTTPS connection, verified by a valid certificate.

eg: In 2017, the DigiNotar CA breach allowed fake google certificates, leading to wide spread MitM attacks until revoked.

▲ Common Defense Strategies :→

⊙ Use strong key length (eg→ 2048 bit or higher). and secure generation methods.

⊙ Regularly renew certificates and monitor expiration dates.

⊙ Implement CRLs or OCSP to revoke and check compromised certificates.

⊙ Enforce strict CA auditing and multi-factor authentication for CA access.

▲ Certificate Lifecycle :-

- Creation (Issuance) :-
  Process begins with a certificate request (eg→ CSR - Certificate Signing Request) from the entity (user or server).

- CA verifies the identity (eg; via documents or domain ownership) and generates the certificate, embedding the public key, subject details, and CA signature.

→ Initial validity period is set (for 1,2 yrs)

→ Usage→ • Active period where the certificate is used for authentication, encryption, or integrity (eg→ in TLS handshake)

→ Regular monitoring ensures it remains valid and uncompromised.

- Renewal :→
  Occurs before expiration to maintain trust; involves submitting a new CSR or reusing existing keys with CA revalidation.

  - Prevents service disruption

⊙ Revocation :-

⊙ Triggered if compromised (eg 'key theft') or no longer needed (eg→ Server decommission).

⊙ CA updates CRL (Certificate Revocation List), or OCSP, making the certificate as invalid.

⊙ Users/system check these lists to reject revoked certificates.

⊙ Expiration :→

⊙ Automatic end of validity period; certificate become invalid and must be renewed or replaced.

⊙ Failure to renew can lead to security warnings (eg→ browser alerts)

▲ Importance → Proper lifecycle management ensures continuous secure prevents misuse, and maintain trust in PKI system

HTTP v/s HTTPS : (comparison)

1) Definition :-

- HTTP : A protocol for transferring webpages over internet (port 80)
- HTTPS : Enhanced HTTP with SSL/TLS for secure data transmission (port → 443)

→ HTTP → No encryption
HTTPS = Encrypts using SSL/TLS to protect data.

→ HTTP : Vulnerable to eavesdropping and MiTM

→ HTTPS : Ensures CIA.

→ Certificate :- HTTPS use digital certificates issued by certificate authorities (CA) for trust.

→ Data Integrity → HTTPS : Verifies [illegible] data hasn't been tampered [illegible]

- Performance → HTTPS → Slightly slower due to encryption, but modern system (TLS 1.3) optimize speed.

SEO & Browser Behaviour :-

→ HTTPS : Preferred by Google for SEO ; browsers flag HTTP as 'Not Secure'.

→ Use Cases :-

① HTTP :- Suitable for non-sensitive content

② HTTPS :- Essential for login pages, payments, and personal data

Real-world Example :-

→ http://example.com ⇒ Unsecured, no pad
→ https://example.com = Secured, shows padlock, data pr

→ Comparision Table :→

Feature	HTTP	HTTPS
Port	80	443
Encryption	None	Yes
Security	Vulnerable	authentic
Certificates	Not used	Required
Data Integrity	Not ensured	Ensured
Authentication	none	server verified
Performance	faster	Slightly slower
SEO Impact	Penalized	Preferred
Use Case :-	Non-sensitive	sensitive data

•] Web Concepts.

① Client :
A client is a device or application (eg-) mobile app) that requests resource from server.

▲ Function :- Sends HTTP/HTTPS requests, receives response, and executes client side scripts

→ Role in security → Manages user interf (UI), input validation, and session managment; vulnerable to XSS if scripts are malicious.

Tip :- Ensure scripts are from trusted source to avoid injection attacks

② Server :-
A server (eg) Apache, NginX) is a s that hosts applications, serves cont and responds to client requests.

▲ Function :- Processes logic, executes d base (DB) queries, handles authentication, and manages data Storage.

• Security Focus : Misconfigurations (eg: open ports) or weak authentication can lead to breaches, use HTTPS to secure data.

• Tip → Regularly patch servers to prevent exploits.

3. HTTP/HTTPS Protocol :-

Model → Follows a request-response cycle where the client sends a request, and the server processes it to send a response.

• Request :- Contain method, URL, headers (eg, User-Agent), and optional body.

• Response :- Includes status code, headers (eg → Content-type), and body (eg HTML).

▸ Methods :-

⊙ Get, ⊙ POST, ⊙ PUT, ⊙ DELETE.

⊙ GET → Retrieves data (eg, webpage); Safe and idempotent.

⊙ POST → Sends data to the server (eg → form submission); not idempotent.

⊙ PUT : Updates existing resources.

⊙ DELETE :- Removes resources.

⊡ Security Note → Misuse (eg → Exposed GET data) can leak sensitive info; use POST for secure data.

▸ Status Codes :-

• 200 (OK) : Request successful.
• 301 (Moved Permanently) : Resource relocated
• 404 (Not Found) : Resource unavailable.
• 500 (Server Error) : Internal server issue

• Security Insight : Codes like 403 (Forbidd indicate access control issues.

▸ HTTPS Enhancements :-

• Uses TLS/SSL for encryption, ensuring

data confidentiality and integrity.

⊙ Includes a handshake (Client Hello, server certificate, key exchanges) to establish a secure connection

▸ Cybersecurity Benefits :- Protect against eavesdropping, MiTM attacks.

▸ Version Difference :

⊙ HTTP/1.1 = Supports persistent connections but can be slow with many requests

⊙ HTTP/2 :- Improves performance with multiplexing and header compression, requires HTTPS.

☐ Security Tip :- Always upgrade to HTTP/2 with TLS for better security and speed.

4.) URL Structure :-

⊙ Components : Protocol, domain, Path, query (https), (example.com), (/page), (?id=5).

Eg ⇒ https://example.com/page?id=5.

• Security Note :⇒ Query parameters can expose data; use POST for sensitive inputs.

Tip :- Validate all URL inputs to prevent injection attacks.

5.) DNS Resolution :-

⊙ Function : Converts domain names (eg → example.com) to IP addresses for connectivity.

⊙ Process : Root server → Top-level Domain (TLD) servers → Authoritative DNS servers.

⊙ Security Risk : DNS spoofing can redirect users to malicious sit

⊙ Tip : Use DNSSEC to ensure resolut integrity.

6. Cookies and Sessions :-

⊙ Function :- Cookies stores user data (eg. login status) ; session track uses ~~eta~~ state on the server.

⊙ Risk : Vulnerable to session hijacking (stealing session IDs) or cookie theft (eg via XSS).

⊙ Security Practice : Use secure, HttpOnly and Samesite cookies to mitigate risks.

Tip → Enforce encryption (HTTPS) to protect cookie data).

7.) Web Architecture :-

⊙ Frontend :→ Built with HTML, CSS, Java Script, for user interface ; target for client-side attacks

⊙ Backend :- Server-side code and database handle logic, prone to SQL injection if misconfigured.

⊙ API Integration : Connects frontend & backend ; secure with authentication (eg, OAuth).

⊙ Static vs Dynamic : Static (fixed content) v/s Dynamic (generated on request) ; dynamic sites need input validation

⊡ Tip : Segregate frontend and backend security control.

8.) Browser Security Features :-

⊙ HTTPS Padlock : Indicates a secure, trusted connection via valid certificate.

- Mixed Content Warnings : Alters when HTTPS pages load unsecured (HTTP) elements.

- Same-Origin Policy : Restricts scripts to the same domain, preventing unauthorized access.

- Tip :- Enable these features to enhance user trust & security.

9. CORS (Cross-Origin Resources Sharing).

=> - Function :- Controls cross-origin requests for APIs.

- Configuration :- Managed via response headers (eg -> Access-Control-Allow-Origin)

- Security Note -> Misconfigured CORS can allow unauthorized data access.

- Tip => Restrict origins to trusted domain only.

10. Content Security Policy :- (CSP)

-> Function -> HTTP headers that prevent XSS and code injection by restricting resource loading (eg -> scripts, style).

eg> Content-Security-Policy : default-src 'self'.

- Security Benefit : Mitigates risks from malicious scripts.

Tip :- Implement CSP to harden web application security.

(-> Intermediate is not done).
in depth is not done