Digital Signature Schemes & Authentication Protocols

Q. Explain the Needham-Schroeder Authentication Protocol.

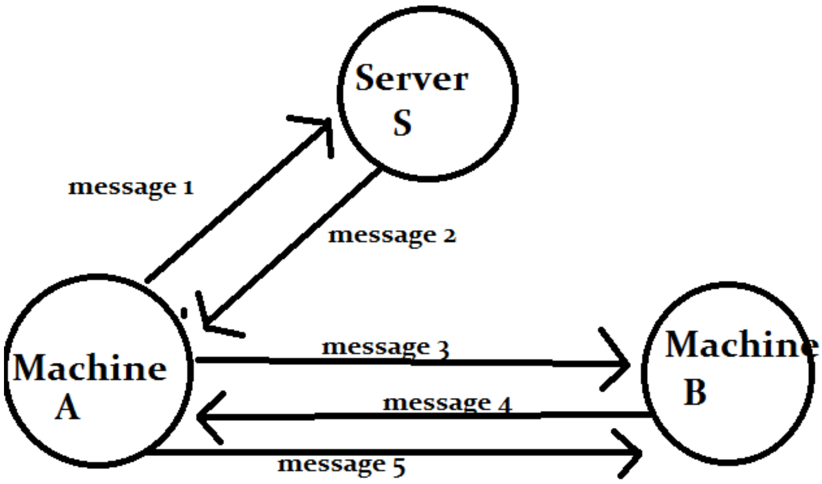
The **Needham–Schroeder Authentication Protocol** is a classic protocol designed to provide **mutual authentication** between two users (or systems) over an insecure network.

It uses a **trusted third party** (called the **Server**) to:

- Verify identities
- Distribute **session keys**
- Prevent **replay attacks** (in the updated version)

There are two main versions:

- **Symmetric-key version** (explained below)
- **Asymmetric-key version** (uses public/private keys)



- **Machine A** (initiator)
- **Machine B** (receiver)
- **Server S** (trusted key distribution center)

Entities and Notation:

- SK(AS): Symmetric key between A and S

- SK(BS): Symmetric key between B and S
- SK(S): New session key generated by S for A and B to use
- NON(A): Nonce generated by A (to ensure freshness)
- NON(B): Nonce generated by B (to authenticate A)

Protocol Steps:

1. A → S: Request to talk to B

msg1: A, B, NON(A)

A sends its ID, B's ID, and a nonce to the trusted server S.

2. S → A: Key + Ticket for B

msg2: {SK(S), B, NON(A), {SK(S), A}SK(BS)}SK(AS)

S sends:

- The new session key SK(S) for A to communicate with B
- The original nonce NON(A) to assure freshness
- A ticket for B {SK(S), A}SK(BS) encrypted with B's key

3. A → B: Forward ticket

msg3: {SK(S), A}SK(BS)

A sends the ticket to B so B can decrypt and obtain the session key.

4. B → A: Challenge

msg4: {NON(B)}SK(S)

B decrypts the ticket, retrieves SK(S), and sends a nonce encrypted with SK(S) to A.

5. A → B: Response

msg5: {f(NON(B))}SK(S)

A applies a function (e.g., f(NON(B)) = NON(B) - 1) and encrypts it with the session key to prove knowledge of SK(S).

Security Properties:

- **Authentication:** Both A and B confirm the identity of the other via nonce challenge-response.
- **Replay Protection:** Use of nonces ensures that even if an attacker replays old messages, the recipient can detect stale messages.

- **Confidentiality:** Session key SK(S) is protected through encryption with long-term keys.

Q. Explain the digital signatures?

A **digital signature** is a **cryptographic technique** that proves:

- The **authenticity** of the sender
- The **integrity** of the message
- **Non-repudiation** (the sender **cannot deny** sending it)

It is the **digital equivalent of a handwritten signature**, but much more secure because it is based on **mathematics and encryption**.

How Digital Signatures Work (Asymmetric Cryptography)

Digital signatures use **public key cryptography**, where:

- The **sender signs** the message using their **private key**
- The **receiver verifies** the signature using the sender's **public key**

Steps in Creating and Verifying a Digital Signature

A. Signing (Sender's Side)

1. **Hash the Message**:

   h=H(Message)

2. **Encrypt the Hash with Private Key**:

   Signature=EncryptPrivate Key(h)

3. **Send** the message along with the **digital signature**

B. Verifying (Receiver's Side)

1. **Hash the Received Message**:

   h1=H(Received Message)

2. **Decrypt the Signature using the Sender's Public Key**:

   h2=DecryptPublic Key(Signature)

3. **Compare Hashes**:
   - If $h_1 == h_2$ → The message is **authentic and unaltered**

    - If not → Message is **forged or modified**

Features of Digital Signatures

Feature	Description
Authentication	Confirms the sender’s identity
Integrity	Ensures the message was not tampered with
Non-repudiation	The sender cannot deny having sent the message
Legally Binding	Accepted in digital contracts and legal systems in many countries

---

Common Algorithms Used

Algorithm	Type
RSA	Uses private key to encrypt hash
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
Schnorr	Lightweight, efficient, used in Bitcoin

Real-World Applications

- Secure Email (PGP, S/MIME)
- Software Signing (Microsoft, Apple apps)
- E-Government & E-commerce
- Digital Contracts & E-signatures

Q. Differentiate between authentication and digital signatures.

Feature	Authentication	Digital Signature
Definition	Confirms the **identity** of the communicating party	Proves **who signed** a message and that it hasn't changed
Purpose	To verify **who** is communicating	To ensure **message authenticity**, **integrity**, and **non-repudiation**

Feature	Authentication	Digital Signature
Scope	Identifies a **user or device**	Verifies a **specific message or document**
Achieved By	Passwords, OTPs, biometrics, certificates, MAC, etc.	Asymmetric encryption + hash (e.g., RSA, DSA + SHA-256)
Non-repudiation	Not guaranteed	Yes – sender cannot deny having signed
Integrity Assurance	Not always (unless combined with MAC/HMAC)	Always – change in message invalidates signature
Verification Method	Usually server-side (login systems, tokens, etc.)	Anyone with public key can verify the signature
Example	Logging into Gmail using 2FA	Signing an email with PGP or digitally signing a contract

Q. What is the difference between digital certificate and digital signature?

Feature	Digital Certificate	Digital Signature
Definition	A file that **binds a public key to an identity**, issued by a CA	A **mathematical proof** that a message or document is authentic
Purpose	To **verify the identity** of the public key owner	To verify the **authenticity and integrity** of a message
Issued By	**Certificate Authority (CA)**	**Message sender** using their private key
Contains	Public key, identity info (name/domain), CA signature	Encrypted **hash** of the message (digest)
Proves	That the public key belongs to the claimed entity	That the sender signed the message and it was not modified
Key Used	Signed with CA's **private key**	Created using sender's **private key**, verified by **public key**
Verifies What?	The **trustworthiness** of the public key	The **integrity and origin** of a specific message

Feature	Digital Certificate	Digital Signature
Used In	HTTPS (SSL/TLS), VPNs, PKI systems	Email signing, software signing, legal documents

Q. What are the benefits of using digital signatures?

Benefit	Explanation
Authentication	Confirms the identity of the sender — only someone with the private key could have signed it
Integrity	Ensures the message/document has **not been altered** after signing — even a 1-bit change invalidates the signature
Non-repudiation	The sender **cannot deny** having signed the message, because the signature is mathematically tied to their private key
Speed & Automation	Digital signatures are **faster and more secure** than manual signing, and can be verified instantly by computers
Legal Validity	In many countries (e.g., under IT Act in India, eIDAS in EU), digital signatures are **legally recognized** as valid and binding
Remote Verification	Verifiers **don't need to meet the signer** — they just need the public key to verify authenticity
Security in Digital Communication	Used in **secure emails, software signing, e-contracts**, and **blockchain** to prevent tampering and forgery

Q. What is the need for both digital signatures and certificates?

Digital signatures and **digital certificates** are both essential parts of **public key cryptography**, but they serve **different roles**:

- A **digital signature** proves **who signed** a message and that the message has **not been altered**.
- A **digital certificate** proves **who owns a public key**, helping establish **trust** in the digital world.

Both are **used together** to ensure **secure, authenticated, and trusted communication** over the internet.

Need	How It's Solved
1. Authenticity of the sender	Digital **signature** ensures the sender actually signed the message
2. Verifying the sender's public key	Digital **certificate** tells you that the public key really belongs to the sender
3. Prevent forgery	Signature is tied to private key — only the real sender can generate it
4. Prevent impersonation	Certificate is issued and signed by a **trusted CA**, confirming the signer's identity
5. Build a chain of trust	Browser or system trusts the **CA** that issued the certificate
6. Ensure integrity	Any change in the message breaks the digital signature
7. Legal compliance	Digital certificates and signatures together provide **non-repudiation** and **legal trust**

Q. Explain the RSA digital signature scheme.

The **RSA digital signature scheme** is a cryptographic method used to ensure:

- **Authenticity** (message really comes from the sender)
- **Integrity** (message was not altered)
- **Non-repudiation** (sender cannot deny signing it)

It is based on the **RSA public-key algorithm**, using a **private key to sign** and a **public key to verify**.

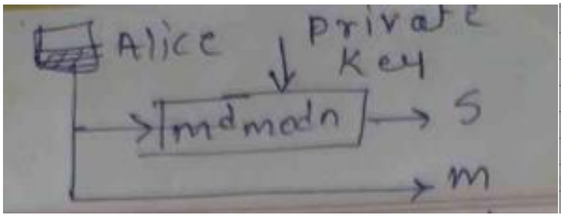
Key Idea

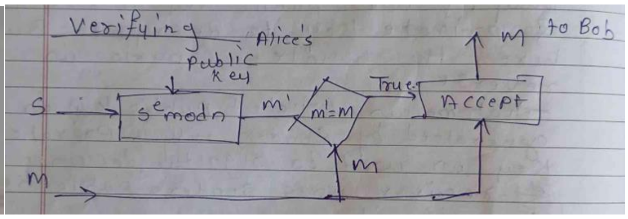
In RSA encryption:

- You **encrypt with the public key** and **decrypt with the private key**

In **digital signatures**, it's the reverse:

- You **sign with the private key**
- Anyone can **verify using the public key**





---

1. Key Generation (Same as RSA Encryption)

Let's say **Alice** wants to send a signed message to **Bob**:

1. Alice chooses two large prime numbers: p and q
2. Computes:
    - $n = p \times q$
    - $\varphi(n) = (p - 1) \times (q - 1)$
3. Chooses a public exponent e such that:
    - $1 < e < \varphi(n)$
    - $\gcd(e, \varphi(n)) = 1$
4. Computes private exponent d such that:
    - e^d ≡ 1 mod φ(n)

So the key pairs are:

- **Public Key**: (e, n) — shared with everyone
- **Private Key**: (d, n) — kept secret by Alice

---

2. Signing the Message

Let m be the original message (or its hashed version).

To **sign** the message:

s = m^d mod n

- s is the **digital signature**
- m is the message or a secure hash of the message (e.g., SHA-256)
- d is Alice's private key

Alice sends both m and s to Bob.

---

3. Verifying the Signature

Bob receives m and s.

To **verify**:

m' = s^e mod n

Then compare:

If m' == m → Signature is valid

Else → Signature is invalid

Bob uses Alice’s **public key** (e, n) for this step.

Q. Explain the ElGamal digital signature algorithm.

The **ElGamal Digital Signature Algorithm** is a public key signature scheme based on the **Discrete Logarithm Problem**, providing:

- **Authenticity** – Proves the message is from the claimed sender
- **Integrity** – Detects tampering
- **Non-repudiation** – Sender cannot deny the signature

It was proposed by **Taher ElGamal** in 1985 and is the basis for other schemes like **DSA (Digital Signature Algorithm)**.

System Parameters (Publicly Known)

- A large **prime number** p
- A **primitive root** g of p
- A **hash function** H(M)H(M), such as SHA-256

---

1. Key Generation

a) System Parameter Generation (public constants):

- Choose a large **prime** number p
- Choose a **generator** g such that 1 < g < p
- Choose a **cryptographic hash function** H (e.g., SHA-256)

b) User-specific Key Pair:

- Choose a **private key** x, randomly from {1, ..., p−2}
- Compute **public key** y = g^x mod p

Key Pairs:

- **Private key**: x (kept secret)
- **Public key**: (p, g, y) (shared with others)

---

2. Key Distribution

- The **signer** sends the **public key (p, g, y)** to anyone who needs to verify their signature.
- The **private key x** is kept secret by the signer.

---

3. Message Signing

To **sign a message M**, the signer does:

1. Choose a random k such that:
    - $1 < k < p-1$
    - $\gcd(k, p-1) = 1$ (i.e., k is co-prime with p−1)
2. Compute:

    $r = g^k \bmod p$
3. Compute:

    $s = (H(M) - x*r) * k^{-1} \bmod (p-1)$
    - H(M) is the hash of the message
    - k^{-1} is the modular inverse of k modulo (p−1)

Signature = (r, s)

---

4. Signature Verification

The **receiver** verifies the signature (r, s) on message M as follows:

1. Check that $0 < r < p$ and $0 < s < p-1$
2. Compute:
3. $v1 = y^r * r^s \bmod p$
4. $v2 = g^{H(M)} \bmod p$

If:

$v1 \equiv v2 \bmod p$

Then the **signature is valid**

Q. Explain the Schnorr digital signature scheme.

The **Schnorr digital signature scheme** is a secure and efficient signature algorithm based on the **Discrete Logarithm Problem**.

- Proposed by **Claus Schnorr** in the 1990s
- Known for being **lightweight, fast, and compact**
- Used in modern cryptographic systems like **Bitcoin Taproot**

---

1. Key Generation

1. Choose a **large prime p** and a **prime q** such that:

   q divides (p - 1)

2. Choose a generator a of order q mod p, i.e.:

   $a^q \equiv 1 \mod p$

3. Choose a random private key:

   $s \in \{1, 2, ..., q-1\}$

4. Compute the **public key**:

   $v = a^{(-s)} \mod p$

---

2. Signature Generation

To sign a message m:

1. Choose a random number $r \in \{1, 2, ..., q-1\}$
2. Compute:

   $x = a^r \mod p$

3. Compute the hash:

   $e = H(m \,||\, x)$ // Hash of the message concatenated with x

4. Compute:

   $y = (r + s \cdot e) \mod q$

5. The signature is:

(e, y)

---

3. Signature Verification

To verify signature (e, y) on message m:

1. Compute:

   $x' = a^y \cdot v^e \bmod p$

2. Compute:

   $e' = H(m \| x')$

3. Accept the signature if:

   $e' == e$

Features of Schnorr Signature

Feature	Description
Security	Based on **Discrete Logarithm Problem**
Compact	Smaller signature size than RSA or DSA
Fast	Efficient in both signing and verifying
Unforgeable	Cannot forge signature without knowing the private key
Modern Use	Used in **Bitcoin Taproot**, privacy-enhancing systems, and secure messaging protocols

---

Advantages of Schnorr:

- Shorter signatures
- Computationally efficient
- Strong provable security under the discrete log assumption

Q. Compare Schnorr, ElGamal, and DSS signature schemes.

Feature	RSA	ElGamal	Schnorr
Underlying Problem	Integer factorization	Discrete logarithm	Discrete logarithm

Feature	RSA	ElGamal	Schnorr
Key Generation	Receiver generates (e, d, n)	Sender generates (p, g, x, y)	Sender generates (p, q, a, s, v)
Key Size	Large (e.g., 2048-bit)	Large (p ~ 2048 bits, q ~ 256 bits)	Smaller (q ~ 256 bits, p ~ 2048 bits)
Signature Size	1 value (same size as n)	2 values (r, s)	2 values (e, y)
Hash Function	Optional	Required	Required
Randomness Needed	No	Yes (random k per signature)	Yes (random r per signature)
Security Assumption	Factoring is hard	Discrete log is hard	Discrete log is hard
Efficiency (Signing)	Moderate	Slower (needs modular inverse)	Fast (simpler math)
Efficiency (Verification)	Moderate	Slower	Very Fast (minimal operations)
Signature Size (bits)	~2048	~512–1024 (depends on q)	~256–512 (very compact)
Replay/Deterministic Safe?	Yes if message hashed	No (must use fresh k)	No (must use fresh r)
Provable Security	Under specific assumptions	Partial	Strong under standard assumptions
Standard Use	Legacy systems (e.g., PGP, SSL)	Used in DSA	Used in modern protocols (e.g., EdDSA)

Q. Describe a man-in-the-middle attack and how to prevent it in signature schemes.

A **Man-in-the-Middle (MITM)** attack occurs when a malicious actor secretly **intercepts, alters, or relays communication** between two parties — making both believe they are communicating directly with each other.

The attacker can read, modify, or even forge messages without detection if proper security is not in place.

MITM Attack in the Context of Digital Signatures

Even if a message is signed, a **MITM attacker** can:

- **Intercept** a signed message
- Replace the **message and signature** with their own
- Present their **own public key** pretending to be the original sender

This is possible **only if the receiver cannot verify that the public key really belongs to the sender**.

MITM Attack Example in a Signature Scheme:

1. Alice signs a message with her private key and sends it to Bob.
2. Mallory intercepts the message.
3. Mallory replaces Alice's signature and message with her own.
4. Mallory sends it to Bob, pretending to be Alice, along with her **own public key**.
5. Bob verifies it using **Mallory's public key**, thinking it's Alice's — and the attack succeeds.

How to Prevent MITM in Digital Signature Schemes

Prevention Method	Explanation
Use of Digital Certificates (X.509)	Certificates issued by a **trusted CA** bind a public key to a verified identity, so Bob can be sure the public key is really Alice's
Public Key Infrastructure (PKI)	Uses **CAs and trust chains** to authenticate public keys and prevent impersonation
Key Fingerprint Verification	Users can verify the **hash of public keys** (used in PGP and SSH) to ensure identity
TLS/SSL in Transit	Secures communication channels to prevent interception of public keys or signed data
Use Nonces/Timestamps	Prevents replay attacks in combination with signatures
Signature Verification Logic	Always verify the signature **using the sender's known and trusted public key**, not one received in the same message

Q. Compare authentication protocols (e.g., with/without replay attack protection).

Authentication protocols verify the identity of users or devices.
Some protocols are **vulnerable to replay attacks** if they don't include **freshness verification**, like **nonces** or **timestamps**.

Feature	Without Replay Protection	With Replay Protection
Example Protocols	Basic password exchange, early versions of Needham-Schroeder	Kerberos, Challenge–Response, Modern Needham-Schroeder (with timestamp)
Vulnerable to Replay Attack?	Yes — attacker can capture and resend authentication messages	No — messages are valid only once or expire quickly
Freshness Mechanism	None	Uses **nonces**, **timestamps**, or **session tokens**
Typical Attack Scenario	Attacker replays a valid login request to impersonate user	Attacker's replay fails because message is recognized as duplicate
Efficiency	Fast, but insecure	Slightly slower (requires tracking nonces or clocks)
Security Level	Weak — cannot ensure message is recent or unique	Strong — ensures message **origin and freshness**
Used In	Simple IoT systems, outdated protocols	TLS/SSL, Kerberos, OAuth, SSH, modern VPNs

Q.

Q.