Lecture notes - Chapter 2: Cryptography

Topics: 3 hours

o Basics of Cryptography: Symmetric and Asymmetric Encryption
o Cryptographic Algorithms: DES, AES, RSA, ECC
o Hash Functions and Digital Signatures
o Key Management and Distribution
o Digital Certificates
o Cryptographic Protocols and Applications

Basics of Cryptography: Symmetric and Asymmetric Encryption

Definition: Cryptography is the practice of securing information by transforming it into an unreadable format that can only be decoded by someone with the proper key.

Purpose: Ensures data confidentiality, integrity, and authentication.

Encryption is a specific subset of cryptography. It refers to the process of converting plaintext (readable data) into ciphertext (an unreadable format) using an algorithm and a key.

Symmetric Encryption

- **Definition**: A type of encryption where the same key is used for both encryption and decryption.
- **Key Characteristics**:
  - **Speed**: Faster and more efficient, making it suitable for encrypting large amounts of data.
  - **Key Management**: Requires secure sharing of the secret key between parties.
- **Common Algorithms**:
  - **AES (Advanced Encryption Standard)**: Widely used for securing data.
  - **DES (Data Encryption Standard)**: An older, less secure algorithm.
- **Example Use Cases**:
  - **File encryption**: Encrypting files on a disk to prevent unauthorized access.
  - **Secure communications**: Encrypted messaging within a private network.

Asymmetric Encryption

- **Definition**: A type of encryption that uses a pair of keys – a public key for encryption and a private key for decryption.
- **Key Characteristics**:
  - **Key Pair**: Public key is shared openly, while the private key is kept secret.

    - **Security**: Provides better security for key exchange, but slower than symmetric encryption.
- **Common Algorithms**:
    - **RSA (Rivest-Shamir-Adleman)**: Commonly used for secure data transmission.
    - **ECC (Elliptic Curve Cryptography)**: More efficient, offering the same security with smaller key sizes.
- **Example Use Cases**:
    - **Digital Signatures**: Verifying the authenticity and integrity of a message.
    - **SSL/TLS**: Securing online communications, such as in HTTPS.

Hash Functions and Digital Signatures

What is a Hash Function?

- **Definition**: A hash function takes an input (or "message") and produces a fixed-size string of bytes, typically a hash value or "digest."
- **Purpose**: Used for data integrity, password storage, and digital signatures.

Key Properties of Hash Functions:

1. **Deterministic**: The same input always produces the same hash value.
2. **Fixed Output Length**: Regardless of the input size, the output (hash) has a fixed length (e.g., 256 bits for SHA-256).
3. **Fast Computation**: Hash functions are designed to compute the hash value quickly.
4. **Preimage Resistance**: It's hard to reverse the process—finding the original input from the hash value should be infeasible.
5. **Collision Resistance**: It's difficult to find two different inputs that produce the same hash value.
6. **Avalanche Effect**: A small change in the input (even a single bit) drastically changes the hash value.

How Hash Functions Are Used:

1. **Data Integrity**:
    - **Example**: Downloading a file from the internet. A hash value is often provided with the file. You can compute the hash of the downloaded file and compare it to the provided hash to check if the file was altered or corrupted.
2. **Password Storage**:
    - **Example**: Instead of storing passwords directly, systems store hashes of passwords. When a user logs in, the system hashes the entered password and compares it to the stored hash.

3. **Digital Signatures**:
    - **Example**: Hash functions are used to create a digest of a message. This digest is then signed to verify the message’s authenticity and integrity.
4. **Data Structures**:
    - **Example**: Hash functions are used in hash tables to efficiently locate data.

Common Hash Functions:

- **MD5 (Message Digest Algorithm 5)**:
    - **Output**: 128-bit hash.
    - **Usage**: Previously used for checksums and data integrity, but now considered weak due to vulnerability to collisions.
- **SHA-1 (Secure Hash Algorithm 1)**:
    - **Output**: 160-bit hash.
    - **Usage**: Used in various security applications, but also considered weak due to collision vulnerabilities.
- **SHA-256 (Secure Hash Algorithm 256-bit)**:
    - **Output**: 256-bit hash.
    - **Usage**: Commonly used in modern applications for secure hashing.

Introduction to Password Hashing

- **Purpose**: Password hashing is used to securely store passwords by converting them into a fixed-size, irreversible string of characters (hash).
- **Challenge**: Hash functions alone can be vulnerable to attacks like rainbow tables, where precomputed hash values are used to crack passwords.

What is a Salt?

- **Definition**: A salt is a random value added to a password before hashing.
- **Purpose**: The salt ensures that even if two users have the same password, their hash values will be different.

How Salt Works

- **Salting Process**:
    1. A unique salt is generated for each password.
    2. The salt is concatenated with the user's password.
    3. The combined string (salt + password) is hashed using a hash function.
- **Storage**: Both the salt and the hash value are stored in the database.

Benefits of Using Salt

- **Prevents Rainbow Table Attacks**: Since each password has a unique salt, attackers cannot use precomputed hash tables to crack passwords.
- **Unique Hashes for Identical Passwords**: Even if two users have the same password, the addition of different salts will produce different hash values.
- **Increased Security**: Salting increases the complexity and security of stored passwords, making it harder for attackers to guess or crack them.

What is a Digital Signature?

- **Definition**: A digital signature is a cryptographic technique used to verify the authenticity, integrity, and non-repudiation of a digital message or document. It acts like a virtual fingerprint, ensuring that the document comes from a verified source and hasn't been altered in transit.

How Does a Digital Signature Work?

- Digital signatures are created and verified using asymmetric encryption, which involves a pair of keys: a private key (known only to the signer) and a public key (available to anyone).

Steps to Create a Digital Signature

1. **Hashing the Message:**
    - The message (or document) that needs to be signed is first processed through a hash function.
    - A hash function converts the message into a fixed-size string of bytes, typically represented as a hash value (or message digest).
    - This hash value is unique to the original message; even a small change in the message will result in a different hash value.
2. **Encrypting the Hash:**
    - The hash value is then encrypted using the sender's private key. This encrypted hash is the digital signature.
    - Since the private key is known only to the signer, this ensures that the signature could have only been created by the signer.
3. **Appending the Signature:**
    - The digital signature is attached to the original message, and both are sent to the recipient.

Steps to Verify a Digital Signature

1. **Receiving the Message:**
    - The recipient receives the original message along with the digital signature.
2. **Hashing the Received Message:**
    - The recipient applies the same hash function to the received message to generate a new hash value.
3. **Decrypting the Signature:**
    - The recipient then decrypts the digital signature using the sender’s public key (which is publicly available).
    - This decryption reveals the original hash value (the one that was encrypted with the sender's private key).
4. **Comparing Hashes:**
    - The recipient compares the decrypted hash value (from the signature) with the newly generated hash value (from the received message).
    - If the two hash values match, it confirms that:
        - The message was not altered in transit (integrity).
        - The message was indeed sent by the sender (authenticity).
    - If the hash values do not match, it indicates that the message has been tampered with or that the signature is not authentic.

Key Characteristics of Digital Signatures

- **Authentication:** Confirms the identity of the sender.
- **Integrity:** Ensures the message has not been altered during transmission.
- **Non-Repudiation:** The sender cannot deny having sent the message because only they have access to the private key used to create the signature.

Use Cases of Digital Signatures

- **Email Security:** Ensuring the authenticity and integrity of emails.
- **Document Signing:** Legally binding digital documents, such as contracts.
- **Software Distribution:** Verifying that software has not been tampered with.
- **Secure Transactions:** Ensuring the integrity and authenticity of online transactions.

Digital Signature Standards

- **RSA:** One of the first and most widely used algorithms for creating digital signatures.
- **DSA (Digital Signature Algorithm):** A federal standard for digital signatures in the U.S.
- **ECDSA (Elliptic Curve Digital Signature Algorithm):** A variant of DSA that uses elliptic curve cryptography for smaller, faster signatures.

HMAC (Hash-Based Message Authentication Code):

- HMAC (Hash-Based Message Authentication Code) is a security mechanism that combines a secret key with a hash function to verify the integrity and authenticity of a message. By using a shared secret key, HMAC generates a unique hash value that can be appended to the message. When the message is received, the same key and hash function are used to generate a new HMAC, which is compared to the original. If they match, it confirms that the message has not been altered and is from a trusted source. HMAC is widely used in securing data transmissions, such as in APIs and HTTPS.

Key Difference: Digital signatures provide non-repudiation and use asymmetric keys, while HMACs are faster and use symmetric keys but do not offer non-repudiation.

Key Management and Distribution

Introduction to Cryptographic Key Management

Cryptographic keys are the foundation of securing data in various encryption and security protocols. Improper generation or storage of keys can lead to unauthorized access, data breaches, and loss of confidentiality. Therefore, securing the storage of cryptographic keys is critical in protecting sensitive information.

Key Principles for Safely Creating and Storing Cryptographic Keys

Confidentiality: The key should be stored securely (e.g., in hardware security modules (HSMs), or using encryption at rest) to prevent unauthorized access or leaks. The key must remain confidential and be protected against unauthorized access. Compromise of the key would defeat the security of the encryption system.

Integrity: Ensure that the keys cannot be tampered with or altered.

Availability: The keys must be accessible to authorized users and processes when needed, but not to attackers.

Separation of Duties: Limit key access by ensuring that no single person or system has complete control over the keys.

Sufficient Length: Key size should be large enough to resist brute-force attacks. For example:

  - **Symmetric keys:** At least 128 bits (e.g., AES-128).
  - **Asymmetric keys:** RSA keys of at least 2048 bits or ECC keys of at least 256 bits.

Randomness: The key should be generated using a secure, unpredictable **random number generator (RNG)** to avoid patterns that could be exploited by attackers.

Uniqueness: Each key should be unique to prevent reuse across different sessions or systems, reducing the risk of exposure.

Non-guessability: Keys should be generated in a way that they are **unpredictable and resistant to guessing** attacks, such as dictionary or birthday attacks.

Expiration and Rotation: Cryptographic keys should have a defined **lifetime** and be periodically rotated to limit the damage if a key is compromised.

Best Practices for Cryptographic Key Storage

1. **Limit Key Access:** Use access controls to ensure that only authorized individuals and processes can access or use the cryptographic keys.
2. **Key Rotation:** Regularly rotate keys to minimize the impact of a compromised key. Automate key rotation when possible.
3. **Key Backup and Recovery:** Securely back up keys and ensure there is a secure and tested recovery process in place for critical keys.
4. **Monitor and Audit Key Usage:** Implement logging and monitoring to track key usage and detect any unauthorized access or suspicious activities.
5. **Physical Security:** Ensure that systems storing cryptographic keys (e.g., HSMs or TPMs) are physically secure to prevent tampering or theft.
6. **Multi-Factor Authentication (MFA):** Implement MFA to ensure that accessing cryptographic keys requires multiple forms of verification.

Common Mistakes in Cryptographic Key Management

1. **Storing Keys in Code:** Hardcoding keys in source code is a significant risk, as the code could be exposed in version control systems or during application runtime.
2. **Storing Keys in Plain Text:** Storing cryptographic keys in plain text files on disk is insecure and makes them easily accessible to attackers.
3. **Neglecting Key Rotation:** Failing to rotate keys regularly increases the risk of long-term exposure in case of compromise.
4. **Weak Access Controls:** Inadequate control over who can access the keys can lead to unauthorized use and potential data breaches.
5. **Weak Key Generation:** Using predictable or non-random sources to generate cryptographic keys (e.g., using weak random number generators or hard-coded keys).
6. **Key Reuse**: Reusing the same cryptographic key across multiple systems, sessions, or data types, increasing the risk of compromise.
7. **Using Short Keys:** Choosing keys that are too short for modern cryptographic standards,

making them vulnerable to brute-force attacks (e.g., using 56-bit DES keys).

8. **Insecure Key Transmission:** Transmitting keys over insecure channels (e.g., sending them via email or over unencrypted connections), allowing attackers to intercept them.

Digital Certificates

Definition

Digital Certificates are electronic documents used to verify the ownership of a public key and authenticate the identity of the certificate holder, typically in secure communications (e.g., SSL/TLS).

A **Certificate Authority (CA)** is a trusted entity that issues and manages digital certificates. CAs verify the identity of entities (e.g., websites, organizations, individuals) and bind their public keys to these identities by issuing digitally signed certificates.

Key Components

1. **Subject**: The entity that owns the certificate (e.g., a website, organization).
2. **Public Key**: The public key of the subject, which will be used for encryption or signature verification.
3. **Issuer**: The Certificate Authority (CA) that issued the certificate.
4. **Validity Period**: The time span during which the certificate is valid (start and expiration dates).
5. **Digital Signature**: A signature from the CA that verifies the authenticity and integrity of the certificate.
6. **Serial Number**: A unique identifier assigned to the certificate by the CA.

How It Works

Public Key Binding: The certificate binds a public key to the entity's identity.

Verification: When a client connects to a server, the server presents its digital certificate. The client checks the certificate against a trusted CA's signature to verify its legitimacy.

Common Uses

- **SSL/TLS:** Secures web traffic (e.g., HTTPS).
- **Code Signing:** Verifies the authenticity and integrity of software.
- **Email Encryption:** Provides secure email communication via protocols like S/MIME.

Cryptographic Protocols and Applications

Key Cryptographic Protocols

1. **TLS (Transport Layer Security):**
   - **Purpose:** Secures communication over the internet, such as web browsing (HTTPS), email, and VoIP.
   - **How It Works:** Uses a combination of symmetric encryption, asymmetric encryption, and digital certificates to establish a secure connection between a client and server. The handshake process authenticates the server (and optionally the client) and securely exchanges encryption keys.
   - **Applications:** HTTPS, secure email (SMTP/IMAP/POP), VPNs, and more.
2. **IPsec (Internet Protocol Security):**
   - **Purpose:** Secures Internet Protocol (IP) communications by authenticating and encrypting each IP packet in a data stream.
   - **How It Works:** Operates in two modes: Transport mode (encrypts only the payload) and Tunnel mode (encrypts the entire packet). Uses encryption algorithms (e.g., AES) and key exchange protocols like IKE (Internet Key Exchange).
   - **Applications:** VPNs, secure network traffic, and secure remote access.
3. **SSH (Secure Shell):**
   - **Purpose:** Provides secure remote login and file transfer over an unsecured network.
   - **How It Works:** Uses asymmetric cryptography for session initiation and symmetric encryption for the data exchange once the session is established.
   - **Applications:** Secure remote administration of servers, secure file transfer (SFTP), and tunneling.

Lab: 1 hour

o Homework - Digital Signatures practical applications
o Homework - Case Study on Security Breaches due to weak cryptography
o Symmetric and Asymmetric Encryption efficiency and applications - SSL/TLS handshake - https://forms.gle/GfxMtWm4KzL7cByw7