Ingeniería En Desarrollo Y Gestión De Software

Nombre:

García Arreola Howard Isai

Subject:

Desarrollo movil Integral

Actividad:

Data Encryption Mechanisms in Mobile Applications

Grupo: 10-B

Profesor:

Ray Brunett Parra

Galaviz

Fecha de Realización:

January 24th, 2025

Symmetric Encryption

Symmetric encryption uses the same key for both encryption and decryption. It's fast and efficient, making it suitable for encrypting large amounts of data.

- Advantages: High speed and efficiency.
- Disadvantages: Key distribution can be challenging.
- Common Algorithms: AES (Advanced Encryption Standard), DES (Data Encryption Standard).

Asymmetric Encryption

Asymmetric encryption, also known as public-key cryptography, uses a pair of keys - a public key for encryption and a private key for decryption. It's often used for secure key exchanges and digital signatures.

- Advantages: Enhanced security through public and private key pairs.
- Disadvantages: Slower than symmetric encryption.
- Common Algorithms: RSA (Rivest-Shamir-Adleman), ECC (Elliptic Curve Cryptography).
-

Encryption Standards

- **AES (Advanced Encryption Standard):**
  - Widely used symmetric encryption algorithm.
  - AES-256 is the most secure and recommended variant.
  - Ideal for encrypting sensitive data like files and database entries.
- **RSA:**
  - Asymmetric encryption algorithm, typically used for encrypting small amounts of data or for key exchange.
  - Key sizes of 2048 bits or more are recommended.

- **ECC (Elliptic Curve Cryptography):**
  - A lightweight asymmetric encryption alternative to RSA.
  - Suitable for resource-constrained devices.

Secure Data Storage

- **Keychain (iOS) and Keystore (Android):**
  - Platform-specific secure storage mechanisms for sensitive data like encryption keys, tokens, or passwords.
  - Data stored here is protected using hardware-backed security.
- **Encrypted Shared Preferences (Android):**
  - Use EncryptedSharedPreferences for securely storing small data sets.

Encryption for Data in Transit

- **TLS (Transport Layer Security):**
  - Ensures data sent over the network is encrypted.
  - Always enforce HTTPS for all API communications using TLS 1.2 or TLS 1.3.
- **End-to-End Encryption (E2EE):**
  - Encrypts data on the sender's device and decrypts it only on the receiver's device.
  - Useful for messaging apps or secure communications.

Data Encryption in Mobile Applications: Hybrid and Native Apps

Data encryption in mobile applications, whether hybrid or native, is essential for securing sensitive user and application data. Encryption works by converting plaintext data into an unreadable format, called ciphertext, using cryptographic algorithms like AES (Advanced Encryption Standard) or RSA (Rivest-Shamir-Adleman). In hybrid apps, encryption is typically implemented at the API level using libraries compatible with the app's runtime, such as CryptoJS for JavaScript frameworks like Ionic or React Native. For native apps, platform-specific libraries like Android's EncryptedSharedPreferences or iOS's CryptoKit

are used, often integrating hardware-backed security like Secure Enclave or Trusted Execution Environment (TEE).

Encryption can occur in three primary scenarios: data at rest, data in transit, and during key management. For data at rest, encryption secures local storage such as files, databases, or shared preferences. This is critical when storing sensitive information like authentication tokens, personal details, or financial data. In transit, encryption ensures data integrity and confidentiality as it moves between the app and backend servers, using HTTPS with TLS (Transport Layer Security) or end-to-end encryption protocols. Additionally, key management is crucial, as encryption keys must be securely generated, stored, and accessed ideally using secure storage solutions like Android Keystore or iOS Keychain.

The timing of encryption depends on the data's lifecycle and sensitivity. For data at rest, encryption should be applied immediately when data is written to local storage and decrypted only when accessed. For data in transit, encryption starts before transmitting data over the network and ends when it is securely received and decrypted by the server or recipient. Scenarios where encryption is critical include financial transactions, health records, confidential messaging, and any application handling personally identifiable information. Failure to implement robust encryption mechanisms can lead to breaches, data theft, and non-compliance with regulations like GDPR or HIPAA.

Hybrid apps often use abstraction layers provided by frameworks, making it necessary to ensure the cryptographic implementations are secure and efficient. Native apps, on the other hand, have direct access to platform-specific features and hardware, allowing for finer-grained control over encryption. Regardless of the approach, adhering to best practices such as avoiding hardcoded keys, regular cryptographic library updates, and compliance with industry standards is critical for protecting user data.

References

Zimperium. (2024, May 3). *Hybrid App Security: Enterprise Mobile app best practices*. https://www.zimperium.com/glossary/hybrid-app/?

Team, M. (2023, July 11). *A Look At Security: Hybrid Apps vs. Native Apps*. MindSea. https://mindsea.com/app-security-hybrid-apps-vs-native-apps/

Native App vs Hybrid App Security: Comparison, Advantages & Measures. (2024, February 27). Dddd Blog. https://blog.dddd.ae/en/application-design/native-app-vs-hybrid-app-security-comparison%2C-advantages-measures?