Encryption and Security Documentation

Overview

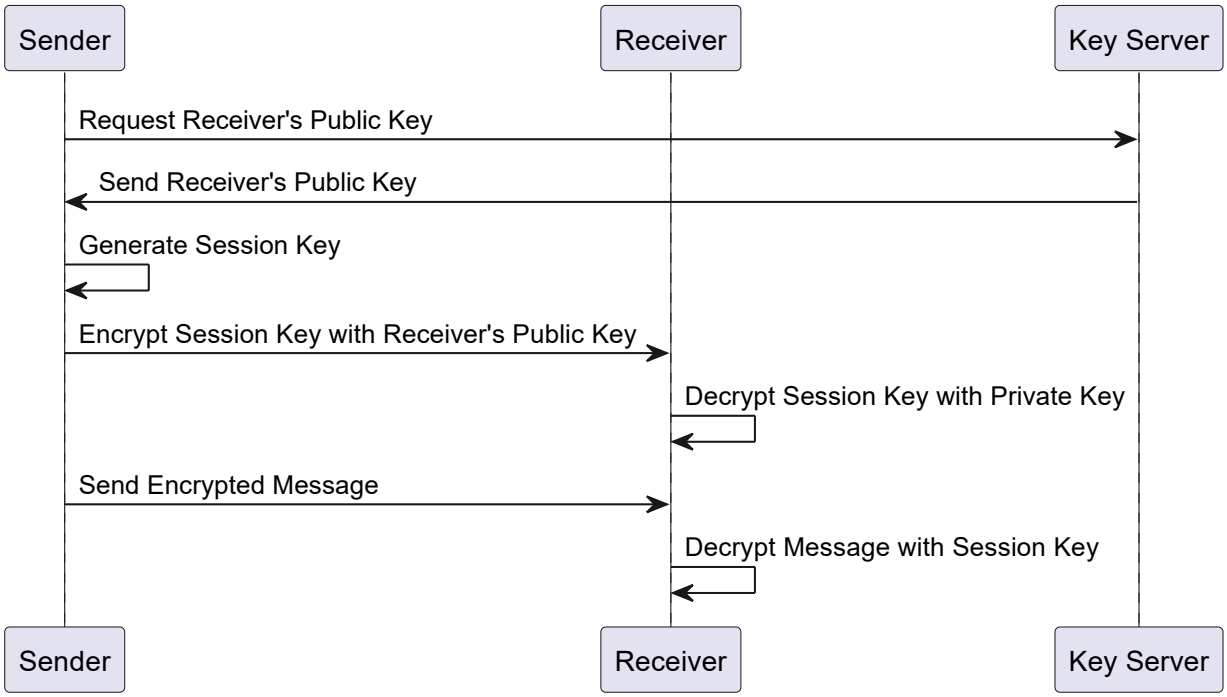
This document outlines the encryption and security measures employed in the messaging application to ensure data privacy, integrity, and protection against unauthorized access.

Encryption Techniques

1. End-to-End Encryption (E2EE)

- Messages exchanged between users are encrypted on the sender's device and decrypted only on the recipient's device.
- **Algorithm**: AES-256 for message encryption and RSA-2048 for key exchange.
- **Key Generation**:
  - Each user generates a public-private key pair using RSA.
  - Public keys are exchanged to securely share session keys for AES encryption.

End-to-End Encryption Workflow



---

2. Data at Rest Encryption

- All data stored in the database (PostgreSQL) and Firebase Firestore is encrypted using AES-256.
- Media files stored in cloud storage are encrypted with a unique key per file.

---

3. Data in Transit Encryption

- All communication between clients and servers is secured using **TLS 1.3** to prevent eavesdropping and tampering.

Security Measures

1. Authentication Security

- **OAuth 2.0** is used for user authentication.
- Passwords are hashed using **bcrypt** before storage.
- Multi-factor authentication (MFA) is enabled for additional account security.

---

2. Message Integrity

- Messages include a **hash-based message authentication code (HMAC)** to ensure integrity.
- HMAC is generated using SHA-256 to prevent message tampering during transit.

---

3. Secure Key Management

- Key server stores public keys only. Private keys are never transmitted or stored on the server.
- Session keys are regenerated periodically to enhance security.

Threat Mitigation

1. Replay Attacks

- Use nonces and timestamps in requests to prevent replay attacks.
- Nonces are unique to each session and verified on the server.

---

2. Man-in-the-Middle (MITM) Attacks

- Enforce **certificate pinning** on the client side to prevent MITM attacks.
- Validate SSL/TLS certificates on all requests.

---

3. SQL Injection and XSS

- Sanitize all user inputs to prevent SQL injection and cross-site scripting (XSS) attacks.
- Use ORM (e.g., TypeORM or Prisma) to manage database interactions safely.

---

4. Rate Limiting and DDoS Protection

- Employ rate-limiting on APIs to prevent brute force attacks.
- Use a web application firewall (WAF) to filter malicious traffic.

Session Security

Session Token Management

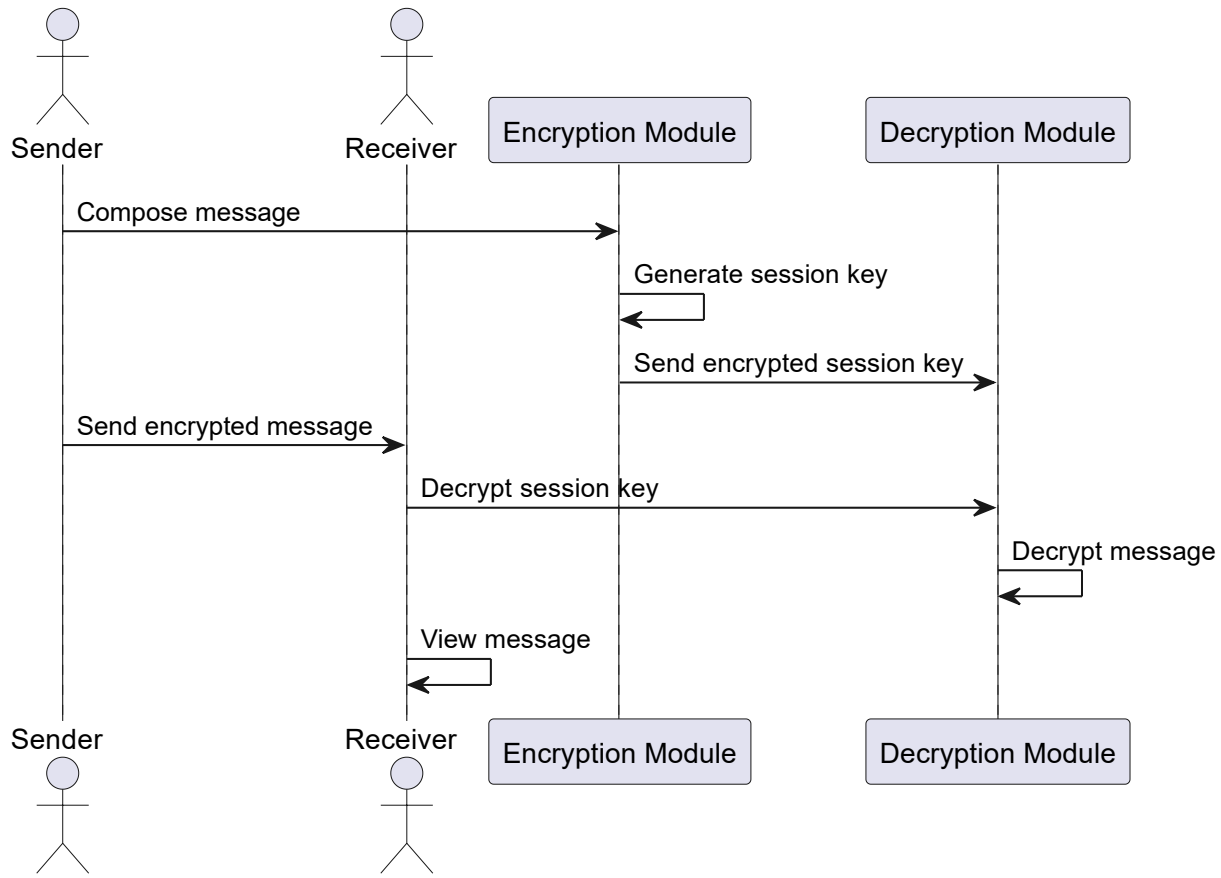
- Tokens are signed using **HMAC SHA-256**.
- Expired tokens are invalidated, and users are logged out automatically.
- Refresh tokens are securely stored and rotated periodically.

Audit Logging

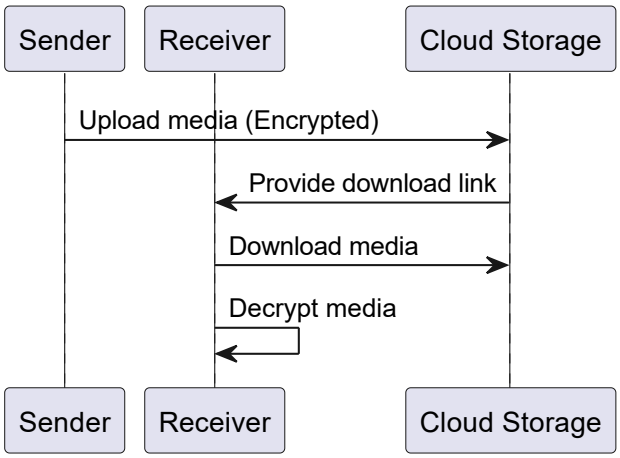
- Maintain audit logs for:
  - User logins and logouts.
  - Failed login attempts.
  - Sensitive actions like account deletion or password changes.

Encryption Workflow

Secure Message Exchange



Secure Media Transfer



Future Enhancements

- Implement **post-quantum cryptography** to safeguard against potential quantum computing threats.
- Introduce **hardware security modules (HSMs)** for key storage and management.
