Eduardo Guerrero A. **00326712**
Sebastián Navarro **00321588**
Mateo Pozo **00320780**

Part 1: Research on Cryptographic Hash Functions

Cryptographic hash functions are mathematical algorithms designed to convert data sets of any size into a fixed-length string of characters, known as a hash (Black Duck, 2015). This means that regardless of the length of the input data, the output will always have a specific length. For example, the complete works of Shakespeare can be summarized into a 40-character string using the SHA-1 function. Although collisions are possible (i.e., two distinct inputs generate the same hash), secure hash functions minimize this probability to the point where it becomes virtually impossible. For example, the hash of the word “Apple” is `476432a3e85a0aa21c23f5abd2975a89b6820d63`, while the hash of “apple,” which only differs by one lowercase letter, is `d0be2dc421be4fcd0172e5afceea3970e2f3d940`, demonstrating how minimal changes in the input result in significantly different hashes.

Among the most popular hash functions are MD5, which produces a 128-bit hash but is now considered insecure due to its collision vulnerabilities; SHA-1, which generates a 160-bit hash but is also deemed vulnerable; and SHA-2, a family that includes variants such as SHA-224, SHA-256, SHA-384, and SHA-512. SHA-256, in particular, is the most widely used variant in modern security systems like Bitcoin and the TLS security protocol because it generates a 256-bit hash. SHA-3, the latest variant in this family, offers additional improvements in resistance to certain types of advanced attacks (freeCodeCamp, 2020).

Cryptographic hashes have crucial applications in digital security, including data integrity verification, secure password storage, digital signatures, blockchain, and cryptocurrencies. These applications depend on certain fundamental properties of cryptographic hash functions: determinism, fixed length, computational efficiency, and resistance to collisions, preimages, and second preimages.

First, determinism ensures that the same input message will always produce the same hash. Fixed length guarantees that, regardless of the size of the input, the output will have a constant and specific length, depending on the hash function used (Sharma, 2024). For example, SHA-256 will always generate a 256-bit hash. Computational efficiency means that the hash calculation is fast, no matter the size of the input data. Resistance to collisions means that it is extremely unlikely for two distinct inputs to generate the same hash, while resistance to preimages makes it practically impossible to deduce the original input if only the resulting hash is known. Finally, resistance to second preimages ensures that, given a combination of input and hash, it is extremely difficult to find another distinct input that produces the same hash.

Resistance to preimages is a fundamental property that ensures data integrity and security by making it practically impossible to deduce the original input from a hash, except through a brute force attack. Mathematically, given a hash function H(x) that takes an input x and produces a hash h, this property implies that, given h, finding an input x such that H(x) = x is computationally infeasible (fiveable, 2024). For example, in the case of a 256-bit hash, there

would be 2^{256} possible input values, making the calculation difficult due to the massive number of combinations.

Similarly, resistance to second preimages is a property where, given an input-hash pair, it is extremely difficult to find a second distinct input that produces the same hash. This challenge lies in preventing any other input from colliding with the original hash. This property is essential to protect data integrity, as it prevents an attacker from replacing the original information with another that generates the same hash (fiveable, 2024). Mathematically, if you have an input value x1 and its hash H(x1), it is extremely complex to find another input x2 such that H(x1) = H(x2).

Another essential property of hash functions is resistance to collisions. This feature is crucial to prevent tampering in critical applications, as it ensures that two distinct inputs cannot produce the same hash value. This is especially challenging to achieve, and the difficulty of preventing collisions increases with the size of the possible input combinations. However, due to the "birthday paradox," the probability of collisions increases as the search space grows, which reduces the effort required to just 2^n/2 attempts for an n-bit hash (Bellare, Kohno, 2003). This is why functions like MD5 and SHA-1 are no longer used, as they are vulnerable to collision attacks, while SHA-256 and SHA-3 are considered more secure against these advanced attacks.

The properties of cryptographic hash functions are crucial for information security, as they ensure the integrity, authenticity, and confidentiality of data. For example, even if the passwords of a database were compromised, resistance to preimages prevents an attacker from recovering the original passwords from the hashes without resorting to a brute force attack. Similarly, resistance to second preimages is critical in document verification systems, as it ensures that even if an attacker knows both the document and its hash, they cannot find another document that generates the same hash. Resistance to collisions is also essential in applications such as digital certificates and blockchain, as it makes it difficult for an attacker to find two distinct transactions with the same hash. This protects the integrity of data in the blockchain, as an attempt to alter a block would break the chain of cryptographic links with subsequent blocks due to the hash change, thus ensuring the integrity of the entire chain.

As mentioned earlier, there are some hash functions that are more common than others. For example, there is message digest 5, known as MD5. This function was developed in 1994 by Ron Rivest and it works by breaking a text into blocks of a length of 512 bits. Then, each block is transformed into a 128 bit hash value, usually represented as a number of 32 hexadecimal symbols (Thompson, 2005). Another common function is Secure Hash Algorithm 1, commonly known as SHA-1, this function was created by the United States National Security Agency in 1995. It generates a 160 bit hash value, 32 bits bigger than the MD5, represented by a number of 40 hexadecimal symbols (Ballejos, 2024), this was mainly used to verify authenticity and integrity of data on digital signatures and certificates (Fischer, 2023). Lastly, another Secure Hash Algorithm, there is the 2 family (SHA-2 family). Also developed by the NSA, in 2006, is an improvement on SHA-1 (Ballejos, 2024). Its knows as a “family” since it includes some variants like SHA-256, SHA-512 or SHA-384, the first two being the most common ones to use creating hash values of 256 and 512 bits respectively, they are used for securing SSL and TLS certificates and for blockchain technology (Ballejos, 2024).

All of these functions have their differences and similarities. One key aspect is the length of the hash value, as stated previously, MD5 produces a 128 bit hash value, SHA-1 a 160 and SHA-2 varies depending on the variant one is applying. Of course a higher hash value gives a better security for collisions, cryptanalysis and brute force attacks and overall improves in security, but it also comes with a higher computational cost and complexity in its implementation. Speaking of security, both MD5 and SHA-1 have been proven to be vulnerable to collision attacks (Ballejos, 2024) (Thompson, 2004). A collision occurs when two different inputs give the same hash value, and a hacker can take advantage of this to undermine the security of a hash (Lake, 2023). The SHA-2 family is also vulnerable to a collision attack, but it's less likely to happen than with the other two functions (specially when using a SHA-512). Finally, all of them use a Merkle–Damgård construction in their algorithm (Pardo & Gomez-Rodriguez, 2015). This takes an arbitrary length of bits and transforms it into a fixed length hash output. It follows a process of padding, vector initialization, splitting into blocks, a compression function and a final output (Pardo & Gomez-Rodriguez, 2015).

Taking into consideration what was said about MD5 and SHA-1 being vulnerable to collision attacks, it makes sense that both of these functions have been deprecated, and are no longer recommended to be used. Nowadays, functions like SHA-2 and SHA-3 (released in 2015) are more secure and the computational advances in technology have made them feasible to implement. Focusing on SHA-3, this new hash algorithm has been approved by the NIST (National Institute of Standards and Technology). This new method implements a sponge construction. This kind of method implements a permutation function called keccak-f that maps all possible input bit combinations to all possible combinations of that same number of bits (Patricio, 2024). Although SHA-2 is not insecure, and it's still in use, SHA-3 has been seen as a complement to its predecessor. Similar to SHA-2, it also consists of a family, and depending on which variant one chooses, the size of the bit varies. The most common ones are SHA3-224, SHA3-256, SHA3-512.

Digital signatures authenticate the sender’s identity and validate a message’s integrity, which is crucial for secure online transactions, including emails, credit card processing, and document exchange (CISA, 2021). This is achieved by encrypting a message digest with the sender’s private key, which the receiver then verifies using the sender’s public key. This process ensures both the sender’s authenticity and that the message remains unchanged (IBM, 2021).

The digital signature process almost always involves a secure cryptographic hash function, which transforms a message of any length into a fixed-length message digest on which the signature scheme is applied (Stinson & Peterson, 2019). This approach improves the process's efficiency because directly signing messages would be computationally expensive; ensures integrity, as any change in the message produces a different hash; and allows for authentication and non-repudiation because of the use of the use of a private key to sign the message (EITCA, 2024).

Hash functions play a crucial role in password storage, as they help protect passwords even if an attacker gains access to the application database. Unlike encryption, which can reveal plaintext data when decrypted, hashing is a one-way operation, making it practically

impossible to reverse-engineer the original password from the hashed value (OWASP, 2024).

Secure password storage relies on well-known hashing algorithms that transform plaintext passwords into hashed values stored in a database. This process often involves combining the password with a unique random string, called a salt, which is added before hashing (Oberai, 2023). The salt increases security by ensuring that identical passwords produce unique hashes, making it difficult for attackers to exploit repeated passwords. Both the hashed password and its corresponding salt are stored in the database (Lukic, 2024). Common password hashing algorithms include Argon2, Bcrypt, and SHA, among others (Oberai, 2023).

Another important application of hash functions comes with blockchain technology, where hashing is used to ensure data integrity and avoid fraudulent transactions. Essentially, blockchain is a public distributed ledger system of transactions, and each one is registered in a single block, whereas each block can contain many transactions (Stinson & Peterson, 2019). In this context, hashes are used on the header of each new block to hash the previous block, creating a secure linked system, making it very difficult to change the information of one block without altering all the subsequent blocks (Bybit, 2023).

In practice, different blockchain technologies use different hashing algorithms, but the idea remains the same. For example, Bitcoin uses the Secure Hashing Algorithm 256 (SHA-256) generating a fixed 256-bit hash value. Furthermore, other blockchains like Ethereum use their own hashing algorithms (Bybit, 2023).

Hash functions are crucial in Hash-based Message Authentication Code (HMAC) mechanisms, where a unique hash function and shared secret key provide both authentication and data integrity (Solomon, 2023). This approach ensures messages are secure and untampered during transmission, adding an essential layer of verification.

Unlike digital signatures or asymmetric cryptography, which rely on distinct public-private keys, HMAC employs a shared secret. At the start of communication, both parties agree on a secret key and a specific hash function. They then hash the message along with the secret key to create the HMAC, which accompanies the message to the recipient. The recipient calculates their own HMAC and compares it to the received one, verifying both authenticity and integrity (Okta, 2024).

HMAC’s role is fundamental in numerous file transfer protocols and has been a FIPS standard since 2002 (Stinson & Peterson, 2019). Due to its efficiency and security, HMAC is widely used in protocols like FTPS, SFTP, and HTTP, where it ensures that messages of any length are hashed into short digests that verify authenticity and integrity (Villanueva, 2024).

References

Ballejos, L. (2024). SHA1 vs SHA2 and SHA256: Main Differences | NinjaOne. NinjaOne. https://www.ninjaone.com/blog/sha1-vs-sha2-and-sha256/#:~:text=SHA1%2C%20or%20Secure%20Hash%20Algorithm,a%2040-character%20hexadecimal%20number

Bellare, M., & Kohno, T. (2003). *Hash Function Balance and its Impact on Birthday Attacks*. Cryptology ePrint Archive, Paper 2003/065. https://eprint.iacr.org/2003/065

Black Duck Software. (n.d.). *Cryptographic Hash Functions: Basics & Applications*. Black Duck Blog. https://www.blackduck.com/blog/cryptographic-hash-functions.html

Bybit. (2023). Explained: What Is Hashing in Blockchain?. https://learn.bybit.com/blockchain/what-is-hashing-in-blockchain/

Cybersecurity & Infrastructure Security Agency CISA. (2021). Understanding Digital Signatures. https://www.cisa.gov/news-events/news/understanding-digital-signatures

European Information Technologies Certification Academy EITCA. (2024). What role does the hash function play in the creation of a digital signature, and why is it important for the security of the signature?. https://eitca.org/cybersecurity/eitc-is-acc-advanced-classical-cryptography/digital-signatures/digital-signatures-and-security-services/examination-review-digital-signatures-and-security-services/what-role-does-the-hash-function-play-in-the-creation-of-a-digital-signature-and-why-is-it-important-for-the-security-of-the-signature/

Fiiveable. (n.d.). *Properties and Applications of Hash Functions*. Fiveable. https://fiveable.me/cryptography/unit-5/properties-applications-hash-functions/study-guide/1n7O5lKPGjIb9njd

Fisher, T. (2023). SHA-1: What It Is & How It's Used for Data Verification. Lifewire. https://www.lifewire.com/what-is-sha-1-2626011

Gornitzka, A. (2023, February 16). *MD5 vs. SHA-1 vs. SHA-2 - Which is the most secure hashing algorithm?* freeCodeCamp. https://www.freecodecamp.org/news/md5-vs-sha-1-vs-sha-2-which-is-the-most-secure-encryption-hash-and-how-to-check-them/

IBM. (2021). Digital signature overview. IBM B2B Advanced Communications. https://www.ibm.com/docs/en/b2badv-communication/1.0.0?topic=overview-digital-signature

Lake, J. (2023). What is a collision attack? Threats to digital signature security. Comparitech. https://www.comparitech.com/blog/information-security/what-is-a-collision-attack/

Lukic, D. (2024). What is Password Hashing and why is it important. SuperTokens. https://supertokens.com/blog/password-hashing-salting

Oberai, A. (2023). How password hashing algorithms keep your data safe. Appwrite. https://appwrite.io/blog/post/password-hashing-algorithms

Okta. (2024). HMAC (Hash-Based Message Authentication Codes) Definition. Okta. https://www.okta.com/identity-101/hmac/

Open Web Application Security Project OWASP. (2024). Password Storage Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Pardo, J. L. G., & Gómez-Rodríguez, C. (2015). The sha-3 family of cryptographic hash functions and extendable-output functions. Maple document.

Patricio, H. (2022). Crea hashes resistentes a balas con Keccak (SHA-3). The Dojo MX Blog. https://blog.thedojo.mx/2022/10/12/crea-hashes-resistentes-a-balas-con-keccak-tambien-llamado-sha-3.html

Pickl AI. (n.d.). *Understanding Hash Function*. Pickl AI Blog. https://www.pickl.ai/blog/understanding-hash-function/

Solomon, S. (2023). What Is HMAC (Hash-Based Message Authentication Code)?. Frontegg. https://frontegg.com/blog/hmac

Stinson, D. & Peterson, M. (2019). Cryptography Theory and Practice (4th ed.). CRC Press.

Thompson, E., T. (2005). MD5 collisions and the impact on computer forensics. Digital Investigation, 2(1).

Villanueva, JC. (2024). What Is HMAC (Hash-based message authentication code), And How Does It Secure File Transfers?. JScape. https://www.jscape.com/blog/what-is-hmac-and-how-does-it-secure-file-transfers

Part 2: Secure Communication System Design

System architecture

The architecture of the proposed system is designed to ensure confidentiality, integrity, authenticity, and non-repudiation of messages. The system implements a hybrid cryptographic approach that combines asymmetric encryption (for key exchange and authentication) with symmetric encryption (for message transmission). With this hybrid approach, we establish an efficient and secure communication system.

A. Actors

- **Endpoints (Alice and Bob):** These are the actors that take part in the communication process. In this case, Alice and Bob. Each party operates as a secure client with modules for encryption, digital signing, key generation, and secure storage. Each one is responsible for performing cryptographic operations locally and handling the communication with the other party over an encrypted channel.
- **Network Communication Channel:** This is the network link over which Alice and Bob exchange messages. All communication on this channel is encrypted and authenticated, taking into account that the network is potentially insecure.
- **Certificate Authority (CA):** This is a trusted authority that issues X.509 certificates for identity verification. The CA ensures that each party is communicating with a verified user.
- **Hardware Security Module (HSM):** If available, it is used for generation and storage of private keys. Also, it's important for performing sensitive operations such as signing without exposing private keys in memory.

B. Protocol Specification

To achieve the desired security properties, the following protocols and standards are used:

- **Asymmetric Key Exchange Protocol:** RSA-4096 is used for initial authentication and key exchange signatures.
- **Symmetric Encryption Protocol:** AES-256-GCM ensures confidentiality and integrity of the message during transmission.
- **Message Integrity Protocol:** HMAC-SHA256 ensures message integrity and authenticity.
- **Identity Verification:** X.509 certificates are used to verify the identity of both parts. Certificates are signed by the trusted authority.

C. Session Management

1. **Session Initialization:**
    - Mutual authentication using X.509 certificates and Diffie-Hellman key exchange.
    - Ephemeral session keys derived for each session to ensure forward secrecy.
2. **Session Maintenance:**
    - Session specific parameters like shared key and ID's are negotiated and monitored.
    - Continuous integrity and authenticity checks to ensure security.
3. **Session Termination:**
    - Clear all session keys from memory and notify the other party.

- Keep a log of termination events including status, duration and any anomalies.
- Both parties acknowledge session termination to effectively close the session.

D. Key Management

1. Key Generation

The system requires both asymmetric and symmetric keys to secure communications. Keys are generated according to the following protocols:

- **Asymmetric Keys:** RSA-4096 asymmetric keys are generated locally by Alice and Bob. They use a cryptographically secure pseudorandom number generator. Private keys are kept in a HSM if available.
- **Symmetric Keys:** For each communication session, a unique 256-bit AES key is generated with a Diffie-Hellman shared secret. The key is unique to each session, providing Perfect Forward Secrecy.

2. Key Exchange Protocol

- **Initial Exchange:** Each party presents a X.509 certificate signed by a trusted authority to verify their identity. Also, certificate pinning is used to secure the verification process.
- **Session Key Exchange:** Each party generates an ephemeral Diffie-Hellman key pair, the public keys are exchanged. When the public keys are exchanged and verified, each side computes the shared secret.

3. Key Storage

Key storage should be carefully managed to prevent unauthorized access and ensure that keys are not exposed to hackers.

- **Asymmetric Private Keys:** These keys should be stored in a secure hardware module, if available. Aldo, these keys should only be accessible to authorized applications in a secure environment.
- **Symmetric Session Keys:** These keys should be stored in a secure memory only for the duration of the session, and should never be written to disk. Also, these keys should be deleted from memory as soon as the session terminates.

4. Key Rotation

Regular rotation of keys is implemented to limit the exposure in case of key compromise. Rotation policies are as follows:

- **Asymmetric keys:** Rotated annually or upon compromise.
- **Session keys:** A new key is generated for each session.

E. Security Features

The proposed communication system includes mechanisms for data integrity (HMAC-SHA256), replay attack prevention (timestamp validation and message IDs), and Perfect Forward Secrecy through ephemeral Diffie-Hellman exchanges. The HMAC-SHA256 tag is attached to each message for integrity checks, while timestamps and unique message IDs prevent unauthorized replays.

F. Data Flow

The proposed communication system operates in two main phases: Key Exchange and Message Transmission.

- **Key Exchange Phase**

This phase establishes a secure session by generating a shared session key using Diffie-Hellman key exchange with RSA signatures for authenticity and X.509 certificates for identity verification.

1. **Mutual Authentication and Key Exchange Process:**
    - **Step 1:** Alice and Bob exchange their X.509 certificates, each signed by a trusted Certificate Authority (CA).
    - **Step 2:** Alice generates an ephemeral Diffie-Hellman (DH) key pair and sends her DH public key to Bob, signed with her RSA private key. Bob verifies Alice's RSA signature and validates her certificate against the CA.
    - **Step 3:** After verifying Alice's certificate and signature, Bob generates his own ephemeral DH key pair and responds with his DH public key, signed with his RSA private key. Alice verifies Bob's RSA signature and validates his certificate against the CA.
    - **Step 4:** Both Alice and Bob use the verified DH public keys and their own DH private keys to independently generate the shared session key.
    - **Step 5:** Alice and Bob now share a unique session key for symmetric encryption, which they use to securely communicate during the message transmission phase.

- **Message Transmission Phase**

  Once the session key is established, messages are exchanged securely using AES-256-GCM (for confidentiality and integrity), and RSA-PSS signatures (for authenticity and non-repudiation).

  1. **Message Encryption**
      - **Step 1:** Alice prepares the message, including a unique message ID, a timestamp to prevent replay attacks, and a header with metadata.
      - **Step 2:** The message is compressed, then encrypted using AES-256-GCM with a unique initialization vector (IV) for each message, ensuring confidentiality.
      - **Step 3:** Alice generates an HMAC-SHA256 to verify the message's integrity and signs the encrypted message using her RSA private key.
      - **Step 4:** Alice transmits the encrypted, signed message with the header to Bob.
  2. **Message Decryption**
      - **Step 1:** Bob receives the message and verifies the timestamp and message ID to prevent replay attacks.
      - **Step 2:** Bob verifies Alice's signature using her public key, ensuring authenticity and non-repudiation.
      - **Step 3:** Bob recalculates the HMAC, and compares it with the received HMAC to ensure integrity.
      - **Step 4:** Bob decrypts the message using the shared session key and decompresses it to retrieve the original message content.
  3. **Acknowledgements**

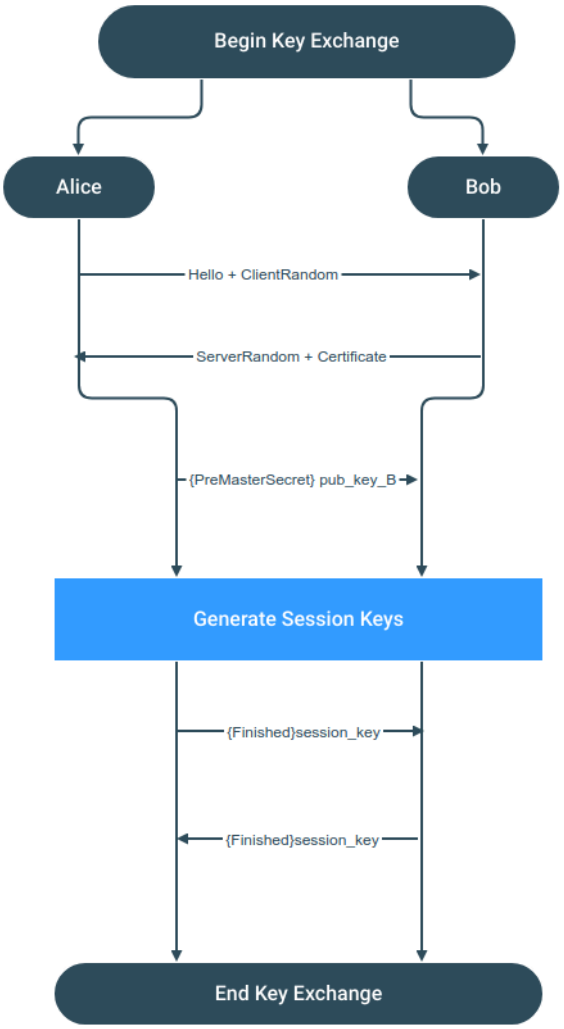
- **Step 1:** Bob generates a receipt referencing the original message ID, the timestamp and adds a status.
- **Step 2:** Bob sends the receipt to Alice, providing proof of delivery.

G. Error Handling

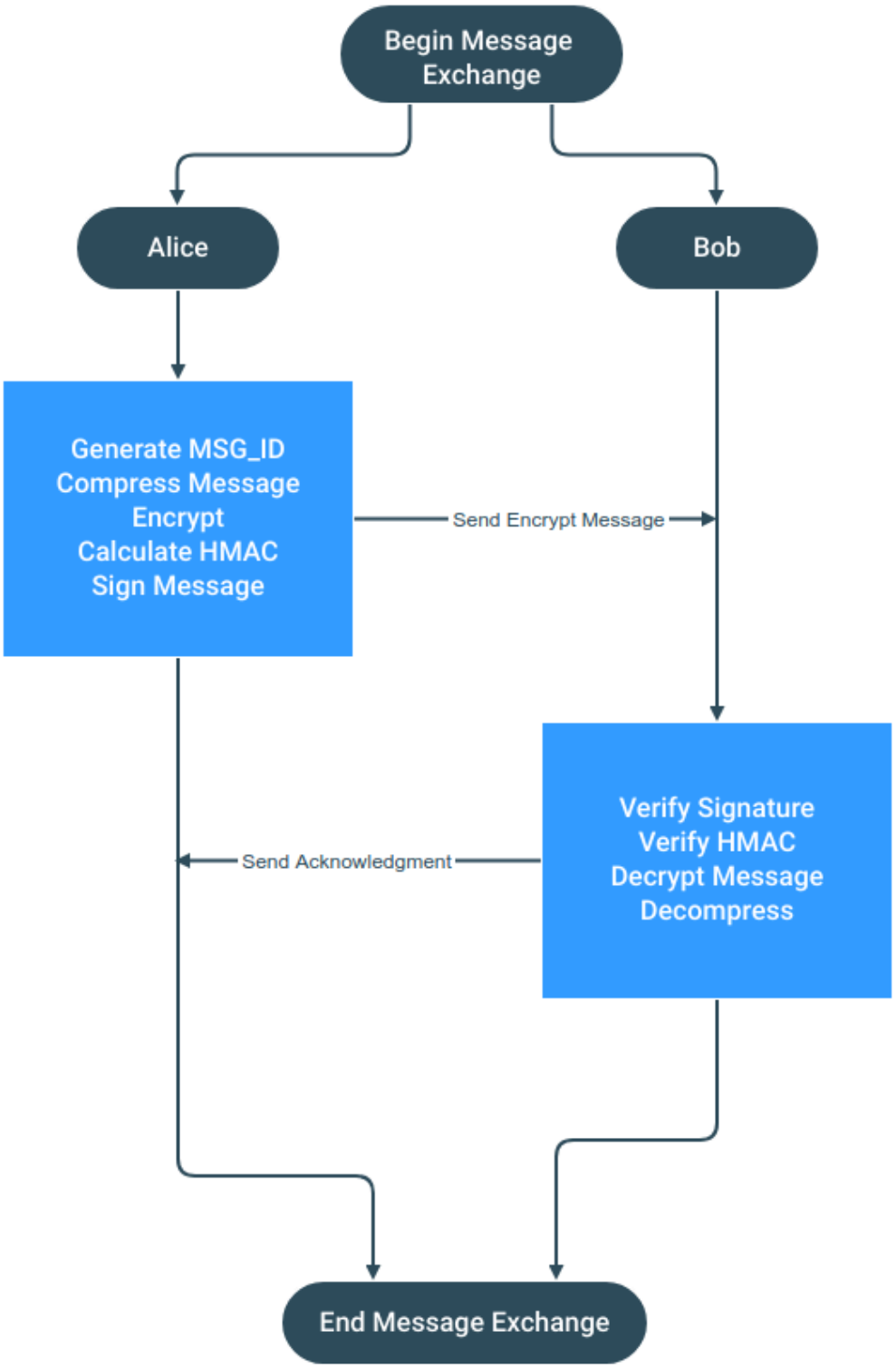
- **Message Decryption Failure**
  a. **Response:** Log failure details and send error response to the sender.
  b. **Retry:** Apply exponential backoff with a maximum of 3 tries before issuing an alert.
- **Signature Verification Failure**
  a. **Response:** Discard the message immediately and log the incident.
  b. **Retry:** No retry is attempted if verification fails.
- **Timestamp Validation Error**
  a. **Response:** Reject the message if the timestamp falls outside of a determined time window to prevent replay attacks.
  b. **Alert:** Record the incident in the log, and monitor for replay attacks.
- **Key Exchange Failure**
  a. **Response:** Log information of the failure, and retry the key exchange up to a certain number of times.
  b. **Alert:** Issue an alert to the administrators if retries fail.

Flow diagrams

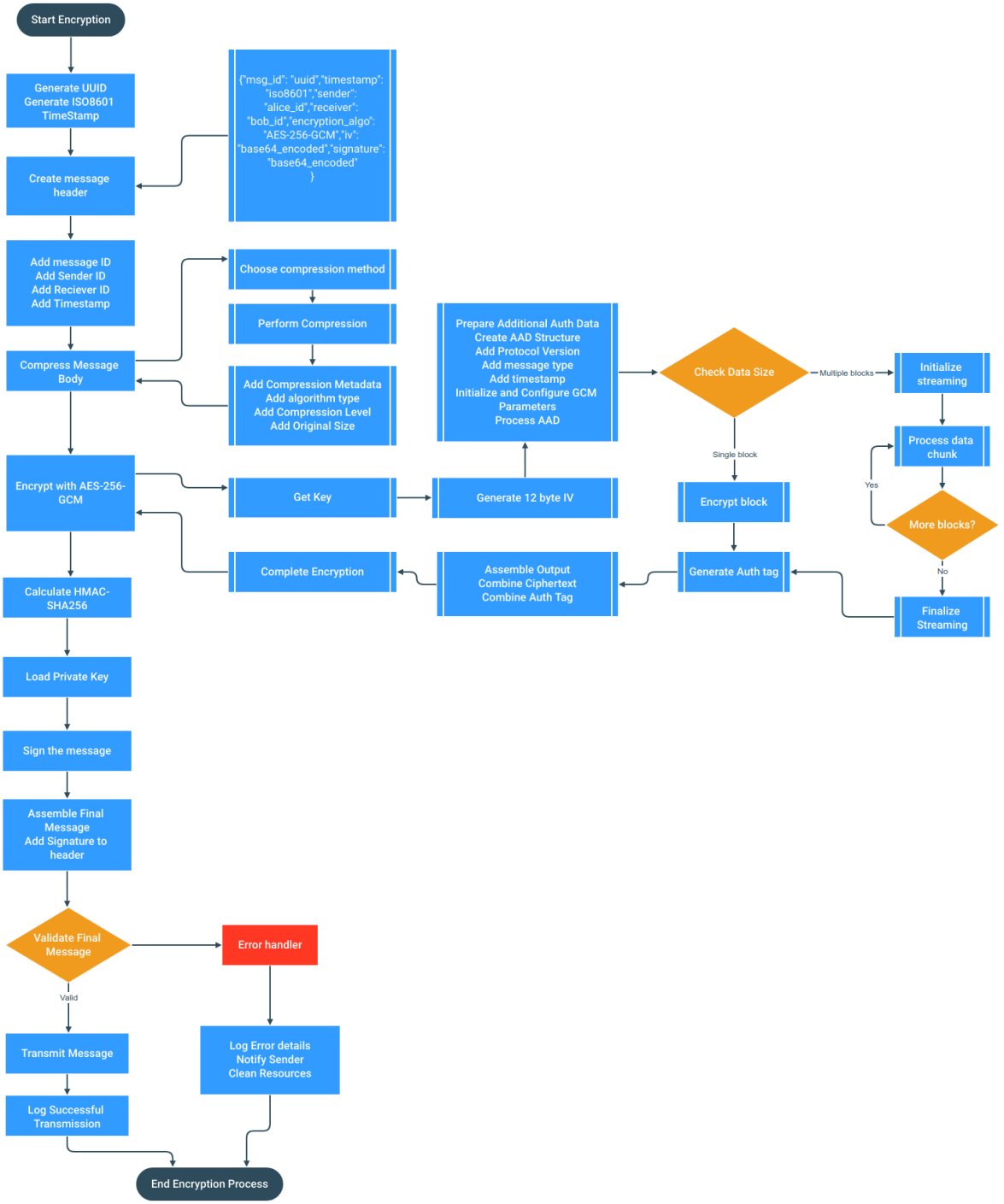
Key Exchange



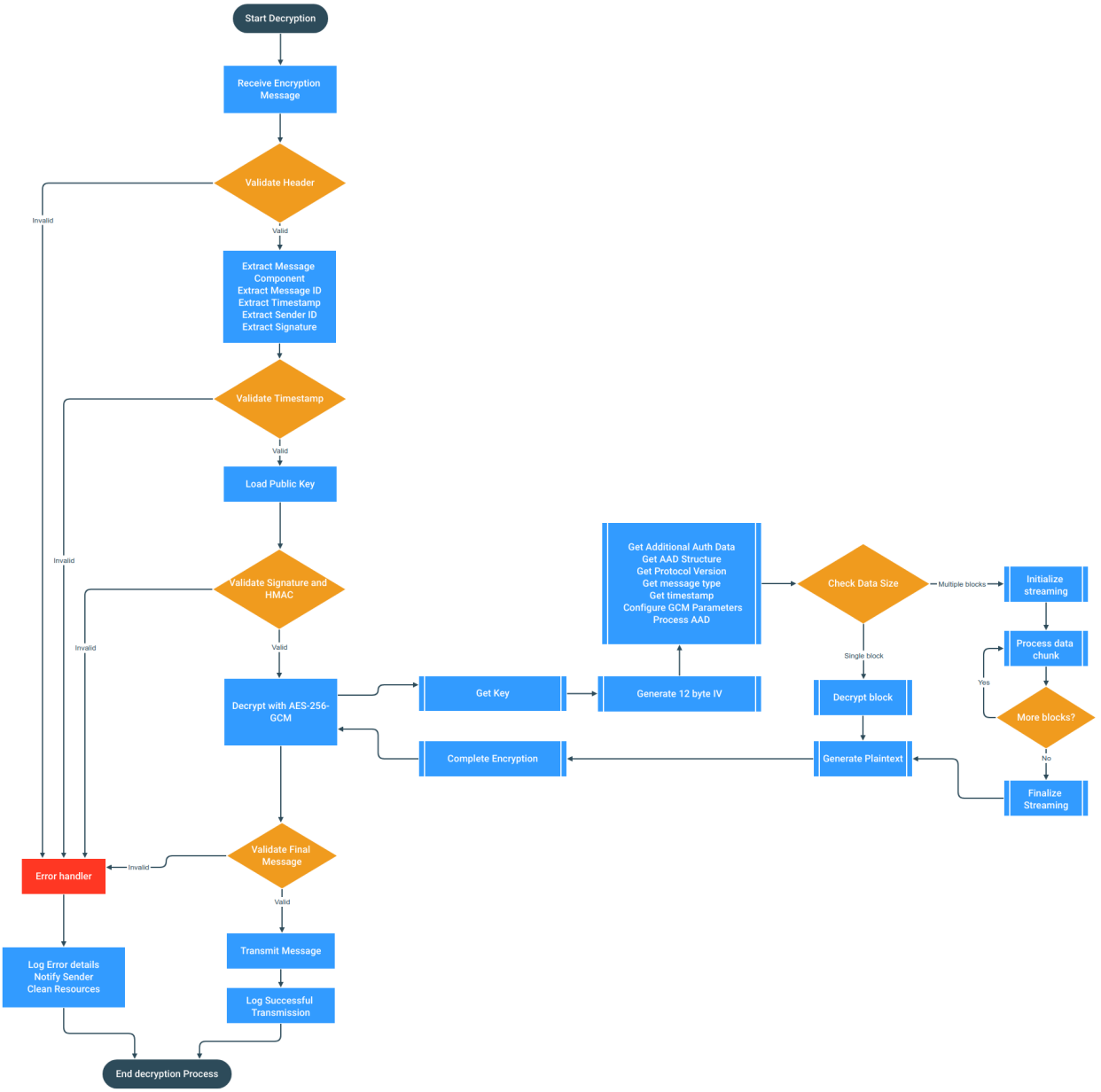
Message Exchange



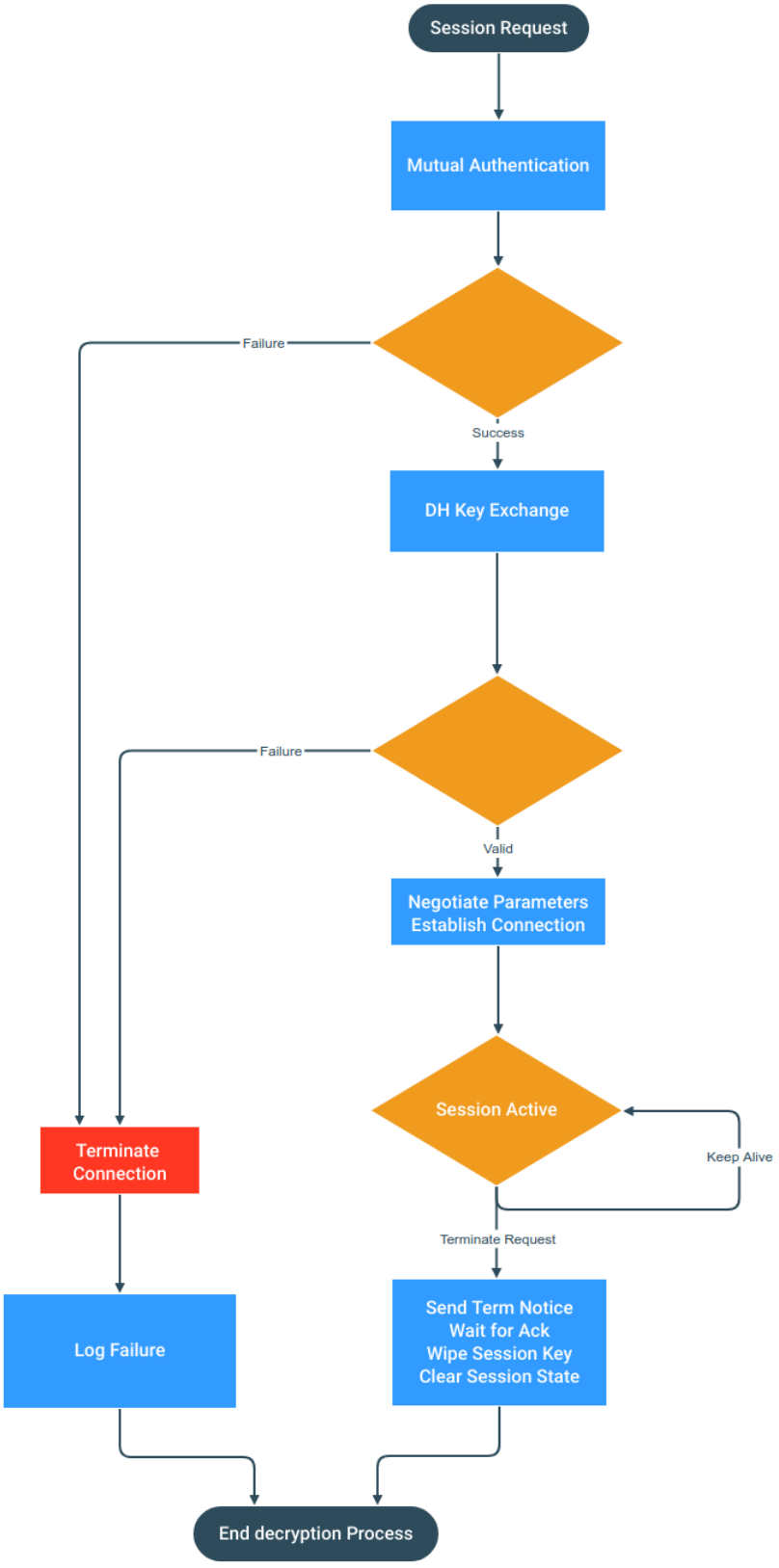
Message Encryption Process



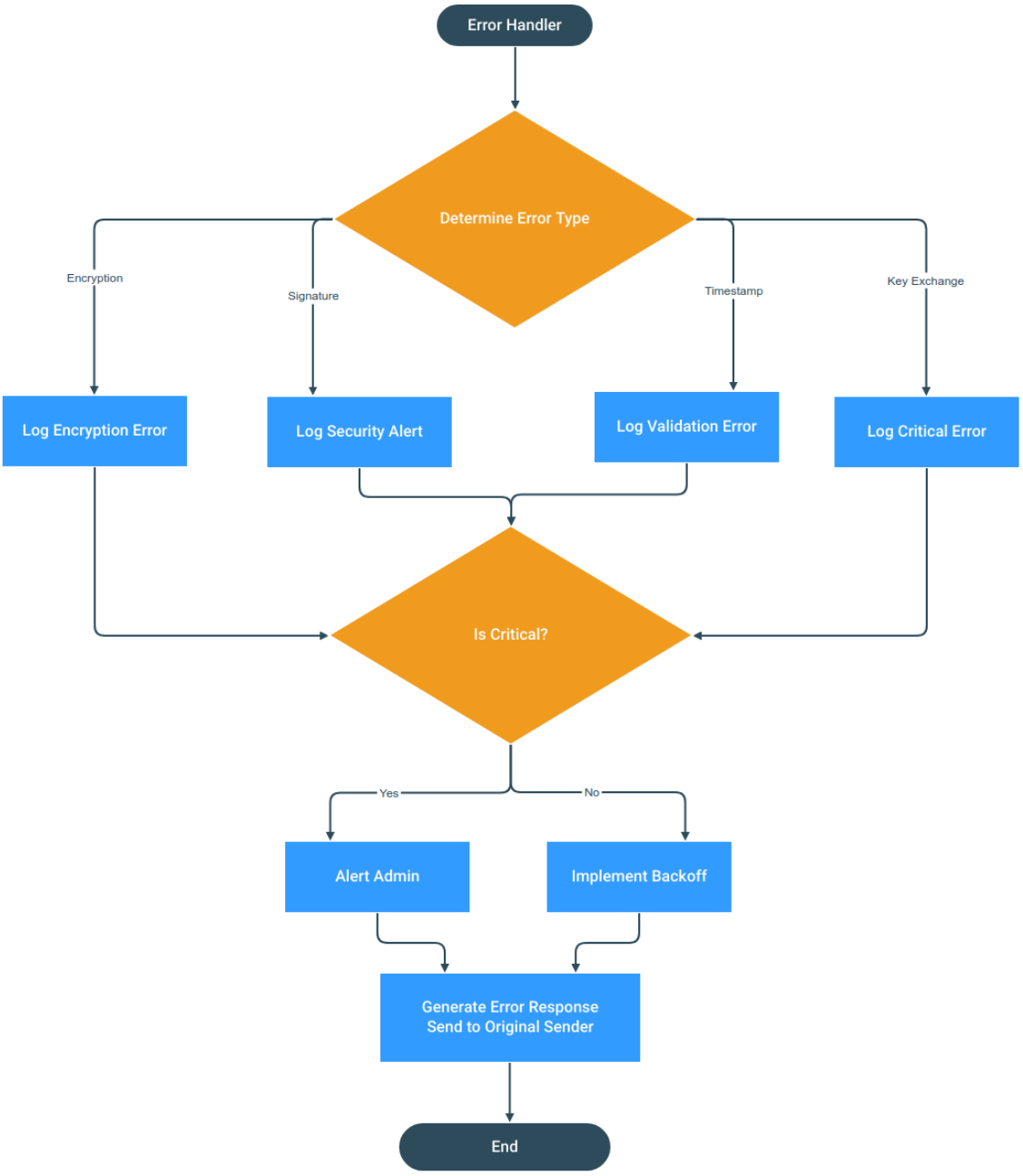
Message Decryption Process



Session Management



Error Handler



Security analysis

The communication system ensures the principles of confidentiality, integrity, authenticity, and non-repudiation for each message exchanged between Alice and Bob, through the following mechanisms:

1. Confidentiality

Confidentiality ensures that only Alice and Bob can read the content of the messages. This system employs a hybrid encryption approach, using RSA to securely exchange a symmetric session key, which is then used to encrypt the messages with AES. Alice generates an AES session key, which is encrypted using Bob's public key via RSA, ensuring that only he can decrypt it. Once the session key is established, AES-256-GCM is used to encrypt and decrypt the content of the messages. Perfect Forward Secrecy is additionally ensured through the use of ephemeral Diffie-Hellman keys for each session, preventing a compromised key from exposing past communications.

2. Integrity

Integrity ensures that messages between Alice and Bob are not altered during transmission. The system uses HMAC-SHA256 to detect any modifications in the data. The recipient verifies the message's integrity before attempting decryption by using a hash-based message authentication code (HMAC). If any part of the message is altered, the generated HMAC will not match, alerting the recipient that the message has been tampered with. This approach follows the Encrypt-then-MAC model, in which the message is encrypted and then authenticated, providing an additional layer of security against data manipulation.

3. Authenticity

Authenticity ensures that the message originates from the legitimate sender. To verify the sender's identity, the system implements digital signatures (RSA-PSS) on both the messages and Diffie-Hellman ephemeral keys. Alice and Bob sign their messages, allowing the recipient to confirm the sender's identity. Additionally, the use of X.509 certificates signed by a trusted certification authority ensures that both parties can verify each other's identity before establishing communication.

4. Non-Repudiation

Non-repudiation ensures that a sender cannot deny having sent a message. Each message and session exchange keys include a digital signature generated with the sender's private key, which cannot be forged by third parties. Additionally, a timestamp and a unique message ID are recorded for each communication, providing traceability and facilitating verification of messages in case of inconsistencies. This prevents the sender from denying participation in a communication at a specific time or claiming that the message was altered or created by someone else.

Attacks and Mitigations

The system is designed to withstand both active and passive attacks.:

Passive Attacks

- Network Sniffing: If an attacker intercepts the data traffic transmitted between Alice and Bob to try to read the messages, AES-256-GCM encryption ensures that only Alice and Bob can access the content.
- Traffic Analysis: Traffic analysis could reveal patterns in message frequency and size. However, this is partially mitigated through compression and the use of fixed-size messages, reducing the amount of information an attacker can infer.

Active Attacks

- Man-in-the-Middle (MitM) Attack: To prevent interception or manipulation of communication, the system uses mutual authentication through X.509 certificates and digital signatures on each key exchange. Additionally, the use of ephemeral Diffie-Hellman key exchange ensures confidentiality and authenticity in each session.
- Replay Attacks: If an attacker attempts to retransmit valid messages to deceive the receiver, the system employs a record of timestamps and unique message IDs. A sliding window technique with a time tolerance (+/- 5 minutes) ensures that repeated or out-of-time messages are not accepted.
- Session Hijacking: If an attacker tries to intercept and take control of an active session, mutual authentication and the generation of new keys for each session minimize the risk. Additionally, session keys are deleted immediately at the end of each session, limiting exposure.

Implementation considerations

The implementation of the secure communication system requires attention to vulnerabilities coming from the user's infrastructure, and potential performance optimizations. To ensure robustness and scalability while maintaining security, aspects like identity management, key handling, and system scaling must be taken into account.

- **Scalability Considerations**

  The proposed system relies on efficient and scalable management of cryptographic keys. Thus, it is advisable to utilize a secure and reliable public key infrastructure (PKI) that can handle a growing number of certificates, authentication requests, and session keys.

  One potential solution for achieving scalability comes with the deployment of a Public Key Infrastructure (PKI) through an actor like Google Cloud Certificate Authority Service (CAS). Google Cloud CAS provides a secure and reliable public key infrastructure. Furthermore, its cloud based nature allows for high scalability in managing and automating the issuance, renewal and revocation of certificates, which are essential for the proposed communication system.

  By leveraging the Google Cloud CAS, the system can:

  1. Simplify, automate, and customize the deployment and security of the CA.
  2. Establish, secure and operate a specific private PKI.
  3. Allow for an easy establishment, operation and customization of a PKI.

4. Reduce the need for a large in-house infrastructure.

- **Considerations for integration**
  1. **Integration with other Google Cloud Services:** The use of Google Cloud CAS allows for an easy integration with other Google Cloud Services, allowing the system to extend its functionality. This includes integration with identity management services, secure logging and monitoring.
  2. **Efficient Use of Resources:** The reliance on cloud services ensures that cryptographic operations and key management are handled efficiently, without the need for dedicated hardware. Google Cloud's cloud-based approach reduces the cost and complexity associated with maintaining on-premise systems while offering robust protection for sensitive data.
- **Cryptography Libraries**
  To implement secure cryptographic operations, such as encryption, decryption, and signing, several well-established libraries are available. For the cloud-based implementation, **OpenSSL** is a highly recommended library due to:
  - Its seamless integration with Google Cloud CAS, ensuring compatibility with cloud-native cryptographic operations.
  - A wide range of supported cryptographic algorithms, including symmetric encryption (AES), public-key encryption (RSA), and secure hashing (SHA-256).
  - Strong community support, ensuring regular updates and vulnerability patches.

Potential Vulnerabilities and Mitigation Measures

Identity Forgery

Since the system relies on X.509 certificates for mutual authentication, an attacker might attempt to manipulate certificates or impersonate one of the participants. Limiting communication to trusted certificates and verifying the integrity of digital signatures can help mitigate this risk.

Signature Forgery

If an attacker finds a vulnerability in the RSA-PSS implementation and uses it to forge signatures, the authenticity of the message could be compromised. This risk can be minimized by using algorithms that adhere to current security standards, such as those from NIST and FIPS.

Replay Attacks

Although the system addresses this risk through timestamps and unique message IDs, an inadequately configured sliding window could allow an attacker to capture and retransmit old messages in an attempt to validate them. Implementing a sliding window control that validates messages with timestamps and keeps a history of received messages is an effective measure to prevent these attacks.

Compromised Integrity in HMAC-SHA256

If the HMAC key is compromised, an attacker could modify messages undetected. As a countermeasure, it is recommended to derive a unique HMAC key for each session using HKDF (Key Derivation Function), ensuring that each key is unique and challenging to compromise.

Side-Channel Attacks

An attacker could attempt to extract information from the system by analyzing execution times, power consumption, or other behavioral patterns. To mitigate this risk, constant-time programming techniques and hardware protections can be used to reduce the information accessible through these channels.