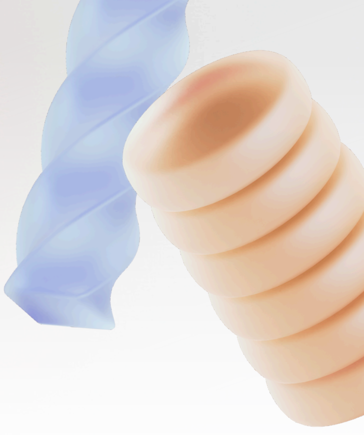

Encryption Algorithm Implementation

AES, HMAC-SHA256, and RSA

Done by:
Ahmed AbdelMaboud 231000224
Yahia Hany 231000412
Abdelrahman Yasser 231000102

Introduction

This document provides a detailed explanation and implementation guide for three fundamental cryptographic algorithms: **AES (Advanced Encryption Standard)** for symmetric encryption, **HMAC-SHA256 (Hash-based Message Authentication Code using SHA-256)** for message authentication, and **RSA (Rivest-Shamir-Adleman)** for asymmetric encryption. Each section includes step-by-step instructions, pseudocode, technical notes, and a comparative analysis of the algorithms.

AES Encryption

AES is a symmetric block cipher widely used for securing sensitive data. It operates on 128-bit blocks of data and supports key sizes of 128, 192, or 256 bits. This guide focuses on AES-128.

Step-by-Step Implementation:

1. **Key Expansion:** The 128-bit key is expanded into 11 round keys, each 128 bits long. This is done using the AES key schedule.
   - **Explanation:** The function takes the original 128-bit key and generates a series of round keys. Applies the S-box substitution to each byte. performs a cyclic permutation. is the round constant.
2. **Initial Round Key Addition:** XOR the input block with the first round key (the original key).
   - **Explanation:** This initial step combines the plaintext with the encryption key to begin the encryption process.
3. **Rounds 1-9:** Perform the following four transformations in sequence for each round:
   - **SubBytes:** Substitute each byte in the state array with a corresponding byte from the S-box.
     - **Explanation:** The transformation provides non-linearity to the cipher, making it resistant to linear cryptanalysis.
   - **ShiftRows:** Cyclically shift the bytes in each row of the state array. The first row is not shifted, the second row is shifted one byte to the left, the third row is shifted two bytes to the left, and the fourth row is shifted three bytes to the left.
     - **Explanation:** The transformation provides diffusion, spreading the influence of each byte across the state.
   - **MixColumns:** Perform a matrix multiplication on each column of the state array. This step involves Galois field arithmetic.

        - **Explanation:** The transformation further enhances diffusion, ensuring that each input bit affects multiple output bits after several rounds.
    - **AddRoundKey:** XOR the state array with the round key for that round.
        - **Pseudocode:** (Same as initial AddRoundKey)
4. **Round 10:** Perform SubBytes, ShiftRows, and AddRoundKey (no MixColumns).
5. **Output:** The resulting state array is the ciphertext.

Technical Notes:

- The S-box is a crucial component for introducing non-linearity.
- MixColumns is the most computationally intensive operation.
- AES is highly resistant to known attacks when implemented correctly.
- AES runtime executed in 0.000090 seconds (message = "Hello World").

HMAC-SHA256 Authentication

HMAC-SHA256 is a message authentication code that uses the SHA-256 hash function to ensure the integrity and authenticity of a message. It prevents tampering and verifies the sender's identity.

Step-by-Step Implementation:

1. **Key Padding:** If the key is longer than 64 bytes, hash it using SHA-256. If it is shorter, pad it with zeros to 64 bytes.
    - **Explanation:** This step ensures that the key is of a consistent length for processing.
2. **Inner Padding:** XOR the padded key with the (0x36 repeated 64 times).
    - **Explanation:** XORing with creates the inner key used for the first hashing operation.
3. **Outer Padding:** XOR the padded key with the (0x5c repeated 64 times).
    - **Explanation:** XORing with creates the outer key used for the second hashing operation.
4. **Inner Hash:** Concatenate the inner key with the message and hash the result using SHA-256.
    - **Explanation:** This creates a hash of the message combined with the inner key.
5. **Outer Hash:** Concatenate the outer key with the inner hash and hash the result using SHA-256.
    - **Explanation:** This performs a second hash, combining the outer key with the result of the inner hash to generate the final HMAC.
6. **Output:** The resulting hash is the HMAC-SHA256 value.

Technical Notes:

- HMAC provides stronger security than simply hashing a message with a key.
- The constants are carefully chosen to maximize security.
- SHA-256 is a collision-resistant hash function.
- HMAC-SHA256 executed in 0.000503 seconds (message = "Hello World").

RSA Encryption

RSA is an asymmetric encryption algorithm widely used for secure communication and digital signatures. It relies on the mathematical properties of prime numbers and modular arithmetic.

Step-by-Step Implementation:

1. **Key Generation:**
    - **Choose two distinct prime numbers, p and q.** These should be large, randomly chosen primes.
    - **Calculate n = p*q.** This is the modulus, and it's part of both the public and private keys.
    - **Calculate the totient of n, φ(n) = (p - 1) * (q - 1).**
    - **Choose an integer e such that 1 < e < φ(n) and gcd(e, φ(n)) = 1.** 'e' is the public exponent. A common choice is 65537 (2^16 + 1) because it's prime and has few set bits, making modular exponentiation faster. However, smaller values of 'e' might be vulnerable to certain attacks if padding is not implemented correctly.
    - **Compute the modular multiplicative inverse of e modulo φ(n), denoted as d. This means finding d such that (d*e) % φ(n) = 1.** 'd is the private exponent. The Extended Euclidean Algorithm is commonly used to find 'd.
    - **The public key is (n, e). The private key is (n, d). Keep p, q, and φ(n) secret.**
        - **Explanation:** This process involves finding two large primes, calculating the modulus and totient, and determining the public and private exponents. The security of RSA relies on the difficulty of factoring large numbers.
2. **Encryption:** To encrypt a message *m* (where *m* is an integer less than *n*), compute the ciphertext *c* as:
    - **c = me mod n**
        - **Explanation:** This step uses the public key to transform the plaintext message into ciphertext.
3. **Decryption:** To decrypt the ciphertext *c*, compute the original message *m* as:
    - **m = cd mod n**
        - **Explanation:** This step uses the private key to recover the original plaintext message from the ciphertext.

Technical Notes:

- RSA's security depends on the difficulty of factoring large numbers. Larger key sizes (e.g., 2048 bits or 4096 bits) provide greater protection.
- Padding schemes like PKCS#1 v1.5 padding or OAEP are crucial to prevent various attacks.
- The choice of 'e' can impact performance; 65537 is a common choice for its speed.
- Modular exponentiation can be efficiently computed using the square-and-multiply algorithm.
- RSA executed in 0.000061 seconds (message = “Hello World”).

Algorithm Comparison Workflow

This section provides a comparative analysis of AES, HMAC-SHA256, and RSA algorithms. The following table summarizes the key characteristics of each algorithm.

Algorithm	Purpose	Key Type
AES	Symmetric Encryption	Secret Key
HMAC-SHA256	Message Authentication	Secret Key
RSA	Asymmetric Encryption (Key Exchange)	Public/Private Key Pair

Key Insights:

- **AES** is suitable for encrypting large amounts of data due to its speed.
- **HMAC-SHA256** ensures that a message has not been tampered with and verifies the sender's identity. It's essential for secure communication protocols.
- **RSA** is used for key exchange and digital signatures. It is slower than symmetric encryption algorithms but provides the benefit of asymmetric cryptography.

These algorithms are often used in combination to provide comprehensive security. For instance, RSA can be used to exchange the AES key securely, and then AES can be used to encrypt the data. HMAC ensures the integrity of the encrypted data.

Algorithm Interaction Visualization

The following outlines how these algorithms interact in a typical secure communication scenario:

1. **Key Exchange (RSA):** The sender and receiver use RSA to securely exchange an AES session key.
2. **Data Encryption (AES):** The sender encrypts the message using the AES session key.
3. **Message Authentication (HMAC-SHA256):** The sender calculates the HMAC-SHA256 of the encrypted message using a shared secret key (or a key derived from the AES session key).
4. **Transmission:** The sender transmits the encrypted message and the HMAC-SHA256 value to the receiver.
5. **Verification (HMAC-SHA256):** The receiver calculates the HMAC-SHA256 of the received encrypted message using the same shared secret key.
6. **Integrity Check:** The receiver compares the calculated HMAC-SHA256 value with the received HMAC-SHA256 value. If they match, the message has not been tampered with.
7. **Decryption (AES):** The receiver decrypts the encrypted message using the AES session key.

Conclusion

This document provided a detailed overview of AES encryption, HMAC-SHA256 authentication, and RSA encryption. Understanding the implementation and interaction of these algorithms is crucial for building secure systems. Remember to implement appropriate security measures, such as key management and padding schemes, to mitigate potential vulnerabilities. Always stay up-to-date with the latest security best practices and research to ensure the continued robustness of your cryptographic implementations.