

Assignment 3: Cryptography

Sahishnu M A (20BCE7273)

Symmetric Key Algorithm:

Symmetric Key algorithms make use of one secret key, shared between the sender and the receiver to encrypt and decrypt the data being secured. Although Symmetric Key algorithms are faster than Asymmetric Key algorithms, they are comparatively less secure, due to only one key being present, that is known to both parties.

An overview of how the algorithm works:

1. **Key Generation:** Both the sender and receiver agree on a secret key before any communication takes place. This key is to be kept confidential and secure.
2. **Encryption:** The sender uses the secret key and the symmetric key algorithm to convert the plaintext (original message) into a ciphertext (encrypted message). The algorithm typically involves several rounds of substitution, permutation, and other operations to completely encrypt the original data.
3. **Transmission:** The sender transmits the ciphertext to the receiver over what might be an insecure channel, such as the internet.
4. **Decryption:** The receiver uses the same secret key and the symmetric key algorithm to convert the received ciphertext back into the original plaintext. The algorithm performs the reverse operations of the encryption process to recover the original data.

An example of Symmetric Key algorithm is the **DES (Data Encryption Standard)**, which was developed by the IBM in the 1970's.

An overview of how the DES algorithm works:

1. **Key Generation:** A 64-bit secret key is used for encryption and decryption. However, only 56 bits are used for the actual encryption, while the remaining 8 bits are used for error detection and correction. The key undergoes a process called key scheduling, which generates 16 sub keys, each 48 bits long.
2. **Initial Permutation (IP):** The plaintext (64 bits) is first subjected to an initial permutation. This rearranges the bits according to a fixed permutation table.
3. **Feistel Network:** DES uses a Feistel network structure, which involves multiple rounds of processing. In each round, the 64-bit plaintext is divided into two 32-bit halves: the left half (L0) and the right half (R0). The Feistel network consists of the following steps: Expansion, Key Mixing, Substitution, Permutation, XOR and Swap.
4. **Final Round:** After 16 rounds of the Feistel network, the left and right halves (L16 and R16) are swapped, and a final permutation called the inverse initial permutation (IP^{-1}) is applied.

5. **Ciphertext:** The final swapped halves (L16 and R16) are combined to form the 64-bit ciphertext.

Key Strengths and Advantages of DES:

1. **Efficiency:** The DES was tailored to be implemented in hardware and tuned to be efficient in encryption and decryption operations. Its simple structure allowed for fast processing, which was important during its time as computational resources were limited.
2. **Widely used:** The DES was widely adopted, enabling it to be compatible across platforms. This allowed secure communication between devices through DES.

Vulnerabilities and Weaknesses:

1. **Key Size:** DES has a relatively small key size of 64 bits, of which only 56 are used for encryption and decryption. With advancements in computation power, it has become vulnerable to exhaustive key search attacks or brute force attacks.
2. **Sub key Generation:** The key scheduling algorithm used in DES to generate the 16 sub keys from the original 56-bit key has known weaknesses. These vulnerabilities have been exploited to weaken the security of DES further.

Real world implementation of DES:

The use of the DES algorithm was made mandatory for all financial transactions of the U.S. government involving electronic fund transfer, including those conducted by member banks of the Federal Reserve System.

Asymmetric Key Algorithm:

Asymmetric key algorithms make use of a pair of mathematically related keys, i.e. a **Public key** and a **Private key**. Unlike the symmetric key algorithm, which uses the same key for encryption and decryption, this algorithm uses different keys for the operations.

An overview of how the algorithm works:

1. **Key Generation:** The first step is to generate a key pair consisting of a public key and a private key. The keys are mathematically related but computationally hard to derive one from the other. The public key is made publicly available, while the private key is kept secret.
2. **Encryption:** To send an encrypted message to a recipient, the sender uses the recipient's public key to encrypt the plaintext. The encryption process transforms the plaintext into a ciphertext, which can only be decrypted using the recipient's corresponding private key.
3. **Transmission:** The sender transmits the encrypted ciphertext to the recipient over an insecure channel, such as the internet or a network. Since the encryption was performed using the recipient's public key, the ciphertext remains secure even if intercepted by an attacker.

4. **Decryption:** Upon receiving the encrypted ciphertext, the recipient uses their private key to decrypt the message. Only the private key corresponding to the public key used for encryption can successfully decrypt the ciphertext. The decryption process recovers the original plaintext message.

An example of the Asymmetric key algorithm is the **RSA (Rivest-Shamir-Adleman)**, which was developed in 1977.

An overview of how the RSA algorithm works:

1. **Key Generation:** RSA employs a key pair consisting of a public key and a private key. The public key is used for encryption, while the private key is used for decryption. The key generation process involves selecting two large prime numbers, computing the modulus and Euler's totient function, and finding the public and private exponents.
2. **Encryption:** To encrypt a message, the sender uses the recipient's public key to transform the plaintext into ciphertext. This process involves modular exponentiation, where the plaintext is raised to the power of the public exponent and then reduced modulo the modulus.
3. **Decryption:** Upon receiving the ciphertext, the recipient applies their private key to perform modular exponentiation, recovering the original plaintext from the encrypted message.
4. **Digital Signatures:** RSA is also used for digital signatures. The signer uses their private key to encrypt a hash of the message, creating a signature. The recipient can verify the signature by decrypting it using the signer's public key and comparing it with a recalculated hash.
5. **Key Exchange:** RSA facilitates secure key exchange between two parties. One party encrypts a random symmetric key using the recipient's public key and sends it. The recipient decrypts the received ciphertext using their private key, obtaining the shared symmetric key for further communication.

Key Strengths and Advantages of RSA:

1. **Secure Key Exchange:** RSA enables secure key exchange without requiring a pre-shared secret key. The recipient's public key can be freely distributed, allowing others to encrypt messages or establish secure communication channels.
2. **Digital Signatures:** RSA supports the creation and verification of digital signatures, ensuring message authenticity and integrity. It allows recipients to verify the sender's identity and detect any tampering with the message.
3. **Key Distribution:** RSA eliminates the need for secure key distribution, as the public keys can be openly shared while the private keys remain confidential. This simplifies key management in large-scale systems.

Vulnerabilities and Weaknesses of RSA:

1. **Key Size:** The security of RSA depends on the size of the key. As computational power increases, longer key lengths are required to resist brute force attacks. Smaller key sizes are vulnerable to factoring attacks, where an attacker attempts to find the prime factors of the modulus.

2. **Implementation Flaws:** Weaknesses can arise from incorrect implementation or flawed random number generation. These vulnerabilities can undermine the security of RSA if not properly addressed.

Real-World Implementation of RSA:

RSA is widely adopted and used in various real-world applications. It is employed in secure email communication, SSL/TLS for secure web browsing, virtual private networks (VPNs), and digital certificate infrastructures, among others. Many cryptographic libraries and software applications provide RSA implementations, ensuring interoperability across different systems and platforms.

Hash Functions:

A hash function is a **mathematical function**, that typically takes an input and produces a fixed size string of characters, called the **hash code** or **hash value**.

An overview of how a hash function works:

1. **Input Data:** The hash function takes an input, which can be of any length or size. It can be a file, a password, a message, or any other form of data.
2. **Deterministic Output:** The hash function applies a deterministic algorithm to the input data, performing a series of mathematical operations such as modular arithmetic, bitwise operations, and logical functions. The algorithm ensures that the same input always produces the same output.
3. **Fixed Output Size:** The hash function produces a fixed-size output, regardless of the size or length of the input. For example, a common hash function like SHA-256 produces a 256-bit hash value.

An example of hash functions is the **SHA-256 (Secure Hash Algorithm)**, which is part of the SHA-2 family, published in 2001 by the NSA.

An overview of how SHA-256 works:

1. **Input Data:** The SHA-256 algorithm takes an input message of any length or size and processes it in fixed-size blocks.
2. **Padding:** The input message is padded to ensure it meets the required block size. The padding includes adding a '1' bit followed by zeros, and the length of the original message is appended in binary format.
3. **Message Digest Initialization:** The SHA-256 algorithm initializes a set of constants and an initial hash value, known as the "initialization vector" (IV), which serves as the starting

point for the computation.

4. **Compression Function:** The algorithm divides the padded message into fixed-size blocks and processes each block through a series of rounds. Each round involves various logical and arithmetic operations, such as bitwise operations, modular addition, and logical functions. The compression function updates the internal state of the hash algorithm.
5. **Final Hash Value:** After processing all the blocks, the final hash value is obtained. It represents a fixed-size output, typically 256 bits (32 bytes), and serves as the unique fingerprint or hash code of the input message.

Key Strengths and Advantages of SHA-256:

1. **Irreversibility:** Given a hash value, it is computationally infeasible to determine the original input message. The hash function works in only one direction, from input to hash value.
2. **Avalanche Effect:** Changing even a single bit in the input message will result in a significantly different hash value, ensuring that small modifications yield completely different hash codes.
3. **Collision Resistance:** A secure hash function like SHA-256 makes finding two different inputs that produce the same hash value computationally infeasible. Collisions, while theoretically possible, are exceedingly rare.

Vulnerabilities and Weaknesses of SHA-256:

1. **Advances in Computing Power:** As computational power increases, it becomes more feasible to perform brute-force attacks against hash functions. With sufficient computing resources, an attacker may be able to generate hash collisions or break the security of the algorithm.
2. **Length Extension Attacks:** SHA-256, like many other hash functions, is susceptible to length extension attacks. In such attacks, an adversary with knowledge of a hash value and its corresponding input can append additional data to the original input without knowing the actual input, generating a new valid hash value.
3. **Quantum Computing:** The advent of quantum computing poses a potential threat to many classical cryptographic algorithms, including SHA-256. Quantum computers have the potential to solve certain mathematical problems significantly faster than traditional computers, potentially compromising the security of hash functions and other cryptographic algorithms.

Real-World Implementation of SHA-256:

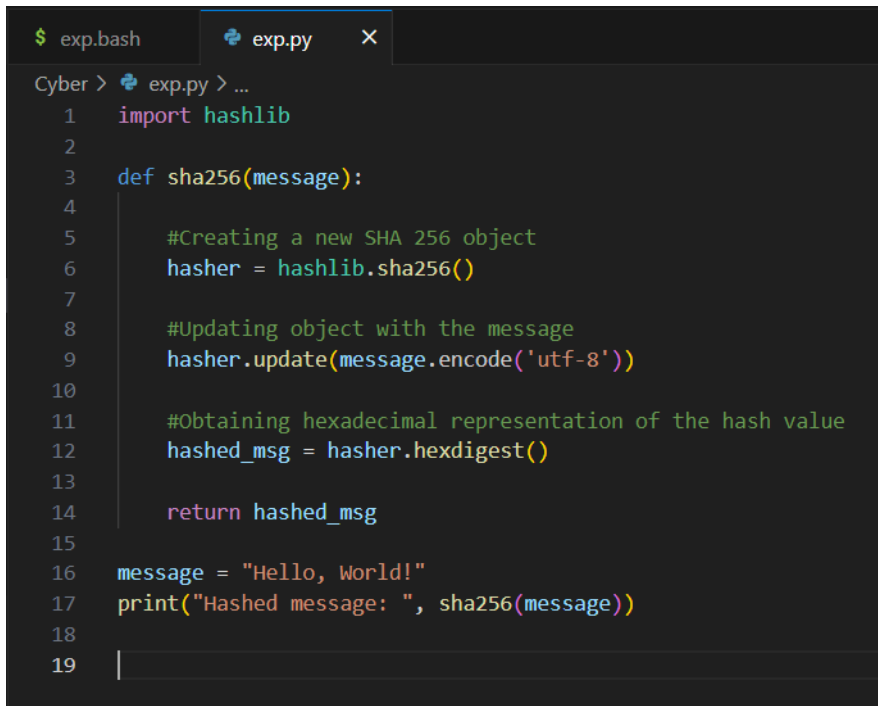
1. **Data Integrity:** SHA-256 is commonly used to ensure data integrity by verifying that the received data matches its corresponding hash value.
2. **Password Storage:** SHA-256, along with salt (a random value), is often used to securely store passwords. The hash value of a password, combined with a unique salt value, is stored in a database. When a user enters their password during authentication, the entered password is hashed with the stored salt, and the resulting hash value is compared with the stored hash value.
3. **Digital Certificates:** SHA-256 is used in digital certificates to ensure the authenticity and integrity of the certificate. The hash value of the certificate is signed with a private key,

creating a digital signature.

Implementation:

Here is the implementation of the **SHA-256** hash function using python. The python library, hashlib has been used to import hashing functions.

Code:

A screenshot of a code editor with a dark theme. The editor has two tabs at the top: 'exp.bash' and 'exp.py'. The 'exp.py' tab is active. The code is written in Python and implements a sha256 function. It imports the hashlib library, defines a function sha256(message) that creates a SHA-256 hasher, updates it with the message, and returns the hexadecimal digest. At the bottom, it sets a message to 'Hello, World!' and prints the hashed message.

```
$ exp.bash
exp.py x
Cyber > exp.py > ...
1  import hashlib
2
3  def sha256(message):
4
5      #Creating a new SHA 256 object
6      hasher = hashlib.sha256()
7
8      #Updating object with the message
9      hasher.update(message.encode('utf-8'))
10
11     #Obtaining hexadecimal representation of the hash value
12     hashed_msg = hasher.hexdigest()
13
14     return hashed_msg
15
16 message = "Hello, World!"
17 print("Hashed message: ", sha256(message))
18
19 |
```

This code performs a simple yet powerful purpose that is hashing. The properties of hashing make it a very versatile and irreplaceable tool in Network Security.

This implementation was very simple due to the usage of an already established python library called hashlib.

The output of the code would be:

```
Hashed message:  dffd6021bb2bd5b0af676290809ec3a53191dd81c7f70a4b28688a362182986f
```

Security Analysis:

Some potential attack vectors are:

1. **Collision Attacks:** Although SHA-256 is designed to be collision-resistant, meaning it should be extremely difficult to find two different inputs that produce the same hash

value, the possibility of collision attacks cannot be completely ruled out. As computational power advances, attackers may discover new techniques or algorithms that could potentially weaken the collision resistance of SHA-256.

2. **Implementation and Side-Channel Attacks:** Vulnerabilities can arise in the implementation of SHA-256, such as software bugs or flaws in hardware implementations. Additionally, side-channel attacks, which exploit information leaked during the execution of the algorithm (e.g., timing information or power consumption), can potentially reveal information about the hash function or its inputs.

Countermeasures to aforementioned vectors:

1. **Collision Attacks:**
 - Regularly update and use the latest version of the SHA-256 algorithm, as improvements and updates may address any potential vulnerabilities.
 - Consider using longer hash functions, such as SHA-3, which provide a larger hash size and enhanced security against collision attacks.
2. **Implementation and Side-Channel attacks:**
 - Employ side-channel attack mitigation techniques, such as constant-time implementations, to prevent leakage of sensitive information through timing or power consumption variations.
 - Regularly conduct security audits, code reviews, and penetration testing to identify and address implementation vulnerabilities.