

Assignment 3: Cryptography Analysis and Implementation

Name: D.Pramod

Collage: Vignan Lara Institute Of Technology And Science

Regs_No: 20FE1A0326

Objective: The objective of this assignment is to analyze cryptographic algorithms and implement them in a practical scenario.

Instructions:

Research: Begin by conducting research on different cryptographic algorithms such as symmetric key algorithms (e.g., AES, DES), asymmetric key algorithms (e.g., RSA, Elliptic Curve Cryptography), and hash functions (e.g., MD5, SHA-256). Understand their properties, strengths, weaknesses, and common use cases.

Analysis: Choose three cryptographic algorithms (one symmetric, one asymmetric, and one hash function) and write a detailed analysis of each. Include the following points in your

Implementation:

Select one of the cryptographic algorithms you analyzed and implement it in a practical scenario. You can choose any suitable programming language for the implementation.

Clearly define the scenario or problem you aim to solve using cryptography.

Provide step-by-step instructions on how you implemented the chosen algorithm.

Include code snippets and explanations to demonstrate the implementation.

Test the implementation and discuss the results.

Security Analysis:

Perform a security analysis of your implementation, considering potential attack vectors and countermeasures.

Identify potential threats or vulnerabilities that could be exploited.

Propose countermeasures or best practices to enhance the security of your implementation.

Discuss any limitations or trade-offs you encountered during the implementation process.

Conclusion: Summarize your findings and provide insights into the importance of cryptography in cybersecurity and ethical hacking.

Submission Guidelines:

Prepare a well-structured report that includes the analysis, implementation steps, code snippets, and security analysis.

Use clear and concise language, providing explanations where necessary.

Include any references or sources used for research and analysis.

Compile all the required files (report, code snippets, etc.) into a single zip file for submission.

MY ASSIGNMENT WORK

Symmetric cryptography:

Symmetric-key cryptography involves encrypting and decrypting using the same cryptographic keys. Here, the sender and all receivers share a common secret key. The plaintext messages are transformed into cipher text using a particular encryption key. The receiver can use the same encryption key to decrypt the message using the shared secret key.

Ex:

Advanced Encryption Standard (AES)

Data Encryption Standard (DES)

Triple Data Encryption Standard (Triple DES)

International Data Encryption Algorithm (IDEA)

TLS/SSL protocol

Triple DES:

The Triple Data Encryption Standard (DES) provides more security than the standard DES algorithm by using three passes of the DES rather than one. The security of standard DES was found to be less secure than AES. It uses 56-bit length keys.

Triple DES was developed as a way to prevent man in the middle attacks.

Once the weaknesses of normal DES became more apparent, 3DES was adopted in a wide range of applications. It was one of the more commonly used encryption schemes before the rise of AES.

3DES will be deprecated in the next few years, it is best to use other algorithms. While keying option one is still considered secure for many applications, there aren't many good reasons for why it should be used instead of an alternative like AES.

Strength and advantages:

3DES is a strengthening of DES introduced in 1998, because 56 bit keys had become feasible to brute force. 3DES is simply three DES encryptions with two different keys, for an effective 112 bit key; or with three different keys, for an effective 168 bit key. 3DES is an encryption algorithm widely used in the finance industry.

Ease of implementation in hardware and software and its widespread support among cryptographic libraries and protocols.

weakness:

The short block size of 64 bits makes 3DES vulnerable to block collision attacks if it is used to encrypt large amounts of data with the same key. The Sweet32 attack shows how this can be exploited in TLS and OpenVPN.

Practical Sweet32 attack on 3DES-based cipher-suites in TLS required $2^{36.6}$ blocks (785 GB) for a full attack.

OpenSSL does not include 3DES by default since version 1.1.0 (August 2016) and considers it a "weak cipher".

Some real-world examples of its implementations included:

Microsoft Office

Firefox

EMV payment systems

Asymmetric cryptography

It is also known as "public-key cryptography," uses different cryptographic keys for the encryption and decryption processes. The sender and the receiver have a private key and a public key.

The public key is shared with all the parties that must communicate with the sender.

The private key is kept secret from each party.

Though there is a mathematical connection between these private key and public key pairs, the public key cannot generate the private key.

Public key cryptography is commonly used in digital signatures for message authentication.

Senders use their private keys to digitally sign their messages to prove their authenticity. Thus, the receiver knows exactly that the sender is a trusted third party.

some examples:

Elliptical Curve Cryptography (ECC)

Rivest Shamir Adleman (RSA)

The Diffie-Hellman exchange method

TLS/SSL protocol

RSA algorithm (Rivest-Shamir-Adleman):

The basis of a cryptosystem -- a suite of cryptographic algorithms that are used for specific security services or purposes which enables public key encryption and is widely used to secure sensitive data, particularly when it is being sent over an insecure network such as the internet. RSA was first publicly described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology, though the 1973 creation of a public key algorithm by British mathematician Clifford Cocks was kept classified by the U.K.'s GCHQ until 1997. RSA is a type of asymmetric encryption, which uses two different but linked keys. In RSA cryptography, both the public and the private keys can encrypt a message. The opposite key from the one used to encrypt a message is used to decrypt it.

how does it work?

Alice generates her RSA keys by selecting two primes: $p=11$ and $q=13$. The modulus is $n=p \times q=143$. The totient is $n \phi(n)=(p-1) \times (q-1)=120$. She chooses 7 for her RSA public key e and calculates her RSA private key using the Extended Euclidean algorithm, which gives her 103. Bob wants to send Alice an encrypted message, M , so he obtains her RSA public key (n, e) which, in this example, is $(143, 7)$. His plaintext message is just the number 9 and is encrypted into ciphertext, C , as follows:

$$M^e \bmod n = 9^7 \bmod 143 = 48 = C$$

When Alice receives Bob's message, she decrypts it by using her RSA private key (d, n) as follows:

$$C^d \bmod n = 48^{103} \bmod 143 = 9 = M \text{ key}$$

strength and advantages:

Security based on the computational difficulty of factoring large composite numbers.

Public-key cryptography enables secure communication without a shared secret key.

Widely adopted and standardized, with applications in key exchange and digital signatures.

Strong security:

RSA encryption provides strong security based on the difficulty of factoring large composite numbers, making it difficult for unauthorized parties to decrypt encrypted data.

Public-key infrastructure:

RSA encryption utilizes a public-key system, allowing for secure communication and key exchange without the need for a shared secret key.

Versatile applications:

RSA encryption is widely adopted and standardized, making it compatible with various systems and enabling its use in digital signatures, secure data transmission, and other cryptographic protocols.

Weakness:

Key size requirements:

RSA encryption requires larger key sizes compared to symmetric encryption algorithms for equivalent security levels. As the desired level of security increases, the key size grows exponentially, which can impact computational and storage resources.

Vulnerability to quantum computers: RSA encryption is vulnerable to attacks by quantum computers, which have the potential to factor large numbers significantly faster compared to classical computers. As quantum computing advances, RSA encryption may become less secure if quantum-resistant alternatives are not used.

known vulnerability:

One of the known vulnerabilities of RSA encryption is its susceptibility to attacks based on the factorization of large composite numbers. If an attacker can successfully factorize the modulus used in RSA, they can potentially obtain the private key and decrypt encrypted data. The security of RSA relies on the assumption that factoring large numbers is a computationally difficult problem. However, advancements in computing power and factoring algorithms, such as the General Number Field Sieve (GNFS), can pose a threat to the security of RSA encryption if the key size used is not sufficiently large.

Real world examples:

Secure online transactions: RSA encryption is used in e-commerce platforms, online banking systems, and payment gateways to secure sensitive information, such as credit card details, during online transactions.

Secure messaging and chat applications: Many messaging and chat applications, including popular ones like WhatsApp and Signal, employ RSA encryption to protect the confidentiality of messages and ensure secure communication between users.

Virtual private networks (VPNs): RSA encryption is utilized in VPNs to establish secure connections between remote users and private networks, safeguarding data transmission and ensuring privacy.
and in digital certificates.

Hash functions:

Hash functions compute a fixed-length hash value or a “fingerprint” on the plain text message. These hashes are unique to each plaintext. Therefore, this type of cryptography does not use a cryptographic key. Hash functions help ensure data integrity between communicating parties. If

the hash produces the same output, it indicates that the information has not been altered, compromised or damaged.

Hash functions are used in many cryptographic algorithms and protocols, including MAC algorithms, digital signature algorithms, and authentication protocols.

Some of the most common hashing algorithms include:

SHA-1

SHA-2

SHA-3

MD5

Whirlpool

Blake 2

Blake 3

Sha-256

SHA-256 (Secure Hash Algorithm 256-bit):

SHA-256 is part of the SHA-2 family of hash functions. It generates a 256-bit hash value, providing a high level of security. It is widely adopted and used in various applications, including blockchain technology, digital signatures, and password storage.

Briefly explain how the algorithm works?

Message Padding: The input message is padded to ensure its length is a multiple of 512 bits, as required by the algorithm.

Initialization: A set of initial constants (known as "hash constants" or "round constants") and an initial hash value (known as "initial state") are defined.

Message Processing: The padded message is divided into blocks of 512 bits. The algorithm then processes each block sequentially.

Compression Function: Each block is further divided into 32-bit words, and a series of operations (such as bitwise logical functions, modular addition, and logical shifts) are performed on these words to iteratively update the hash value.

Rounds: The compression function consists of multiple rounds (64 rounds for SHA-256) that involve different operations and a unique set of constants for each round. These operations ensure a high degree of diffusion and confusion, making the hash function resistant to cryptographic attacks.

Final Hash Value: After processing all blocks, the final hash value is obtained by concatenating the updated hash values from each round.

The resulting 256-bit hash value is unique to the input message. Even a small change in the input message will produce a significantly different hash value. This property makes SHA-256

suitable for various applications, such as verifying data integrity, generating digital signatures, and securely storing passwords.

Discuss the key strengths and advantages of the algorithm:

This hashing algorithm is a variant of the SHA2 hashing algorithm, recommended and approved by the National Institute of Standards and Technology (NIST). It generates a 256-bit hash value. Even if it's 30% slower than the previous algorithms, it's more complicated, thus, it's more secure.

Strong security: SHA-256 offers a high level of security and collision resistance. It is designed to be resistant to various cryptographic attacks, including pre-image attacks, second pre-image attacks, and collision attacks. The 256-bit output size provides a large search space, making it computationally infeasible to find two different inputs that produce the same hash value.

Widely adopted and standardized: SHA-256 is a widely adopted and standardized hash function. It is implemented in many cryptographic libraries and systems, making it compatible with a wide range of applications. Its popularity and extensive analysis by the cryptographic community contribute to its reputation and security.

Identify any known vulnerabilities or weaknesses:

Quantum vulnerability: One of the primary vulnerabilities of SHA-256 (and other traditional cryptographic hash functions) is its susceptibility to attacks by quantum computers. Quantum computers have the potential to break the underlying mathematical principles that provide security to SHA-256, rendering it insecure. As quantum computing technology advances, there is a need to transition to quantum-resistant hash functions.

Collision resistance limitations: While collision attacks on SHA-256 are computationally infeasible in practice, there is a theoretical possibility of finding collisions due to the birthday paradox. With a sufficiently large number of hash operations, the probability of finding a collision increases. This limitation highlights the importance of periodically reevaluating and potentially transitioning to stronger hash functions to maintain long-term security.

Provide real-world examples of where the algorithm is commonly used.

Cryptocurrency: SHA-256 is the underlying hash function used in Bitcoin and many other cryptocurrencies. It is used for mining, where miners perform computational work to find a hash value below a certain target, ensuring the integrity and security of the blockchain.

Password Hashing: SHA-256, along with other iterations of SHA-2, is often used for secure password hashing. When storing user passwords, SHA-256 helps protect sensitive information by producing a fixed-length hash representation that is difficult to reverse-engineer. Digital Signatures: SHA-256 is commonly used in digital signature algorithms, such as the

Digital Signature Algorithm (DSA) and the Elliptic Curve Digital Signature Algorithm (ECDSA). It helps ensure the authenticity and integrity of digital documents, messages, and software updates.

IMPLEMENTATION:

Scenario: We want to securely store user passwords in a database by hashing them using SHA256. This ensures that even if the database is compromised, the original passwords remain secure.

```
import hashlib
```

```
def hash_password(password):
```

```
# Create a SHA-256 hash object
```

```
sha256_hash = hashlib.sha256()
```

```
# Convert the password string to bytes and update the hash object
sha256_hash.update(password.encode('utf-8'))
```

```
# Get the hexadecimal representation of the hashed password
hashed_password = sha256_hash.hexdigest()
```

```
# Return the hashed password
return hashed_password
```

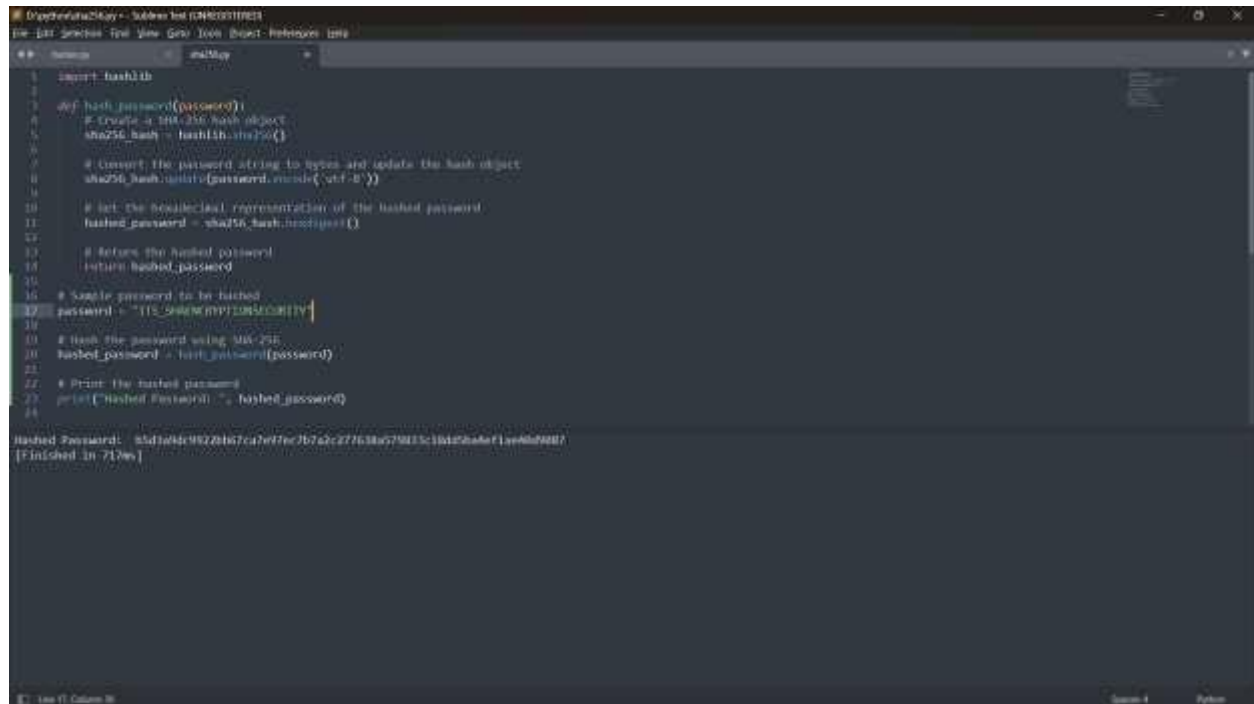
```
# Sample password to be hashed
password = "ITS_SHAENCRYPTIONSECURITY"
```

```
# Hash the password using SHA-256
hashed_password = hash_password(password)
```

```
# Print the hashed password
print("Hashed Password: ", hashed_password)
```


Results:

When you run the code, it will output the hashed password using the SHA-256 algorithm. Refer the below pic.



```
1 import hashlib
2
3 def hash_password(password):
4     # Create a SHA-256 hash object
5     sha256_hash = hashlib.sha256()
6
7     # Convert the password string to bytes and update the hash object
8     sha256_hash.update(password.encode('utf-8'))
9
10    # Get the hexadecimal representation of the hashed password
11    hashed_password = sha256_hash.hexdigest()
12
13    # Return the hashed password
14    return hashed_password
15
16 # Sample password to be hashed
17 password = "ITS@PACIFICCOMBATIV"
18
19 # Hash the password using SHA-256
20 hashed_password = hash_password(password)
21
22 # Print the hashed password
23 print("Hashed Password: ", hashed_password)
24
```

Hashed Password: sha256dc9822b67ca2e7f9e3b7a2c277638e79833c5b5d5a8e1a6b6d807
[Finished in 717ms]

Security analysis: potential

threat:

One potential threat to the SHA-256 algorithm is the advancement of quantum computers. Quantum computers have the potential to break the underlying mathematical principles on which the security of SHA-256 relies.

Currently, classical computers rely on computational complexity to make factoring large numbers infeasible within a reasonable time frame. However, quantum computers can potentially perform certain calculations, such as integer factorization, significantly faster due to their ability to leverage quantum phenomena like superposition and entanglement.

If sufficiently powerful quantum computers become available, they could potentially break the security of SHA-256 and other cryptographic algorithms based on similar principles. This would allow an attacker to efficiently compute collisions, pre-image attacks, and other vulnerabilities that could compromise the integrity and security provided by SHA-256.

counter measures and best practices:

Transition to post-quantum cryptography: Stay updated with advancements in post-quantum cryptography and migrate to quantum-resistant algorithms as they become standardized. Use

salting and iterations for password hashing: Employ techniques like salting and iterations to enhance password security and protect against precomputed hash tables and brute-force attacks. Ensure secure key management: Generate strong and random keys, securely store and protect them, follow recommended key rotation practices, and utilize key derivation functions for secure key generation from passwords or shared secrets.

what are its limitations:

Vulnerability to Quantum Computing: The SHA-256 algorithm is vulnerable to attacks by quantum computers, which have the potential to break the underlying mathematical principles that provide its security. As quantum computing technology advances, the security of SHA-256 and other similar algorithms may be compromised, necessitating the adoption of quantumresistant cryptographic algorithms.

Deterministic Output: SHA-256 always produces the same hash output for the same input. While this property is desirable for data integrity verification, it can limit certain use cases where nondeterministic behavior is required, such as generating unique identifiers or random values.

Fixed Input and Output Sizes: SHA-256 has a fixed input block size of 512 bits and a fixed output hash size of 256 bits. This can be limiting when working with larger data sets or when compatibility with systems using different hash sizes is required. It may require additional steps, such as chunking or truncation, to accommodate larger inputs or different output requirements.

Conclusion:

In conclusion, SHA-256 is a widely adopted and secure cryptographic hash function that offers strong security and collision resistance for most practical applications. It provides a reliable means of verifying data integrity, generating digital signatures, and securely storing hashed passwords.

However, it is important to acknowledge that SHA-256 has its limitations. It is vulnerable to potential attacks by quantum computers, and its deterministic nature may not be suitable for scenarios requiring non-deterministic behavior. Additionally, its fixed input and output sizes may present challenges when working with larger data sets or requiring compatibility with different hash sizes.

Despite these limitations, SHA-256 remains a robust and widely used algorithm. It is crucial to stay updated on advancements in cryptographic research and be prepared to transition to postquantum cryptographic algorithms when necessary. By following best practices, such as proper key management, secure implementation, and regular security audits, the use of SHA-256 can provide strong security for data integrity and other cryptographic applications.

