CYBER SECURITY AND ETHICAL HACKING
ASSIGNMENT – 3

ANNGELA ROY, 20MIS0186

1. 3DES (TRIPLE DATA ENCRYPTION STANDARD):

3DES is a symmetric encryption algorithm that applies the Data Encryption Standard (DES) cipher three times in succession. It uses a block cipher with a block size of 64 bits and a key size of 168 bits (using three 56-bit keys). The algorithm works by encrypting the plaintext with the first key, decrypting the result with the second key, and finally encrypting it again with the third key.

Key strengths and advantages of 3DES:

One of the key strengths of 3DES is its security. Despite its age, 3DES is still considered secure against most attacks. The multiple encryption rounds provide increased resistance against brute-force attacks and known-plaintext attacks. Furthermore, 3DES is widely compatible with cryptographic libraries and hardware, making it compatible with a wide range of systems. This compatibility makes it an attractive choice for legacy systems that have not yet migrated to more modern algorithms. Additionally, 3DES has a long history of use and analysis, which provides confidence in its security and reliability.

Known vulnerabilities or weaknesses:

While 3DES is generally secure, its main weakness is its relatively short key length of 168 bits. With the advancement in computational power, a brute-force attack on 3DES has become more feasible. Additionally, the algorithm's efficiency is lower compared to newer encryption algorithms. The multiple encryption and decryption rounds in 3DES can be computationally expensive and may not be suitable for resource-constrained devices.

Real-world examples of 3DES usage:

3DES has been widely used in various applications. For example, it has been extensively employed in financial systems for securing ATM transactions and payment processing. Additionally, 3DES has been used in secure communications protocols such as SSL/TLS, ensuring the confidentiality and integrity of data transmitted over the internet. It has also been employed in legacy systems for data encryption and protection.

Overall, while 3DES has been widely used in the past, it is gradually being phased out in favor of more modern symmetric encryption algorithms such as AES, which offer better security and efficiency.

2. ECC (Elliptic Curve Cryptography):

ECC is an asymmetric encryption algorithm based on the mathematics of elliptic curves. It utilizes the algebraic structure of elliptic curves over finite fields for key generation, encryption, and digital signatures. ECC provides the same level of security as traditional algorithms (such as RSA) but with shorter key lengths, making it more computationally efficient.

Key strengths and advantages of ECC:

One of the main advantages of ECC is its efficiency. ECC offers equivalent or stronger security than traditional algorithms (e.g., RSA) with shorter key lengths. This results in faster computations and lower resource usage, making it well-suited for resource-constrained devices like mobile phones and IoT devices. ECC also provides scalability, as it can handle larger key sizes efficiently. Additionally, ECC is considered resistant to attacks from both classical and quantum computers, making it a good choice for long-term security.

Known vulnerabilities or weaknesses:

ECC's vulnerabilities are mainly related to poor implementation or weak parameters. If the elliptic curve parameters are not chosen properly or if an implementation has flaws, it could weaken the algorithm's security. However, when implemented correctly with appropriate parameters, ECC is considered secure.

Real-world examples of ECC usage:

ECC is widely used in various applications. It is employed in secure communications protocols such as HTTPS and VPNs, providing confidentiality and integrity for data transmitted over networks. ECC is also used for digital signatures in protocols like DNSSEC and PGP, ensuring data authenticity and non-repudiation. Additionally, ECC is used in mobile communication protocols like 4G LTE and 5G for secure and efficient encryption.

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

SHA-256 is a cryptographic hash function that takes an input message and produces a fixed-size 256-bit hash value. It belongs to the SHA-2 family of hash functions and operates by iterating a compression function on blocks of the message data.

Key strengths and advantages of SHA-256:

SHA-256 offers several advantages. One of its main strengths is collision resistance. SHA-256 is designed to have a high probability of producing unique hash values for different inputs, making it difficult to find two different inputs that produce the same hash. This property ensures the integrity of data, as even a slight change in the input will result in a drastically different hash value. SHA-256 is deterministic, meaning that given the same input, it will always produce the same hash output. This property is useful for verifying data integrity or detecting changes in stored files. Additionally, SHA-256 is computationally efficient and can quickly generate hash values for large amounts of data.

Known vulnerabilities or weaknesses:

As of now, no practical vulnerabilities have been discovered for SHA-256. However, advancements in technology and cryptanalysis techniques always pose a potential risk. While no vulnerabilities have

been found in SHA-256 itself, its security depends on the secure storage and transmission of hash values, as an attacker could potentially perform a collision attack if they can manipulate the hash values.

Real-world examples of SHA-256 usage:

SHA-256 is widely used in various applications. It is commonly used in blockchain technology, such as in cryptocurrencies like Bitcoin. In this context, SHA-256 ensures the integrity and immutability of transaction data. SHA-256 is also utilized in digital certificates for SSL/TLS, providing secure and trusted communication over the internet. Furthermore, SHA-256 is commonly used for password hashing, securely storing passwords by generating hash values that are computationally expensive to reverse.

IMPLEMENTATION OF SHA256 ALGORITHM

Code:

main.py

```python
import hashlib

def compute_sha256_hash(input_data):

    sha256_hash = hashlib.sha256()

    if isinstance(input_data, str):
        input_data = input_data.encode('utf-8')

    sha256_hash.update(input_data)

    hash_value = sha256_hash.hexdigest()

    return hash_value

input_text = input("Enter the text for which you want to compute the SHA-256 hash: ")


sha256_hash = compute_sha256_hash(input_text)
print("SHA-256 hash:", sha256_hash)

```

Output:

```
Enter the text for which you want to compute the SHA-256 hash: Anngela
SHA-256 hash: ac3b6fb5c96d907d46d603bc3032f33e3650ba3f4b0d3571e6c514ef434c113a


...Program finished with exit code 0
Press ENTER to exit console.
```

Security Analysis:

1. Potential Threats or Vulnerabilities:

- Input Validation: The implementation assumes that the input data is provided by the user and doesn't perform thorough input validation. This can lead to potential vulnerabilities such as code injection or other malicious input attacks.

- Side-Channel Attacks: The implementation doesn't address potential side-channel attacks like timing attacks or power analysis attacks. These attacks exploit information leaked during the computation of the hash, such as timing differences or power consumption variations.

2. Countermeasures and Best Practices:

- Input Sanitization: Implement input validation and sanitization techniques to ensure that the input data is properly validated and doesn't contain any malicious content. This helps prevent code injection and other input-based attacks.

- Constant-Time Implementation: Use constant-time implementations of cryptographic operations to mitigate side-channel attacks. This involves avoiding operations that could leak information, such as branching based on secret values or performing variable-time computations.

3. Limitations and Trade-Offs:

- The provided code focuses on computing the SHA-256 hash and doesn't address other aspects of cryptographic security, such as key management or data integrity verification. These additional considerations are important in practical applications.

- The code assumes UTF-8 encoding for converting the input data to bytes. In scenarios where a different encoding is used, appropriate modifications should be made to ensure correct hashing.

- The implementation is based on the hashlib module provided by Python's standard library, which is generally considered reliable and secure. However, it's important to keep the library and its dependencies up to date to address any vulnerabilities or patches.

Conclusion:

Cryptography plays a critical role in cybersecurity and ethical hacking by providing mechanisms to secure sensitive information and protect systems against unauthorized access. The implementation of cryptographic algorithms, such as the SHA-256 hash function, is an essential component of secure systems.

However, it's crucial to ensure that the implementation follows best practices and addresses potential vulnerabilities. Input validation and sanitization are vital to prevent attacks like code injection, while constant-time implementations help mitigate side-channel attacks.

While the provided code serves as a starting point, it should be part of a more comprehensive security framework that includes secure key management, secure transmission channels, and proper handling of cryptographic material. Regular updates and adherence to best practices will enhance the overall security of the implementation.