ASSIGNMENT – 3

NAME: Pranshu Sangwan
REG.NO : 20MIC0146

AES ANALYSIS

1. *The symmetric encryption method known as AES (Advanced Encryption Standard) works on blocks of data. Here is a breakdown of the AES algorithm, including its main benefits and advantages, known flaws, and examples of how it is typically used in real-world settings:*

Summary of the algorithm:
- AES works with blocks of data, each of which has a set size of 128 bits (16 bytes).

- It works with keys that are 128, 192, and 256 bits long.
Depending on the key size, AES uses 10, 12, or 14 rounds of substitution, permutation, and mixing operations.
To achieve its security features, it combines substitution, diffusion, and confusion approaches.
- Both encryption and decryption use the same key.

2. Key Benefits and Strengths:
- Security: AES is widely used by businesses, organisations, and governments around the world and is recognised as a secure encryption algorithm.
- Efficiency: AES works well on a variety of platforms, including PCs, mobile devices, and embedded systems, and is efficient.
- Versatility: AES allows a range of key sizes, allowing it to be adapted to specific security needs. - Standardisation: AES is a recognised standard that guarantees compatibility with many operating systems and programming languages.
- Broad Support: AES is widely supported by a number of cryptographic frameworks and libraries, making it simple to use in a variety of applications.

3. Recognised Weaknesses or Vulnerabilities:
- Timing and Side-Channel Attacks: In some circumstances, attackers can take advantage of timing inconsistencies or side-channel data (such power usage or electromagnetic radiation) to deduce details about the encryption key.
- Key Management: Key management is not handled by AES itself. The right key generation, storage, and distribution processes are essential for the encryption system's security.
AES is regarded as secure against attacks on conventional computer systems. Future developments in quantum computing could, however, make it less secure. Algorithms for post-quantum encryption are being explored as a possible successor.

4. Real-World Examples: - Secure Communication: AES is frequently used to encrypt data sent between clients and servers in secure communication protocols like TLS (Transport Layer Security) and VPNs.
- File and Disc Encryption: To protect sensitive data saved on storage devices, AES is used in a variety of file and disc encryption software.
- Database Encryption: To prevent unauthorised access, sensitive data stored in databases is encrypted using AES.
- Wireless Security: The WPA2 (Wi-Fi Protected Access 2) protocol, which enables secure wireless connection, has AES as one of its components.

RSA ANALYSIS

RSA (Rivest-Shamir-Adleman) is a widely used asymmetric encryption algorithm. Here's an analysis of the RSA algorithm, its key strengths and advantages, known vulnerabilities or weaknesses, and real-world examples of its common usage:

1. Overview of the Algorithm: The mathematical challenge of factoring huge composite numbers into their prime factors is the foundation of the RSA algorithm.
- A public-private key pair is used, with the public key being used for encryption and the private key for decryption.
- The private key is composed of the prime factors of the product of two large prime numbers, from which the public key is obtained.
- The public key is used to calculate the modulus of the result after raising the plaintext message to the power of the public exponent.
- RSA decryption entails taking the modulus with the private key and increasing the ciphertext to the power of the private exponent.

2. Key Benefits and Strengths:
- Security: The security of RSA is built on the fact that factoring huge numbers is difficult. The size of the key affects how secure a system is.
Asymmetric encryption is supported by RSA without the need for a shared secret key.It makes data shared between parties genuine and confidential.

- For digital signatures, which guarantee the validity and integrity of digital documents, RSA can be employed.
- Key Exchange: In situations like setting up secure communication channels or secure remote logins, RSA can assist secure key exchange.
- Standardisation: RSA is extensively used and standardised, ensuring compatibility with many applications and systems.

3. Recognised Weaknesses or Vulnerabilities:
- Key Size: The key's size determines how secure RSA is. Smaller key sizes may be susceptible to assaults like factoring or brute-force.

- Problems with implementation: Weak random number generation or side-channel attacks may result from improper RSA implementation or insufficient key generation techniques. - Timing and Side-Channel Attacks: RSA implementations are susceptible to side-channel and timing attacks that take advantage of data that is exposed during the encryption or decryption process. - Quantum Computers: Because quantum computers are capable to effectively factoring big numbers, they can attack RSA. As alternatives, post-quantum encryption techniques are being created.

4. Real-World Illustrations
- Secure Communication: To create secure connections between clients and servers, RSA is frequently used in secure communication protocols like SSL/TLS.
- Public Key Infrastructure (PKI): For certificate-based authentication, digital signatures, and secure email transmission, PKI systems use RSA.
- Secure File Transfer: RSA is used for encryption and authentication in secure file transfer protocols as SFTP (Secure File Transfer Protocol) and PGP (Pretty Good Privacy).
- Secure Shell (SSH): Secure remote logins, encrypted remote command execution, and secure file transfers are all made possible through SSH.

MD5 ANALYSIS

MD5 (Message Digest Algorithm 5) is a widely used cryptographic hash function. Here's an analysis of the MD5 algorithm, its key strengths and advantages, known vulnerabilities or weaknesses, and real-world examples of its common usage:

- *1. Algorithm Overview: MD5 generates a fixed-size, 128-bit hash value from an input message of any length. - It processes the input message using a number of logical operations, such as bitwise operations, modular arithmetic, and nonlinear functions.*

- *- Because the generated hash value is specific to the input message, even minor changes to the input will result in noticeably different hash values.*

- *2. Key Benefits and Strengths:*

- *Quickness and Effectiveness: Because MD5 is rather quick and effective, it can be used in systems that need real-time hashing.*

- *Usability: MD5 is simple to implement and use and uses only a little amount of processing power.*

- Checksum Validation: The integrity of files or data can be confirmed using the MD5 algorithm. It may instantly verify whether a file has been altered by comparing the calculated MD5 hash of the file with a previously created hash.
3. Known Vulnerabilities or Weaknesses: - Collision Vulnerabilities: When two different inputs result in the same hash value, MD5 is thought to be weak against collision attacks. It is inadequate for cryptographic security due to this flaw.

- Security Vulnerabilities: A number of security flaws, such as pre-image attacks and the capacity to produce hash collisions, have been found in the method.
- Deprecated Usage: Because of its flaws, MD5 is no longer advised for use in applications that require security or for cryptographic operations.

4. Real-World Examples:
- Data Integrity Checking: The integrity of downloaded files can be confirmed using the MD5 algorithm. To make sure the file wasn't altered during transmission, the MD5 hash of the downloaded file is compared with the known hash value.
- Password Storage (Less Secure Usage): In certain legacy systems, password hashes have been stored using the MD5 algorithm. However, because to the openness to hash cracking and preimage attacks, this practise is no longer regarded as secure.

AES IMPLEMENTATION USING PYTHON

```python
import os
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Hash import SHA256

def encrypt(key, filename):
        chunksize = 64*1024
        outputFile = "(enc)"+filename
        filesize = str(os.path.getsize(filename)).zfill(16)
        IV = Random.new().read(16)

        encryptor = AES.new(key, AES.MODE_CBC, IV)

        with open(filename, 'rb') as infile:#rb means read in binary
                with open(outputFile, 'wb') as outfile:#wb means write in the binary mode
                        outfile.write(filesize.encode('utf-8'))
                        outfile.write(IV)

                        while True:
                                chunk = infile.read(chunksize)

                                if len(chunk) == 0:
                                        break
                                elif len(chunk)%16 != 0:
                                        chunk += b' '*(16-(len(chunk)%16))

                                outfile.write(encryptor.encrypt(chunk))
```

```
def decrypt(key, filename):
        chunksize = 64*1024
        outputFile = filename[11:]

        with open(filename, 'rb') as infile:
                filesize = int(infile.read(16))
                IV = infile.read(16)

                decryptor= AES.new(key, AES.MODE_CBC, IV)

                with open(outputFile, 'wb') as outfile:
                        while True:
                                chunk = infile.read(chunksize)

                                if len(chunk) == 0:
                                        break

                                outfile.write(decryptor.decrypt(chunk))

                        outfile.truncate(filesize)

def getKey(password):
        hasher = SHA256.new(password.encode('utf-8'))
        return hasher.digest()
```

```
def Main():
        choice = input("Would you like to (E)encrypt or (D)Decrypt ")

        if choice == 'E':
                filename = input("File to encrypt: ")
                password = input("Password: ")
                encrypt(getKey(password), filename)
                print('Done.')
        elif choice == 'D':
                filename = input("File to decrypt: ")
                password = input("Password: ")
                decrypt(getKey(password),filename)
                print("Done.")

        else:
                print("No option selected, closing ... ")


Main()
```

```
└─$ python3 aescode.py
Would you like to (E)encrypt or (D)Decrypt
```

```
└─$ python3 aescode.py
Would you like to (E)encrypt or (D)Decrypt E
File to encrypt: bank_credentials.txt
Password: verysecretpassword11
Done.
```

```
└─$ cat bank_credentials.txt
0000000000000098◆G□Iw◆◆B◆◆W◆SZ◆2◆5W?◆Oo◆◆|◆◆Y(n◆ ◆▒◆v◆\0◆RQ◆◆◆:
                                                    ◆◆|V◆f'  d◆\.O◆◆◆_{◆_◆◆◆◆"t◆ħ◆qed◆`4◆'◆◆
.C◆v◆◆u◆◆◆◆P◆◆v◆◆Fu◆
```

```
└─$ python3 aescode.py
Would you like to (E)encrypt or (D)Decrypt D
File to decrypt: bank_credentials.txt
Password: verysecretpassword11
Done.
```

```
└─$ cat tials.txt
BANK NAME: BANK1234
USERNAME: USER1234
PASSWORD: PASS1234
SECURIT QUESTION: FAVORITE COLOR - BLUE
```

Security Analysis of the AES File Encryption and Decryption Implementation:

Potential threats or weaknesses:
Brute-force attacks: By repeatedly attempting all potential key combinations, an attacker could try to decipher the encryption key. Longer and more difficult passwords can lessen this danger.

b. Password-based Attacks: If the password that was used to create the key is flimsy or simple to guess, it may be susceptible to dictionary attacks or password cracking methods. Strong password policies must be followed.

c. Side-Channel Attacks: The implementation may be vulnerable to side-channel attacks, which take advantage of data that is revealed during the encryption or decryption process. Examples of such attacks include timing attacks and power analysis. These attacks can be lessened by employing defences like constant-time implementations or suitable hardware safeguards.

a. Malware or Tampering: If an attacker gains access to the system or the files during the encryption or decryption process, they can modify the files or inject malware. Employing strong access controls and regularly scanning systems for malware can help mitigate this threat.

2. Countermeasures and Best Practices:
a. Key Strength: Ensure the use of strong and unique passwords to generate the encryption key. Implement password complexity requirements and encourage users to use password managers.

b. Key Management: Store encryption keys securely, such as using a key management system or hardware security modules. Protect keys from unauthorized access and regularly rotate them.

c. Secure File Handling: Implement secure coding practices when handling files, such as validating input, sanitizing file names, and avoiding path traversal vulnerabilities.

d. Secure Communication: If transmitting encrypted files over a network, use secure communication protocols like TLS/SSL to protect against interception and tampering.

e. System Hardening: Regularly update the system with security patches, use firewalls, and employ intrusion detection and prevention systems to protect against attacks on the underlying system.

3. Constraints and Compromises:

Key management: The secure storage and distribution of encryption keys are not addressed by the implementation. In order to guarantee the system's overall security, proper key management procedures should be performed independently.

b. Algorithm Selection: The implementation employs the secure AES with CBC mode. Other modes, such as GCM (Galois/Counter Mode), offer more data integrity and authentication features, though.

c. User Input Validation: The implementation relies on reliable and accurate input from the user. To stop attacks like route traversal, SQL injection, or buffer overflows, input validation should be used.

Conclusion:

By offering methods to safeguard confidential information and guarantee data integrity, cryptography plays a significant part in cybersecurity and ethical hacking. To minimise potential dangers and weaknesses, it is crucial to correctly implement cryptographic algorithms and adhere to best practises. The analysed implementation offers a fundamental framework for AES-based file encryption and decryption, but extra security precautions like key management and input validation should be taken into account to increase the system's overall security. To ensure the reliability of cryptographic implementations and guard against changing attack vectors, regular security audits, updates, and adherence to secure coding practises are required.

.