

## Assessment-3

### Title: Cryptography Analysis and Implementation

---

*Name: B Venkata Mounish Reddy*

*Reg No: 20BDS0149*

*Campus: VIT Vellore*

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

#### **Introduction:**

Cryptographic algorithms play a crucial role in ensuring the security and integrity of digital communications. In this report, we will analyze three different types of cryptographic algorithms: one symmetric encryption algorithm, one asymmetric encryption algorithm, and one hash function. We will explore the working principles, strengths, weaknesses, and real-world applications of each algorithm.

#### **I. Symmetric Encryption Algorithm: Advanced Encryption Standard (AES)**

1. Explanation: The Advanced Encryption Standard (AES) is a widely used symmetric encryption algorithm. It operates on fixed-size blocks of data, typically 128 bits, and uses a secret key for encryption and decryption. AES performs a series of substitution, permutation, and mixing operations known as rounds to provide secure encryption.
2. Key Strengths and Advantages:
  - Strong Security: AES has been extensively analyzed by cryptographers worldwide and is considered secure against known attacks when implemented correctly.
  - Efficiency: AES is highly efficient and computationally fast, making it suitable for resource-constrained devices.
  - Flexibility: AES supports key sizes of 128, 192, and 256 bits, allowing users to choose the level of security needed.
  - Wide Adoption: AES is the standard encryption algorithm used by governments, financial institutions, and organizations worldwide.
3. Known Vulnerabilities or Weaknesses:
  - Side-Channel Attacks: AES is vulnerable to side-channel attacks, such as timing attacks or power analysis attacks, where an attacker exploits information leaked during the encryption process.

- Key Management: AES relies on a secure key distribution mechanism, and any compromise in key management can lead to the compromise of encrypted data.

#### 4. Real-World Examples:

- AES is commonly used in various applications, including:
  - Secure communication protocols (e.g., SSL/TLS) for secure web browsing.
  - Full disk encryption software (e.g., BitLocker, FileVault) to protect data on storage devices.
  - Wireless networks (e.g., Wi-Fi networks using WPA2) to encrypt network traffic.

## **II. Asymmetric Encryption Algorithm: RSA (Rivest-Shamir-Adleman)**

1. Explanation: RSA is an asymmetric encryption algorithm named after its inventors. It relies on the mathematical properties of large prime numbers and uses a public-private key pair for encryption and decryption. The public key is freely distributed, while the private key is kept secret.
2. Key Strengths and Advantages:
  - Strong Encryption: RSA provides secure encryption based on the difficulty of factoring large numbers into their prime factors.

- Digital Signatures: RSA supports digital signatures, allowing the verification of message integrity and authentication of the sender.
- Key Exchange: RSA enables secure key exchange between two parties, facilitating secure communication channels.
- Widely Accepted: RSA is widely supported and implemented in various cryptographic libraries and systems.

### 3. Known Vulnerabilities or Weaknesses:

- Key Size: RSA's security is directly related to the key size used. As computational power advances, longer key sizes are required to maintain security.
- Timing Attacks: Implementation vulnerabilities, such as timing differences during encryption or decryption, can be exploited to recover the private key.

### 4. Real-World Examples:

#### ● RSA is commonly used in the following scenarios:

- Secure email communication using protocols like OpenPGP or S/MIME.
- Digital signatures in document signing or certificate authorities.
- Key exchange in secure communication protocols like SSH or SSL/TLS.

### **III. Hash Function: SHA-256 (Secure Hash Algorithm 256-bit)**

1. Explanation: SHA-256 is a cryptographic hash function that takes an input message of any size and produces a fixed-size 256-bit hash value. It employs a series of logical operations, including bitwise operations and modular arithmetic, to compute the hash value. The resulting hash is unique to the input data, and even a slight change in the input will produce a significantly different hash.
2. Key Strengths and Advantages:
  - Collision Resistance: SHA-256 is designed to have a negligible probability of producing the same hash value for different inputs, providing strong collision resistance.
  - Deterministic Output: Given the same input, SHA-256 will always produce the same hash output, allowing for verification and integrity checks.
  - Fixed Size: SHA-256 generates a fixed-size hash value, regardless of the input size, making it suitable for various applications.
  - Wide Adoption: SHA-256 is widely adopted and supported, making it a standard choice for cryptographic hash functions.
3. Known Vulnerabilities or Weaknesses:

- Length Extension Attacks: SHA-256 is vulnerable to length extension attacks, where an attacker extends the hash value without knowing the original data. This vulnerability can be mitigated by using HMAC constructions or specialized variants like SHA-3.
- Quantum Computing: In the future, quantum computers could potentially break the security of SHA-256 and other commonly used cryptographic algorithms. Post-quantum hash functions are being researched as potential replacements.

#### 4. Real-World Examples:

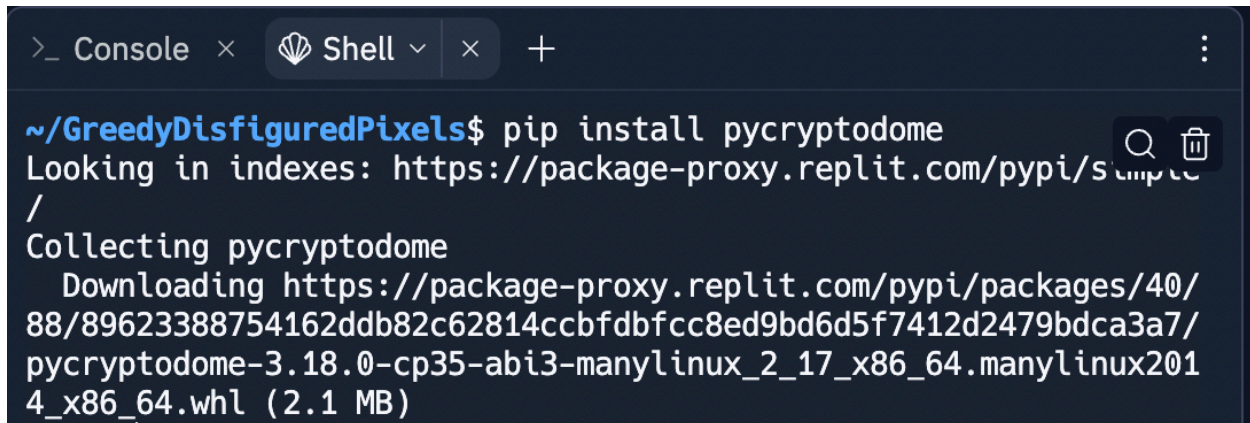
##### ● SHA-256 is commonly used in the following applications:

- Digital Signatures: Hash functions like SHA-256 are used in combination with asymmetric encryption algorithms to create digital signatures, providing integrity and authentication of digital documents.
- Password Storage: SHA-256, along with additional techniques like salting and stretching, is used to securely store user passwords in databases.
- Blockchain Technology: SHA-256 is extensively used in blockchain networks, such as Bitcoin, to ensure the integrity of transaction data and block validation.

## Implementation:

Implementation and Security Analysis of AES Encryption Algorithm using Python:

Step 1: Setting up the Environment

A screenshot of a terminal window with a dark background. The title bar shows a tab labeled 'Console' and a dropdown menu set to 'Shell'. The terminal text shows a user at a prompt installing pycryptodome. It shows the package being collected and downloaded from a Replit proxy, including the file name and size (2.1 MB).

```
>_ Console x Shell v x +  
~/GreedyDisfiguredPixels$ pip install pycryptodome  
Looking in indexes: https://package-proxy.replit.com/pypi/simple/  
/  
Collecting pycryptodome  
  Downloading https://package-proxy.replit.com/pypi/packages/40/  
88/89623388754162ddb82c62814ccbfdbfcc8ed9bd6d5f7412d2479bdca3a7/  
pycryptodome-3.18.0-cp35-abi3-manylinux_2_17_x86_64.manylinux201  
4_x86_64.whl (2.1 MB)
```

Step 2: Generating the AES Key

Step 3: Encrypt the data from a file using the AES algorithm.

Step 4: Decrypt the encrypted data using same key used during encryption.

Step 5: Encrypt a file and then decrypt the encrypted data to verify the correctness of our implementation.

Code snippets:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

# Generate a random 256-bit key
key = get_random_bytes(32)

# Generate a random 128-bit IV
iv = get_random_bytes(16)

def encrypt_file(file_path, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    encrypted_data = b''

    with open(file_path, 'rb') as file:
        while True:
            chunk = file.read(16) # Read 16 bytes at a time
            if len(chunk) == 0:
                break
            elif len(chunk) % 16 != 0:
                # Pad the chunk if its length is not a multiple of 16
                chunk += b' ' * (16 - (len(chunk) % 16))

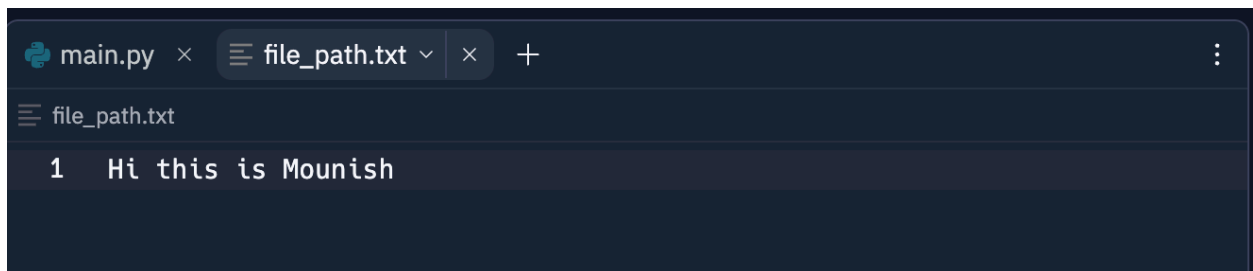
            encrypted_chunk = cipher.encrypt(chunk)
            encrypted_data += encrypted_chunk

    return encrypted_data
```



```
def decrypt_data(encrypted_data, key, iv):  
    cipher = AES.new(key, AES.MODE_CBC, iv)  
    decrypted_data = cipher.decrypt(encrypted_data)  
  
    return decrypted_data  
  
file_path = 'file_path.txt'  
  
# Encrypt the file  
encrypted_data = encrypt_file(file_path, key, iv)  
  
# Decrypt the encrypted data  
decrypted_data = decrypt_data(encrypted_data, key, iv)  
  
# Print the decrypted data  
print(decrypted_data.decode())
```

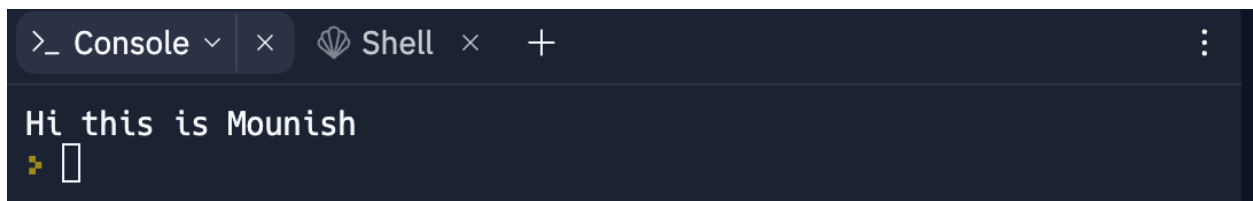
Text file:



The screenshot shows a text editor with two tabs: 'main.py' and 'file\_path.txt'. The 'file\_path.txt' tab is active, displaying the text 'Hi this is Mounish' on the first line.

Output:

◆ The encrypted file is successfully decrypted:



The screenshot shows a terminal window with two tabs: 'Console' and 'Shell'. The 'Console' tab is active, displaying the output 'Hi this is Mounish' followed by a prompt character '❖' and a cursor.

## Security Analysis:

### **Potential Threats or Vulnerabilities:**

1. **Key Management:** Protecting the secret key is crucial. Storing the key securely and using strong access controls are essential to prevent unauthorized access.
2. **Side-Channel Attacks:** AES can be vulnerable to side-channel attacks, such as timing or power analysis attacks. Implementing countermeasures like constant-time implementations can mitigate these risks.
3. **Weak IVs:** Using weak or predictable IVs can compromise the security of AES. Generating random and unique IVs for each encryption operation is necessary.

### **Countermeasures and Best Practices:**

1. **Key Protection:** Store the secret key securely, such as using hardware security modules (HSMs) or protected key vaults. Use strong access controls to restrict key access to only authorized personnel.
2. **IV Management:** Generate random and unique IVs for each encryption operation. Never reuse the same IV with the same key.
3. **Implement Countermeasures:** Implement countermeasures against side-channel attacks, such as constant-time implementations or using hardware with built-in protection against timing attacks.

## **Limitations and Trade-offs:**

1. **Performance:** AES encryption and decryption can be computationally expensive, especially for large files or high data volumes, introducing overhead and impacting resource-constrained environments.
2. **Key Management:** Securely managing and protecting encryption keys is essential, requiring careful design and implementation of key distribution, storage, and revocation processes.
3. **Compatibility:** Variations in AES implementations and support for different key sizes across systems and platforms can pose challenges in ensuring compatibility.

## **Conclusion:**

Cryptography is crucial for securing digital communications and protecting sensitive data. AES is a widely adopted symmetric encryption algorithm, providing strong security, efficiency, and flexibility. Its real-world applications include secure communication protocols, disk encryption, and wireless networks.