

## **Assignment: Cryptography Analysis and Implementation**

*NAME: PRANAV DATT*

*Reg No: 20BCE2722*

*CAMPUS: VIT - Vellore*

**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

### **analysis:**

Briefly explain how the algorithm works.

Discuss the key strengths and advantages of the algorithm.

Identify any known vulnerabilities or weaknesses.

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

### **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.

## ANALYSIS:

I have selected AES for symmetric encryption, RSA for asymmetric encryption and SHA hash function for my analysis.

### 1. Advanced Encryption Standard (AES):

AES is a symmetric encryption algorithm that operates on fixed-size blocks of data. It uses a substitution-permutation network (SPN) structure and performs several rounds of substitutions, permutations, and mix-column operations to provide encryption and decryption. AES supports key sizes of 128, 192, and 256 bits. AES is widely regarded as secure and efficient. Its strengths include resistance against known cryptographic attacks, good performance on various hardware platforms, and suitability for both software and hardware implementations. AES has been extensively analyzed, and no practical vulnerabilities have been discovered so far. However, attacks could potentially arise if new weaknesses are found in the underlying mathematics or if implementation flaws are present. AES is commonly used in various applications, such as securing sensitive data in communication protocols (e.g., TLS/SSL), disk encryption (e.g., BitLocker), and file encryption (e.g., ZIP archives).

### 2. RSA:

RSA is an asymmetric encryption algorithm based on the mathematical properties of prime factorization. It uses a pair of keys: a public key for encryption and a private key for decryption. The security of RSA relies on the difficulty of factoring large composite numbers into their prime factors. RSA provides secure key exchange, digital signatures, and encryption. Its strengths include robustness against known attacks when implemented correctly, support for key management and distribution in a public key infrastructure (PKI), and widespread compatibility with various systems and platforms. RSA's security relies on the difficulty of factoring large numbers, but advancements in computing power and factorization algorithms could potentially weaken its security. In recent years, vulnerabilities have been discovered in specific implementations or side-channel attacks targeting the implementation itself, rather than the mathematical foundation. RSA is widely used in various applications, such as secure email (e.g., OpenPGP), secure remote access (e.g., SSH), digital signatures (e.g., code signing certificates), and secure communication protocols (e.g., HTTPS).

### 3. SHA-256:

SHA-256 is a hash function that operates on input data of arbitrary size and produces a fixed-size hash value (256 bits). It uses a series of logical operations, bitwise operations, and modular arithmetic to process the input data and generate the hash output. SHA-256 provides a high level of collision resistance, making it suitable for applications such as digital signatures, password hashing, and data integrity verification. It is widely supported and has a faster implementation speed compared to some other hash functions. While SHA-256 is considered secure, some concerns have been raised regarding the theoretical possibility of collision attacks. As computational power advances, there is always a chance that new vulnerabilities may be discovered. Additionally, SHA-256 alone does not provide key confidentiality or privacy. SHA-256 is commonly used in various applications, including digital certificates (e.g., X.509 certificates), blockchain technology (e.g., Bitcoin), password storage (e.g., bcrypt with SHA-256), and data integrity checks (e.g., verifying file integrity with checksums).

### IMPLEMENTATION:

I choose to implement AES algorithm in Python to encrypt and decrypt a file. The scenario I aim to solve is securely storing and transmitting sensitive files by encrypting them with AES.

Firstly, we need to install the “pycryptodome” library, which provides AES encryption functionalities in Python.

```
pip install pycryptodome
```

Then we will import the packages such as AES, pad and unpad.

```
from Crypto.Cipher import AES  
from Crypto.Util.Padding import pad, unpad
```

Then we will create the functions for encrypting the given file and decrypt the given file using AES encryption.

```
def encrypt_file(key, input_file, output_file):
    cipher = AES.new(key, AES.MODE_CBC)
    with open(input_file, 'rb') as file:
        plaintext = file.read()
    ciphertext = cipher.encrypt(pad(plaintext, AES.block_size))
    iv = cipher.iv
    with open(output_file, 'wb') as file:
        file.write(iv + ciphertext)

def decrypt_file(key, input_file, output_file):
    with open(input_file, 'rb') as file:
        data = file.read()
    iv = data[:AES.block_size]
    ciphertext = data[AES.block_size:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)
    with open(output_file, 'wb') as file:
        file.write(plaintext)
```

In the encrypt file function, it will take 3 parameters key, input file, and output file. First it open the input file in binary mode and reads the data present in it. Then it encrypts it using the key and write the data in the output file.

In the decrypt file function, it will also take 3 inputs. It opens the encrypted file and reads the data. Then it decrypt the data using key and stores the data in the output file.

Now lets generate a random key and call the encrypt file function.

```
import secrets

encryption_key = secrets.token_bytes(16)

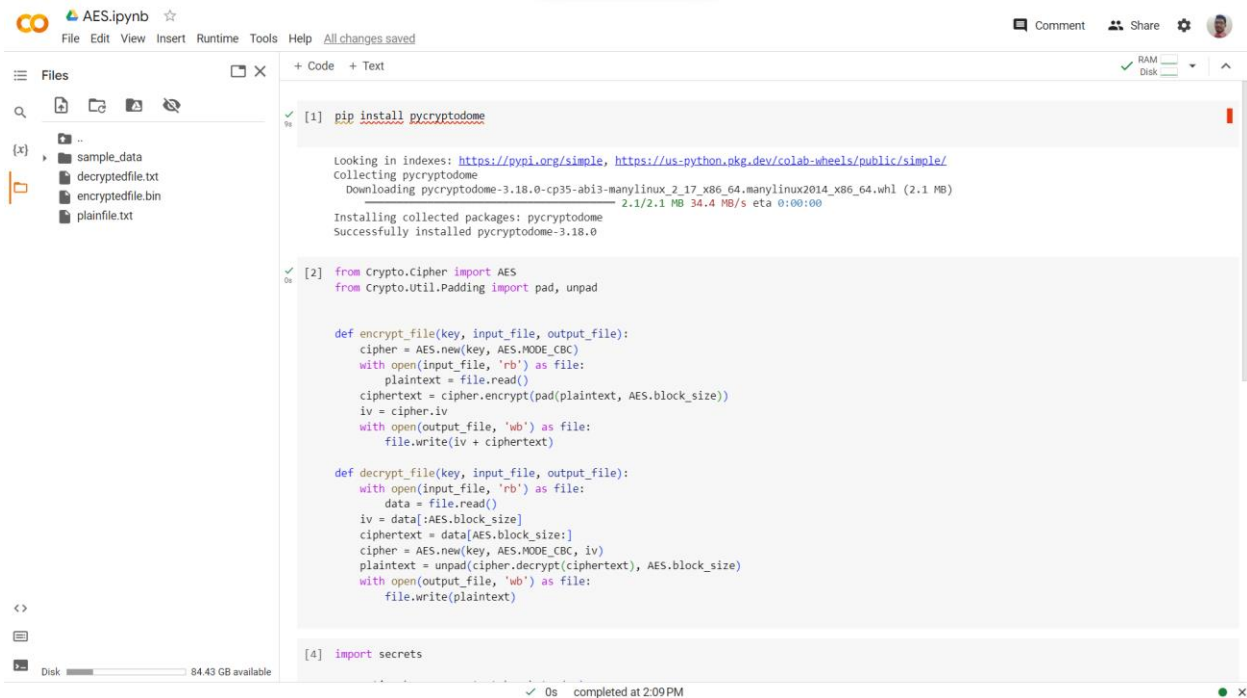
input_file = 'plainfile.txt'
output_file = 'encryptedfile.bin'
encrypt_file(encryption_key, input_file, output_file)
print(output_file)
```

Here plainfile.txt is the file that gets encrypted and gets stored in encryptedfile.bin.

Now let's pass encryptedfile.bin as input file to the decryptfile file and decrypt it.

```
input_file = 'encryptedfile.bin'
output_file = 'decryptedfile.txt'
decrypt_file(encryption_key, input_file, output_file)
```

Screenshots:



The screenshot displays a Jupyter Notebook titled 'AES.ipynb'. The left sidebar shows a file explorer with the following files: 'sample\_data', 'decryptedfile.txt', 'encryptedfile.bin', and 'plainfile.txt'. The main area contains four code cells. Cell [1] shows the command to install pycryptodome, with output indicating the package was successfully installed. Cell [2] contains the implementation of two functions: 'encrypt\_file' and 'decrypt\_file', both using AES in CBC mode. Cell [3] is empty. Cell [4] shows the command to import secrets. The bottom status bar indicates 'Disk: 84.43 GB available' and 'completed at 2:09 PM'.

```
[1] pip install pycryptodome

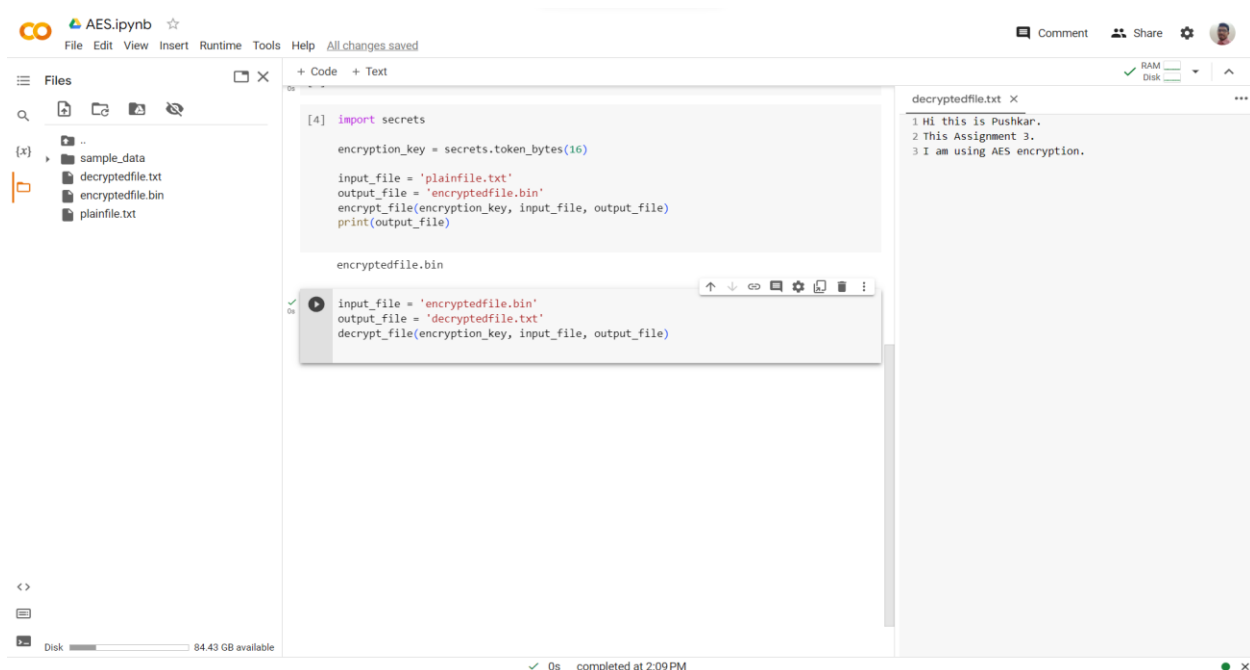
looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
collecting pycryptodome
  Downloading pycryptodome-3.18.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
    2.1/2.1 MB 34.4 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.18.0

[2] from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

def encrypt_file(key, input_file, output_file):
    cipher = AES.new(key, AES.MODE_CBC)
    with open(input_file, 'rb') as file:
        plaintext = file.read()
    ciphertext = cipher.encrypt(pad(plaintext, AES.block_size))
    iv = cipher.iv
    with open(output_file, 'wb') as file:
        file.write(iv + ciphertext)

def decrypt_file(key, input_file, output_file):
    with open(input_file, 'rb') as file:
        data = file.read()
    iv = data[:AES.block_size]
    ciphertext = data[AES.block_size:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)
    with open(output_file, 'wb') as file:
        file.write(plaintext)

[4] import secrets
```



## SECURITY ANALYSIS:

### POTENTIAL THREATS:

**Brute force attacks:** AES is vulnerable to exhaustive key search if the key size is small or weak. As computational power increases, it becomes easier for attackers to try all possible keys.**Side-channel attacks:** AES implementations can leak information through side channels, such as timing or power consumption, which can be exploited by attackers to recover the key.**Implementation flaws:** Weaknesses in the implementation of AES algorithms can lead to vulnerabilities, such as incorrect key handling, improper padding, or insufficient entropy in key generation.

### COUNTERMEASURES AND BEST PRACTICES:

**Use a strong and sufficiently long key:** Choosing a key with sufficient entropy and length (128, 192, or 256 bits) makes brute force attacks computationally infeasible.**Implement proper key management:** Protect the encryption key with strong access controls and consider key rotation or key distribution mechanisms depending on the application requirements.**Employ secure cryptographic libraries:** Use trusted and well-tested cryptographic libraries that implement AES securely, as they often include countermeasures against side-channel attacks.**Implement secure coding**

practices: Follow secure coding guidelines to prevent implementation flaws, such as using well-vetted libraries, avoiding insecure random number generators, and ensuring proper key handling and storage. Regularly update and patch software: Keep your cryptographic software and libraries up to date with the latest security patches to address any discovered vulnerabilities.

## LIMITATIONS:

AES alone does not provide authentication: It only offers confidentiality, not integrity or authenticity. To ensure data integrity and authenticity, it is recommended to use AES in combination with cryptographic primitives like digital signatures or message authentication codes (MACs). Encryption is only as secure as the key management: Strong encryption is essential, but proper key management, including secure key generation, storage, and distribution, is equally crucial for overall security. AES performance considerations: While AES is generally efficient, it may have performance implications for certain applications, especially when large amounts of data need to be encrypted or decrypted in real-time. It's important to evaluate the performance requirements and choose appropriate modes of operation and key sizes accordingly.

## CONCLUSION:

AES is widely regarded as a secure and efficient cryptographic algorithm. However, its security depends not only on the algorithm itself but also on proper key management, implementation security, and adherence to best practices. Regular updates and patching, secure coding practices, and the use of strong keys are crucial to mitigating potential vulnerabilities. Cryptography plays a vital role in ensuring the confidentiality, integrity, and authenticity of sensitive data in various cybersecurity applications. Ethical hackers leverage cryptographic knowledge to identify and fix vulnerabilities, contributing to overall system security.