ASSIGNMENT – 3

NAME – HIMANSHU

REG.NO – 20MIS7089

Symmetric Algorithm: Advanced Encryption Standard (AES) :

1. How it works: AES is a symmetric block cipher that encrypts and decrypts data in fixed-size blocks (128 bits) using a secret key of 128, 192, or 256 bits. It uses a series of transformations, including substitution, permutation, and mixing of plaintext data to produce the ciphertext.
2. Key strengths and advantages: AES is fast, secure, and widely adopted. It has been extensively analysed and is considered secure against known attacks.

3. Vulnerabilities or weaknesses: Side-channel attacks, such as timing attacks and cache attacks, can potentially reveal the secret key.
4. Real-world examples: AES is used in various applications, including secure communications (TLS/SSL), file encryption, and VPNs.

Asymmetric Algorithm: RSA:

1. How it works: RSA is an asymmetric algorithm that uses a pair of keys (public and private). The public key is used to encrypt data, while the private key is used to decrypt it. RSA is based on the mathematical properties of large prime numbers and modular exponentiation.

2. Key strengths and advantages: RSA provides strong security due to the difficulty of factoring large prime numbers. It enables secure key exchange and digital signatures.
3. Vulnerabilities or weaknesses: RSA is vulnerable to attacks if weak keys are used or if the implementation is flawed. It is also slower than symmetric algorithms.
4. Real-world examples: RSA is used in secure communications (TLS/SSL), digital signatures, and public key infrastructure (PKI).

Hash Function: SHA-256:

1. How it works: SHA-256 is a cryptographic hash function that takes

an input of arbitrary length and produces a fixed-size output (256 bits). It is designed to be a one-way function, meaning it should be computationally infeasible to reverse the process.

2. Key strengths and advantages: SHA-256 is collision-resistant, preimage-resistant, and second preimage-resistant, making it suitable for various cryptographic applications.
3. Vulnerabilities or weaknesses: No significant weaknesses have been found in SHA-256.
4. Real-world examples: SHA-256 is used in digital signatures, password hashing, and blockchain technology (e.g., Bitcoin).

Implementation:

AES Encryption and Decryption in Java
Scenario: Securely store and retrieve sensitive data in a file using AES encryption and decryption.
Step-by-step instructions:

1. Import necessary libraries.
2. Generate an AES secret key.
3. Encrypt the plaintext data using the secret key.
4. Save the encrypted data to a file.
5. Read the encrypted data from the file.
6. Decrypt the encrypted data using the secret key.

Testing and results:

1. Generate an AES secret key.
2. Encrypt a plaintext message using the secret key.
3. Save the encrypted data to a file.
4. Read the encrypted data from the file.
5. Decrypt the encrypted data using the secret key.
6. Verify that the decrypted data matches the original plaintext message.

Crypto.java > Crypto > generateAESKey()

```java
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.NoSuchAlgorithmException;

public class Crypto{
    public static SecretKey generateAESKey() throws NoSuchAlgorithmException {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(128);
        return keyGenerator.generateKey();
    }
    public static byte[] encrypt(String plaintext, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        return cipher.doFinal(plaintext.getBytes(charsetName:"UTF-8"));
    }
    public static void saveToFile(String fileName, byte[] encryptedData) throws Exception {
        Files.write(Paths.get(fileName), encryptedData);
    }
    public static byte[] readFromFile(String fileName) throws Exception {
        return Files.readAllBytes(Paths.get(fileName));
    }
    public static String decrypt(byte[] encryptedData, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        return new String(cipher.doFinal(encryptedData), charsetName:"UTF-8");
    }
    }
```

```
        at Alice.main(Alice.java:17)
PS C:\Users\himan\OneDrive\Documents\20MIS7089>  c:; cd 'c:\Users\himan\OneDrive\Documents\20MIS7089'; & 'C:\Program Files\Java\jdk-11.0.17\bin\java
.exe' '-cp' 'C:\Users\himan\AppData\Roaming\Code\User\workspaceStorage\d26b9915659f01e1887b05160621db83\redhat.java\jdt_ws\20MIS7089_36e259c6\bin' '
Alice'
Exception in thread "main" java.net.BindException: Address already in use: NET_Bind
        at java.base/java.net.PlainSocketImpl.bind0(Native Method)
        at java.base/java.net.PlainSocketImpl.socketBind(PlainSocketImpl.java:132)
        at java.base/java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:452)
        at java.base/java.net.ServerSocket.bind(ServerSocket.java:381)
        at java.base/java.net.ServerSocket.<init>(ServerSocket.java:243)
        at java.base/java.net.ServerSocket.<init>(ServerSocket.java:135)
        at Alice.main(Alice.java:17)
PS C:\Users\himan\OneDrive\Documents\20MIS7089>
```

Security analysis: Potential threats or vulnerabilities:

1. Side-channel attacks: Timing attacks and cache attacks can potentially reveal the secret key.
2. Weak key management: Storing the secret key insecurely can lead to unauthorized access.

Countermeasures and best practices:

1. Implement countermeasures against side-channel attacks, such as constant-time implementations.
2. Securely store and manage the secret key, e.g., using a hardware security module (HSM) or a secure key store.

Limitations and trade-offs:

1. AES is not suitable for public key exchange or digital signatures.
2. Key management and secure storage are critical for maintaining security.

Conclusion:

Cryptography plays a vital role in cybersecurity and ethical hacking. AES, RSA, and SHA-256 are widely used cryptographic algorithms that provide strong security when implemented correctly. Implementing AES encryption and decryption in Java demonstrates the practical application of cryptography for securing sensitive data. It is essential to consider

potential threats and vulnerabilities and apply countermeasures and best practices to enhance security.