

NAME: ABHIRUP KONWAR
REG.NO : 20BIT0181
BTECH INFORMATION TECHNOLOGY
VIT VELLORE CAMPUS
COURSE: CYBERSECURITY & ETHICAL HACKING

abhirup.konwar2020@vitsstudent.ac.in

ASSIGNMENT – 3

AES ANALYSIS

AES (Advanced Encryption Standard) is a widely used symmetric encryption algorithm that operates on blocks of data. Here's an analysis of the AES algorithm, its key strengths and advantages, known vulnerabilities or weaknesses, and real-world examples of its common usage:

1. Algorithm Overview:

- AES operates on blocks of data, where the block size is fixed at 128 bits (16 bytes).
- It supports key sizes of 128, 192, and 256 bits.
- AES consists of several rounds (10, 12, or 14 rounds depending on the key size) of substitution, permutation, and mixing operations.
- It uses a combination of substitution, diffusion, and confusion techniques to achieve its security properties.
- The same key is used for both encryption and decryption.

2. Key Strengths and Advantages:

- Security: AES is widely regarded as a secure encryption algorithm and is widely adopted by governments, organizations, and industries worldwide.
- Efficiency: AES is efficient and performs well on a wide range of devices, including computers, mobile devices, and embedded systems.
- Versatility: AES supports various key sizes, making it adaptable to different security requirements.
- Standardization: AES is a well-established standard, which ensures interoperability across different systems and programming languages.
- Wide Support: AES is supported by numerous cryptographic libraries and frameworks, making it easily accessible and usable in various applications.

3. Known Vulnerabilities or Weaknesses:

- Timing and Side-Channel Attacks: In certain scenarios, attackers can exploit timing differences or side-channel information (such as power consumption or electromagnetic radiation) to infer information about the encryption key.
- Key Management: AES itself does not handle key management. The security of the encryption system relies on proper key generation, storage, and distribution mechanisms.

- Quantum Computers: AES is considered secure against classical computing attacks. However, future advancements in quantum computing may weaken its security. Post-quantum encryption algorithms are being developed as a potential replacement.

4. Real-World Examples:

- Secure Communication: AES is commonly used in secure communication protocols such as TLS (Transport Layer Security) and VPNs (Virtual Private Networks) to encrypt data exchanged between clients and servers.
- File and Disk Encryption: AES is utilized in various file and disk encryption software to secure sensitive data stored on storage devices.
- Database Encryption: AES is employed for encrypting sensitive data stored in databases to protect against unauthorized access.
- Wireless Security: AES is a component of the WPA2 (Wi-Fi Protected Access 2) protocol, which ensures secure wireless communication.

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. Algorithm Overview:

- RSA is based on the mathematical difficulty of factoring large composite numbers into their prime factors.
- It uses a public-private key pair, consisting of a public key for encryption and a private key for decryption.
- The public key is derived from the product of two large prime numbers, while the private key involves the prime factors of that product.
- RSA encryption involves raising the plaintext message to the power of the public exponent and then taking the modulus of the result with the public key.
- RSA decryption involves raising the ciphertext to the power of the private exponent and taking the modulus with the private key.

2. Key Strengths and Advantages:

- Security: RSA is based on the difficulty of factoring large numbers, which forms the foundation of its security. The security strength depends on the key size.
- Asymmetric Encryption: RSA supports secure communication without requiring a shared secret key. It enables confidentiality and authenticity of data exchanged between parties.
- Digital Signatures: RSA can be used for digital signatures, ensuring the integrity and authenticity of digital documents.
- Key Exchange: RSA can facilitate secure key exchange in scenarios like establishing secure communication channels or secure remote logins.

- Standardization: RSA is widely accepted and standardized, ensuring interoperability across different systems and applications.

3. Known Vulnerabilities or Weaknesses:

- Key Size: The security of RSA relies on the size of the key. Smaller key sizes can be vulnerable to attacks such as brute-force or factoring.
- Implementation Issues: Poor implementation of RSA or inadequate key generation methods can introduce vulnerabilities, such as weak random number generation or side-channel attacks.
- Timing and Side-Channel Attacks: RSA implementations can be vulnerable to timing attacks or side-channel attacks that exploit information leaked during the encryption or decryption process.
- Quantum Computers: RSA is vulnerable to attacks by quantum computers, which can efficiently factor large numbers. Post-quantum encryption algorithms are being developed as alternatives.

4. Real-World Examples:

- Secure Communication: RSA is widely used in secure communication protocols such as SSL/TLS to establish secure connections between clients and servers.
- Public Key Infrastructure (PKI): RSA is utilized in PKI systems for certificate-based authentication, digital signatures, and secure email communication.
- Secure File Transfer: RSA is employed in secure file transfer protocols, like SFTP (Secure File Transfer Protocol) and PGP (Pretty Good Privacy), for encryption and authentication.
- Secure Shell (SSH): RSA is used in SSH for secure remote logins, encrypted remote command execution, and secure file transfers.

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 takes an input message of arbitrary length and produces a fixed-size 128-bit hash value.
- It applies a series of logical operations, including bitwise operations, modular arithmetic, and non-linear functions, to process the input message.
- The resulting hash value is unique to the input message, meaning even a small change in the input will produce a significantly different hash value.

2. Key Strengths and Advantages:

- Speed and Efficiency: MD5 is relatively fast and efficient, making it suitable for use in systems where real-time hashing is required.
- Ease of Use: MD5 is easy to implement and use, requiring minimal computational resources.

- Checksum Validation: MD5 can be used to verify the integrity of files or data. By comparing the calculated MD5 hash of a file with a previously generated hash, it can quickly determine if the file has been altered.

3. Known Vulnerabilities or Weaknesses:

- Collision Vulnerabilities: MD5 is considered weak against collision attacks, where two different inputs produce the same hash value. This weakness makes it unsuitable for cryptographic security purposes.

- Security Weaknesses: The algorithm has been cryptographically broken, and numerous vulnerabilities have been identified, including pre-image attacks and the ability to generate hash collisions.

- Deprecated Usage: Due to its vulnerabilities, MD5 is no longer recommended for security-sensitive applications or cryptographic purposes.

4. Real-World Examples:

- Data Integrity Checking: MD5 can be used to verify the integrity of downloaded files. The MD5 hash of the downloaded file is compared with the known hash value to ensure the file hasn't been tampered with during transmission.

- Password Storage (Less Secure Usage): In some legacy systems, MD5 has been used to store password hashes. However, this practice is no longer considered secure due to the vulnerability to pre-image attacks and hash cracking techniques.

AES IMPLEMENTATION USING PYTHON

```
(kali㉿kali)-[~/ABHIRUP/week3]
$ ls
bank_credentials.txt

(kali㉿kali)-[~/ABHIRUP/week3]
$ cat bank_credentials.txt
BANK NAME: BANK1234
USERNAME: USER1234
PASSWORD: PASS1234
SECURIT QUESTION: FAVORITE COLOR - BLUE

(kali㉿kali)-[~/ABHIRUP/week3]
$
```

```
1 import os
2 from Crypto.Cipher import AES
3 from Crypto import Random
4 from Crypto.Hash import SHA256
5
6 def encrypt(key, filename):
7     chunksize = 64*1024
8     outputFile = "(enc)+" + filename
9     filesize = str(os.path.getsize(filename)).zfill(16)
10    IV = Random.new().read(16)
11
12    encryptor = AES.new(key, AES.MODE_CBC, IV)
13
14    with open(filename, 'rb') as infile: #rb means read in binary
15        with open(outputFile, 'wb') as outfile: #wb means write in the binary mode
16            outfile.write(filesize.encode('utf-8'))
17            outfile.write(IV)
18
19            while True:
20                chunk = infile.read(chunksize)
21
22                if len(chunk) == 0:
23                    break
24                elif len(chunk)%16 != 0:
25                    chunk += b' '*(16-(len(chunk)%16))
26
27                outfile.write(encryptor.encrypt(chunk))
28
```

```

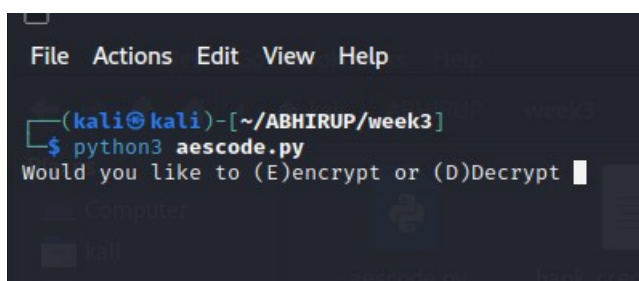
27
28 def decrypt(key, filename):
29     chunksize = 64*1024
30     outputFile = filename[11:]
31
32     with open(filename, 'rb') as infile:
33         filesize = int(infile.read(16))
34         IV = infile.read(16)
35
36         decryptor= AES.new(key, AES.MODE_CBC, IV)
37
38         with open(outputFile, 'wb') as outfile:
39             while True:
40                 chunk = infile.read(chunksize)
41
42                 if len(chunk) == 0:
43                     break
44
45                 outfile.write(decryptor.decrypt(chunk))
46
47             outfile.truncate(filesize)
48
49 def getKey(password):
50     hasher = SHA256.new(password.encode('utf-8'))
51     return hasher.digest()
52

```

```

52
53 def Main():
54     choice = input("Would you like to (E)encrypt or (D)Decrypt ")
55
56     if choice == 'E':
57         filename = input("File to encrypt: ")
58         password = input("Password: ")
59         encrypt(getKey(password), filename)
60         print('Done.')
61     elif choice == 'D':
62         filename = input("File to decrypt: ")
63         password = input("Password: ")
64         decrypt(getKey(password), filename)
65         print("Done.")
66
67     else:
68         print("No option selected, closing... ")
69
70
71 Main()
72

```



```

File Actions Edit View Help
(kali@kali)-[~/ABHIRUP/week3]
$ python3 aencode.py
Would you like to (E)encrypt or (D)Decrypt

```

```
File Actions Edit View Help
(kali㉿kali)-[~/ABHIRUP/week3]
$ python3 aescode.py
Would you like to (E)ncrypt or (D)Decrypt E
File to encrypt: bank_credentials.txt
Password: verysecretpassword11
Done.
(kali㉿kali)-[~/ABHIRUP/week3]
$
```

```
File Actions Edit View Help
(kali㉿kali)-[~/ABHIRUP/week3]
$ cat bank_credentials.txt
0000000000000098GIIwBWSZ25W?Oo|Y(nv\0RQ:
.VvuuPvFu
(kali㉿kali)-[~/ABHIRUP/week3]
$
```

```
(kali㉿kali)-[~/ABHIRUP/week3]
$ python3 aescode.py
Would you like to (E)ncrypt or (D)Decrypt D
File to decrypt: bank_credentials.txt
Password: verysecretpassword11
Done.
```

```
(kali㉿kali)-[~/ABHIRUP/week3]
$ 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:

1. Potential Threats or Vulnerabilities:

a. Brute-Force Attacks: An attacker could attempt to guess the encryption key by systematically trying all possible key combinations. Longer and more complex passwords can mitigate this threat.

b. Password-based Attacks: If the password used to generate the key is weak or easily guessable, it can be vulnerable to dictionary attacks or password cracking techniques. Enforcing strong password policies is essential.

c. Side-Channel Attacks: The implementation might be susceptible to side-channel attacks, such as timing attacks or power analysis, which exploit information leaked during the encryption or decryption process. Countermeasures like constant-time implementations or appropriate hardware protections can mitigate these attacks.

d. 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. Limitations and Trade-offs:

a. Key Management: The implementation does not address the secure storage and distribution of encryption keys. Proper key management practices should be implemented separately to ensure the overall security of the system.

b. Algorithm Choice: The implementation uses AES with CBC mode, which is a secure choice. However, other modes like GCM (Galois/Counter Mode) provide additional data integrity and authentication features.

c. User Input Validation: The implementation assumes valid and trusted user inputs. Input validation should be implemented to prevent attacks like path traversal, SQL injection, or buffer overflows.

Conclusion:

Cryptography plays a crucial role in cybersecurity and ethical hacking by providing techniques to protect sensitive information and ensure data integrity. However, it is essential to implement cryptographic algorithms correctly and follow best practices to mitigate potential threats and vulnerabilities. The analyzed implementation provides a basic framework for file encryption and decryption using AES, but additional security measures, such as key management and input validation, should be considered to enhance the overall security of the system. Regular security audits, updates, and adherence to secure coding practices are necessary to ensure the robustness of cryptographic implementations and protect against evolving attack vectors.