AES Encryption Report

Project 2
Data Security & Privacy
By: Matthew Hartshorn

Environment

My project was created utilizing Python 3.6 and the python Cryptography library. The code was run and tested on the Windows 10 Operating System (OS). Functional compatibility should remain consistent even when using Python3.5 or higher, regardless of the OS.

Implementation

The key generation methods are built using Python's build in random byte and random integer generators. The key generation only allows the standard AES key sizes of 128, 192, and 256. The keys are then written as a hexadecimal string, where each byte is two digits. The initialization vectors in the code are also derived from the same key generation method, using strictly the 128-bit key gen.

Encryption and decryption of the text is done using the Cryptography python lib. These encryption methods did not natively pad the input message. As such padding and padding removal functions had to be created in order to satisfy the Blocking of the message. Without padding the input the encryption methods would simply truncate the data and not properly encrypt the provided message. These padding methods were implemented using what is know as "PKCS5 Padding". This method of padding stores the number of padded bytes in place of the missing bytes. This methodology allows quick lookup of the amount of padding by simply reading the last byte.

ECB vs. CBC

Encryption Code Book (ECB) and Cypher Block Chaining (CBC) are two simple modes for the AES cypher. ECB simply only utilizes the cypher method to encrypt each block separately. This is the easiest method for AES encryption, in addition to being the most naïve. CBC mode, introduces the concept of an Initialization Vector (IV) which adds entropy to the cyphertext.

When using ECB, given an encryption key, the result of the encryption will always remain the same. There is no inherent randomness to the mode and as such means a reverse lookup of encryption results can be applied. This type of encryption is rather susceptible to Rainbow Table attacks as patterns in the resulting cyphertext can be present to an attacker.

However, CBC and the addition of the IV allow for the same secret key to be used more than once on the same piece of data without resulting in the same cyphertext. This prevents easy lookup and reverse engineering the encryption key purely based on the cyphertext.

CBC Mode Timing

To test the efficiency of the AES encryption using CBC mode, I derived a separate test to measure the runtime of both encryption and decryption independently. To properly analyze the runtime of these methods in which take only milliseconds, I had to run many iterations and average their runtime to gather good data. The following code is taken from the file `./src/aestimer.py`.

Encryption Timing

```
# Measure encryption timing
start = time.time()
for i in range(iterations):
    iv = keygenerator.generate(128)
    aescypher.encrypt(b_plaintext, key, mode, iv)
end = time.time()
elapsed = (end - start) * 1000

print("Encryption")
print("-------------------------")
print("Start: {0}".format(start))
print("End: {0}".format(end))
print("Elapsed: {0:.4f} ms".format(elapsed))
print("Avergae: {0:.4f} ms".format(elapsed / iterations))
```

Decryption Timing

```
# Generate cyphertext
iv = keygenerator.generate(128)
ct = aescypher.encrypt(b_plaintext, key, mode, iv)

# Measure decryption timing
start = time.time()
for i in range(iterations):
    aescypher.decrypt(ct, key, mode, iv)
end = time.time()
elapsed = (end - start) * 1000

print("Decryption")
print("-------------------------")
print("IV: {0}".format(iv.hex()))
print("Start: {0}".format(start))
print("End: {0}".format(end))
print("Elapsed: {0:.4f} ms".format(elapsed))
print("Average: {0:.4f} ms".format(elapsed / iterations))
```

The above code shows that after doing a specified number of iterations, you are able to successfully determine an average runtime for each encryption or decryption call.

Timing Results

```
Info
-------------------------
Plaintext: Welcome to data security and privacy .
Key: 4654004e8016690f1f5f7e78a721f2fdfd21a690a03906929b0e05e2046bfef6

Encryption
-------------------------
Start: 1520569198.52308
End: 1520569202.9677918
Elapsed: 4444.7117 ms
Avergae: 0.0444 ms

Decryption
-------------------------
IV: ca1c2e48c3e3d272ba218cdc4a847387
Start: 1520569202.9697983
End: 1520569206.7265074
Elapsed: 3756.7091 ms
Average: 0.0376 ms
```

Based on these results, it is reasonable to assume that the encryption/signing method takes a noticeably longer time to compute. This timing difference would become even more noticeable when encrypting larger data files. The results of this means that a user who is receiving encrypted data would be capable of decrypting the data faster than the other user could send, so long as the receiver has the IV’s and encryption keys accessable.