**ITIS 6200/8200 Principles of Information Security and Privacy**

**Fall Semester of 2018**

**Homework 1**

1. There are two types of encryption algorithms: symmetric key encryption and asymmetric key encryption (also called public key encryption). To reach the same level of security, the key length for different algorithms can vary greatly. Please answer: to reach the same level of security as the RSA 3072 asymmetric encryption algorithm, how long does the key need to be for a symmetric encryption algorithm, and how long does the key need to be for elliptic curve cryptography?

Hint: You can search in Wikipedia and you will find the answers. This is to help you to understand the security comparison between the two encryption algorithm types. Remember, asymmetric encryption algorithms are not always more secure than their symmetric companions. Actually, if Quantum Computers become available, many asymmetric encryption algorithms may be cracked.

**Answer 1:**

* RSA 3072 algorithm has cryptographic strength equivalent to AES algorithm with a 128-bit encryption key.
* Elliptic Curve Cryptography provides the same cryptographic strength as an RSA based system with much smaller key sizes. For instance, a 256-bit key in ECC is equivalent in strength to RSA 3072 bit key.

1. Please describe one example in computer security to show that cryptography cannot solve all problems in security.

**Answer 2:**

* An example to demonstrate that cryptography cannot resolve all the problems in security is its inability to protect against **Distributed Denial of Service attacks**. In a DDOS attack, the attacker develops specialized malware and spreads it to several vulnerable computers, turning them into an army of bots, also known as a botnet. The botnet is then instructed by the attacker to conduct a coordinated and distributed attack to overwhelm the target with a flood of internet traffic. This disrupts the normal traffic of a targeted server or network and renders a legitimate client inaccessible to the desired web service for a specific duration.

An example of DDOS is **SYN Flood attack** which consumes the resources of a target server and renders it unresponsive by exploiting the normal functioning of the TCP three-way handshake protocol. The attacker sends large volumes of SYN packets to every port on the target server to establish a connection. The server acknowledges this by sending back ACK packets and waits to receive an acknowledgment from the client (in this case, the attacker). However, the attacker never sends SYN-ACK packet back to the server and the connection stays open. Before the connection could time out, another SYN packet arrives and the process continues, thereby leaving a large number of connections half open. Eventually, the server’s connection table overflows, thus denying services to legitimate clients. Furthermore, the server itself might crash or malfunction.

This exemplifies that the use of cryptography cannot control the flow of SYN-ACK packets to prevent denial of service attacks.

* Another example to explore the vulnerability in cryptography is understanding the **Heartbleed bug in OpenSSL** cryptographic software library (version 1.0.1 through 1.0.1f and 1.0.2-beta), which allows an attacker to gain access to up to 64 KB of private memory of an application.

Two computers communicate with each other via a “heartbeat request” (RFC 6520 extension) to ensure that the connection between them is active. Crucially, the heartbeat request sent via one computer to another includes a payload and its size. The computer responding to this heartbeat request would contain the same payload and some padding in its response. One major implementation flaw here is that the computer receiving the heartbeat request never checks to ensure that the request is actually as long as it claims to be.

For instance, if an attacker sends a heartbeat request, claiming that it is 30 KB long but it’s actually just 10KB, the receiving computer would blindly set aside 30 KB of memory buffer. It would then store the 10KB of data it actually received and then send back the 10 KB data along with whatever is present in the next 20 KB of memory. This extra 20 KB of memory is the information that the attacker now has unauthorized access to. This memory could either be gibberish or it could be sensitive information such as encryption keys, usernames, passwords, and even personal and financial information. The attacker can carry out this attack multiple times without leaving any traces. Thus, the **Heartbleed Vulnerability** in OpenSSL shows that cryptography cannot solve all the issues in security.

1. Encryption usually needs a feature called “Avalanche Effect”, which means a small change in the input will cause large changes in the output. In this task we will do some experiments. You will need: (1) a plaintext file you generate (not need to be large, 1k byte or so should be fine); (2) an encryption software (I use AEScrypt); and (3) a binary editor/viewer (such as CFF Explorer).

Use the encryption software to encrypt your text file. Then change 1 byte or 1 bit in the text file, encrypt it again. Now use the binary editor to compare the two cipher text files. Are the differences big or small? In your homework submission, you need to attach the screenshots of the text files, and two files opened with the binary editor.

Extra question: Use your binary editor to change 1 bit in the cipher text file. Then use the software to decrypt. What do you get? Do you get a decrypted file similar to the original text file? Or will the software refuse to decrypt? Why do you think this happens?

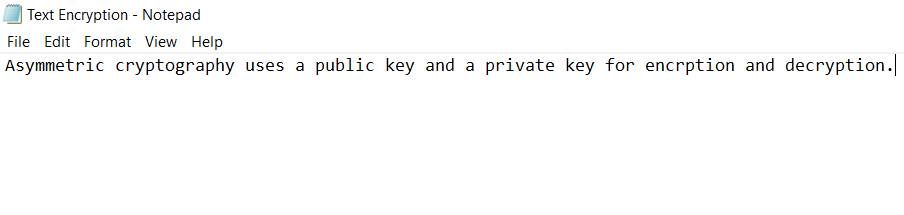
**Answer 3:**

For this experiment, I’ve taken a plaintext file called **“Text Encryption.txt”** with the contents: “Asymmetric cryptography uses a public key and a private key for encryption and decryption.” I then followed the following steps:

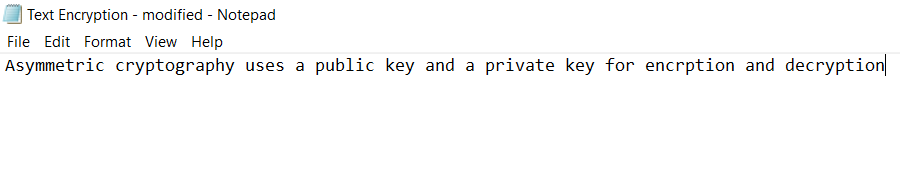
* Encrypted “Text Encryption.txt” using AEScrypt encryption software and used CFF Explorer to view the ciphertext file.
* Modified a byte in the original plaintext file by removing the period from the plaintext and performed encryption using AEScrypt again. Viewed the ciphertext using CFF Explorer.
* Performed bit by bit comparison of the two ciphertext files for differences.

**Below are the results**:

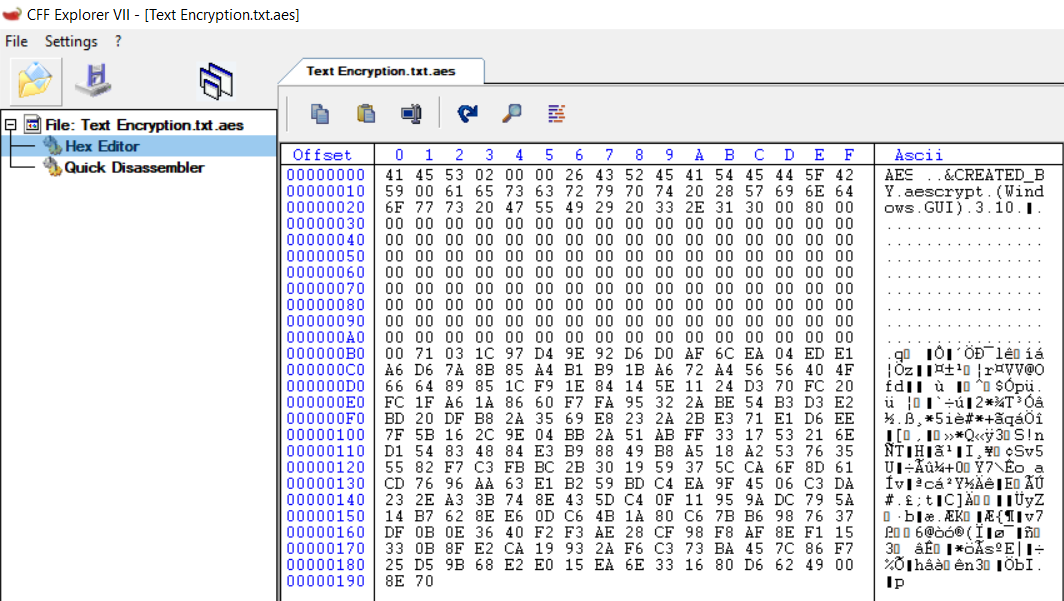
* **Text Encryption.txt: Plaintext file prior to any modification:**



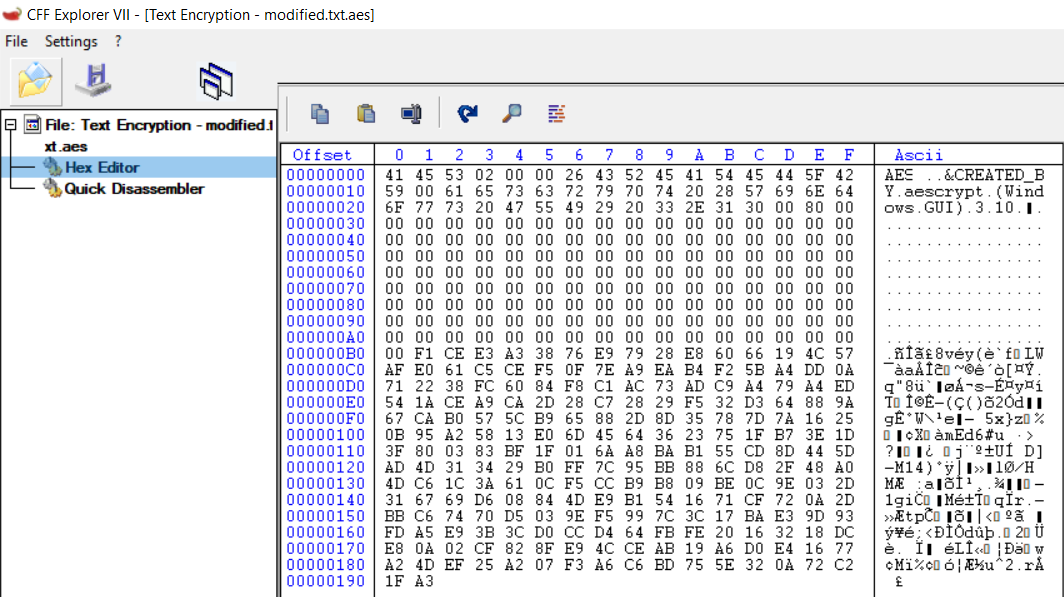
* **Text Encryption-modified.txt: Plaintext file after modification (the period has been removed):**



* **Ciphertext file for the original plaintext (Text Encryption.txt):**



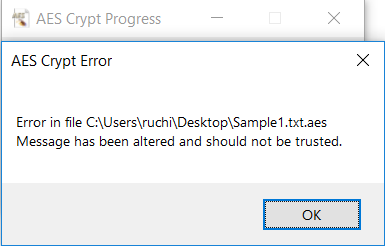
* **Ciphertext file for the modified plaintext (Text Encryption-modified.txt):**



It can be seen from the above screenshots that more than 50% of the ciphertext changes upon removal of just a period from the original plaintext. It can be concluded that changing a single bit or byte of plaintext produces a significant change in the ciphertext. Hence, the **Avalanche Effect** is satisfied.

**Answer to the extra question:**

Upon modification of a bit in the ciphertext file, **the software refuses to perform decryption** and displays the following error:



In my opinion, this could happen because of the following reasons:

* There’s one to one mapping between the plaintext bits and ciphertext bits. Modification of a bit in ciphertext interrupts this mapping of data. Hence, when we change a bit in the encrypted file using the binary editor and try to perform decryption using AEScrypt, the software expects the old data only and will not perform any action with the new data. Therefore, the error mentioned above is thrown.
  + **Scenario 1:** Since the ciphertext’s length could be larger than the plaintext length, there would be a certain bit or bits which would not map to a corresponding bit in the plaintext (under a specific key). Hence, in this case, decryption fails.
  + **Scenario 2 (when ciphertext’s length is equal to the plaintext length):** The ciphertext bit which we are modifying might already be mapped to a plaintext bit. This newly modified ciphertext bit can now correspond to another bit in plaintext. This would cause redundancy since two bits in the plaintext would now be mapped to a single bit in ciphertext. Hence, decryption fails.