CSI4139
Lab 1
Name: NamChi Nguyen
Student #: 7236760

Group members:
Julian Templeton - 8229400
Sean McAvoy - 8263210

Part 1: File protection program

Our program was implemented using the programming language Java and the Java Cryptography Architecture (JCA)/security API packages in a symmetric key environment to encrypt and decrypt a text file. It was assumed that the input was a text file and the output was the same text file being encrypted and decrypted. The two main security functions created was encryptSign and decryptSign. The function encryptSign would sign the text file and then encrypt it whereas decryptSign decrypted the same file and verified its signature.

Chosen algorithms:

RSA: The key pair was generated using this algorithm for creating a public and private key pair. It was chosen as it is one of the standard algorithms that is widely used for public key encryption.

- Key length: 1024 bits that results in 80 bits of security which is the minimum security that's required. The key length was chosen since we were handling simple text files to encrypt and decrypt and not large or confidential documents.

```
private KeyPair getKeyPair() throws NoSuchAlgorithmException {
    KeyPairGenerator keyPair = KeyPairGenerator.getInstance("RSA");
    keyPair.initialize(1024);
    return keyPair.genKeyPair();
}
```

AES: The symmetric key was generated using this algorithm which is used in symmetric encryption. It was chosen as it is one of the most prominent standard algorithms widely used today and became the replacement of DES.

```
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
cryptoKey = keyGenerator.generateKey();
```

SHA256withRSA: The hashing and signing were completed by using the Signature class instead of separating the hashing and signing with a MessageDigest object and Signature object.

When creating the signature, we used this algorithm in which the message digest SHA256 is used to condense large amounts of data which the hash is then signed with RSA [1].

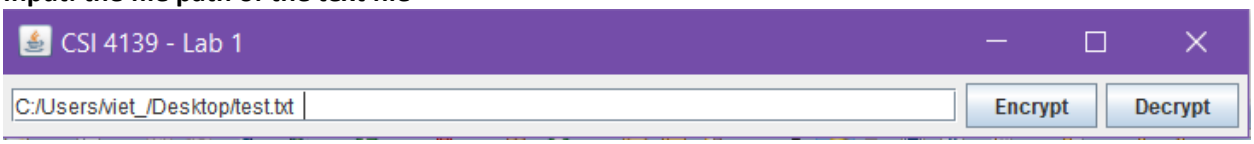
SHA256withRSA was chosen over SHA1withRSA with the intention of increasing the security since SHA1 is more vulnerable to hash collision attacks [6].

```
// Create signature and apply to file
signature = Signature.getInstance("SHA256withRSA");
```

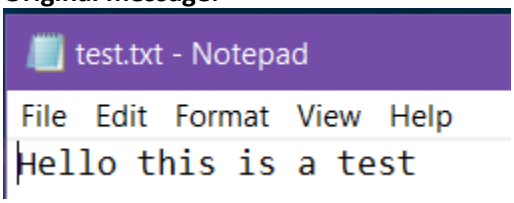
Note: However, it was seen in class that our implementation isn't secure since we used different levels of security in which the algorithm with the weakest security would override the algorithm with the strongest security (ie. RSA-1024 for generating the key pair would override the security of using SHA256). In this case, the security levels of the algorithms should be equivalent, so it would have been better to implement all algorithms with either 80-bit or 256-bit security to be consistent.

Program flow:

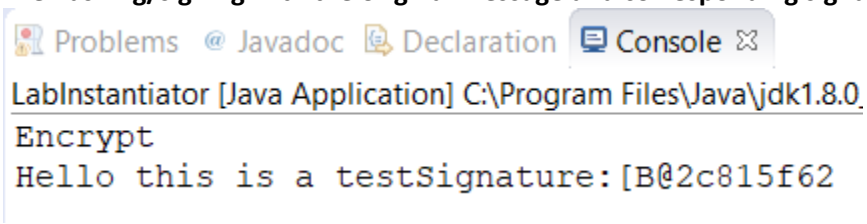
Input: the file path of the text file



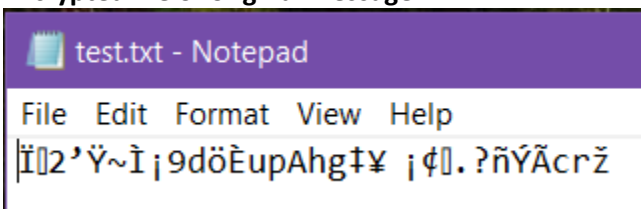
Original message:



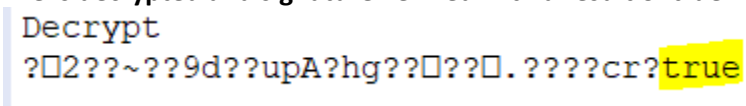
File hashing/signing with the original message and corresponding signature:



Encrypted file of original message:

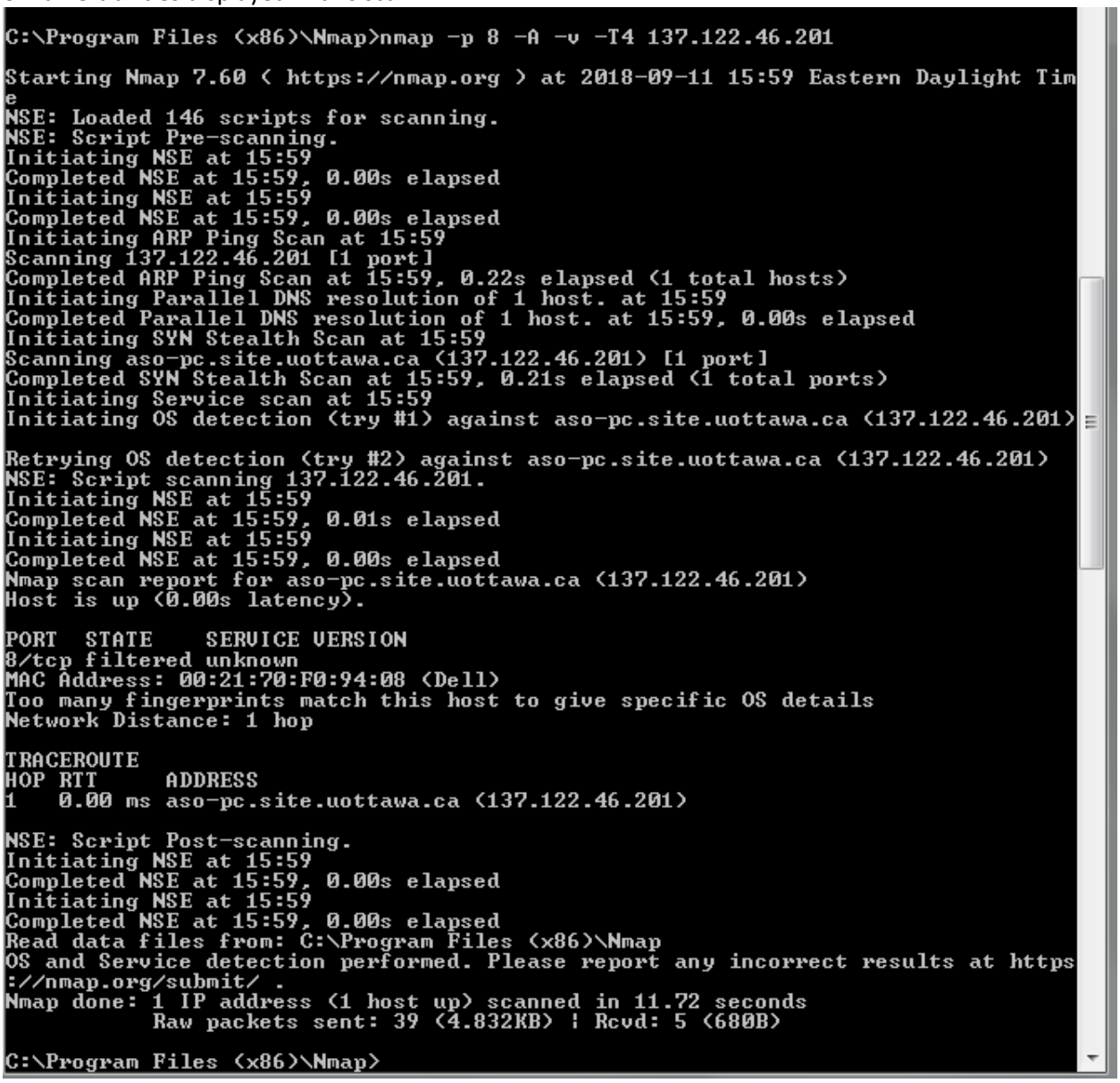


File is decrypted and signature verified with a result of true:



Part 2: Server weaknesses

Note: Since we only scanned on Port 8 instead of scanning on every port, there were a limited amount of vulnerabilities displayed in this scan.



```
C:\Program Files (x86)\Nmap>nmap -p 8 -A -v -T4 137.122.46.201

Starting Nmap 7.60 ( https://nmap.org ) at 2018-09-11 15:59 Eastern Daylight Tim
e
NSE: Loaded 146 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 15:59
Completed NSE at 15:59, 0.00s elapsed
Initiating NSE at 15:59
Completed NSE at 15:59, 0.00s elapsed
Initiating ARP Ping Scan at 15:59
Scanning 137.122.46.201 [1 port]
Completed ARP Ping Scan at 15:59, 0.22s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 15:59
Completed Parallel DNS resolution of 1 host. at 15:59, 0.00s elapsed
Initiating SYN Stealth Scan at 15:59
Scanning aso-pc.site.uottawa.ca (137.122.46.201) [1 port]
Completed SYN Stealth Scan at 15:59, 0.21s elapsed (1 total ports)
Initiating Service scan at 15:59
Initiating OS detection (try #1) against aso-pc.site.uottawa.ca (137.122.46.201)

Retrying OS detection (try #2) against aso-pc.site.uottawa.ca (137.122.46.201)
NSE: Script scanning 137.122.46.201.
Initiating NSE at 15:59
Completed NSE at 15:59, 0.01s elapsed
Initiating NSE at 15:59
Completed NSE at 15:59, 0.00s elapsed
Nmap scan report for aso-pc.site.uottawa.ca (137.122.46.201)
Host is up (0.00s latency).

PORT  STATE    SERVICE VERSION
8/tcp filtered unknown
MAC Address: 00:21:70:F0:94:08 (Dell)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   0.00 ms aso-pc.site.uottawa.ca (137.122.46.201)

NSE: Script Post-scanning.
Initiating NSE at 15:59
Completed NSE at 15:59, 0.00s elapsed
Initiating NSE at 15:59
Completed NSE at 15:59, 0.00s elapsed
Read data files from: C:\Program Files (x86)\Nmap
OS and Service detection performed. Please report any incorrect results at https
://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.72 seconds
           Raw packets sent: 39 (4.832KB) | Rcvd: 5 (680B)

C:\Program Files (x86)\Nmap>
```

Figure 1. Nmap scan of IP address 137.122.46.201 on Port 8

We used the scanning tool Nmap to test the following IP address 137.122.46.201 and ran the command *nmap -p 8 -A -v -T4 137.122.46.201*.

This command can be broken down as follows [2]:
-p 8: port scan on port 8
-A: enable OS and version detection, script scanning and traceroute
-v: increase verbosity level (ie. amount of details)
-T4: aggressive speed scan

Vulnerabilities detected:

1. The port number is known and can be exploited.

   Weakness: An attacker could cause a denial of service (DoS) or distributed denial of service (DDoS) such as UDP or ICMP flooding that flood the target with UDP or ICMP packets respectively [3].

   Suggested Solution: Deploy a firewall to block ports that shouldn't be open. As shown in Figure 1, the scan indicates that the port is *filtered* which means there already exists a firewall or a filter that is blocking the port, so the packet is dropped [5].

2. The DNS is known and can be exploited.

   Weakness: An attacker could cause DNS cache poisoning/spoofing. Cache poisoning is when an attacker can place incorrect information in the server cache in which the attacker can then redirect traffic with their false reply that's now cached in the server [4].

   Suggested Solution: Increase the inconsistency of requests that are sent out by the server by mixing up the upper and lower case letters in the domain names so that the attacker won't be able to spoof easily with their false responses [4].

Overall, to conclude, these were some of the vulnerabilities that could be detected from the scan. From our scan, the OS and version couldn't be detected. This is beneficial that this information wasn't revealed in the scan as knowing the OS system of the device used could help the attacker to narrow down their approach. For example, if the OS was outdated and longer in use with regular patches to maintain security, the attacker could use this to their advantage by implementing malware.

References:

[1] Oracle, "Java Cryptography Architecture (JCA) Reference Guide", 2018, [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#Signature. [Accessed Sept. 20, 2018]

[2] N. House, "Nmap Cheat Sheet", 2017, [Online]. Available: https://www.stationx.net/nmap-cheat-sheet/. [Accessed Sept. 20, 2018]

[3] Imperva Incapsula, "DDOS Attacks", 2018, [Online]. Available: https://www.incapsula.com/ddos/ddos-attacks.html. [Accessed Sept. 20, 2018]

[4] P. Rubens, "How to Prevent DNS Attacks", 2018, [Online]. Available: https://www.esecurityplanet.com/network-security/how-to-prevent-dns-attacks.html. [Accessed Sept. 20, 2018]

[5] G. Lyon, "Port Scanning Basics", n.d. [Online]. Available: https://nmap.org/book/man-port-scanning-basics.html. [Accessed Sept. 20, 2018]

[6] KeyCDN, "SHA1 vs SHA256", July 22, 2016, [Online]. Available: https://www.keycdn.com/support/sha1-vs-sha256/. [Accessed Sept. 20, 2018]

Lab code:

[1] Marilena, "Java – Digital Signatures example", Jan 9, 2017, [Online]. Available: https://www.mkyong.com/java/java-digital-signatures-example/. [Accessed Sept. 12, 2018]

[2] Stackoverflow, "Simplest way to encrypt a text file", 2015, [Online]. Available: https://stackoverflow.com/questions/27962116/simplest-way-to-encrypt-a-text-file-in-java. [Accessed Sept. 12, 2018]

[3] Stackoverflow, "AES Encryption and Decryption in Java", 2013, [Online]. Available: https://stackoverflow.com/questions/20796042/aes-encryption-and-decryption-with-java?noredirect=1&lq=1. [Accessed Sept. 12, 2018]