CS 306 Homework 2

Theodore Jagodits

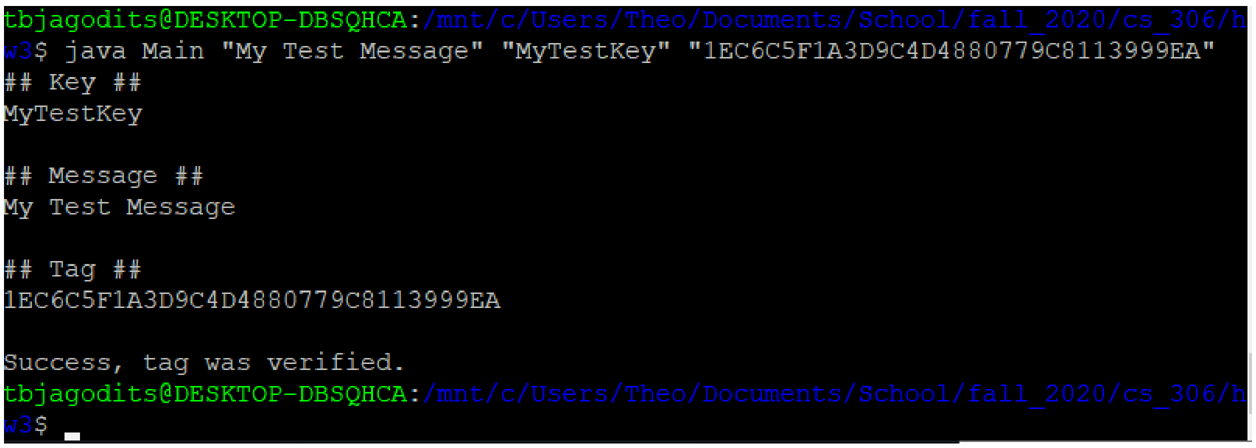
11/19/20

I pledge my Honor that I have abided by the Stevens Honor System.

1.1

Please see code attached.

1.2



```
tbjagodits@DESKTOP-DBSQHCA:/mnt/c/Users/Theo/Documents/School/fall_2020/cs_306/h
w3$ java Main "My Test Message" "MyTestKey" "1EC6C5F1A3D9C4D4880779C8113999EA"
## Key ##
MyTestKey

## Message ##
My Test Message

## Tag ##
1EC6C5F1A3D9C4D4880779C8113999EA

Success, tag was verified.
tbjagodits@DESKTOP-DBSQHCA:/mnt/c/Users/Theo/Documents/School/fall_2020/cs_306/h
w3$ _
```

1.3

I implemented CBC mac with the code provided. I chose this because it was easy to implement and did not require a lot of bandwidth. Also, it was discussed in the slides.

1.4

According to the lecture slides, the CBC MAC implementation works only securely if the messages are of predetermined size. Other than that it is pretty secure I believe.

2.1

Eve can use the old invalid certificate to validate it with Mallory. Since it is already validated with Mallory, Eve can receive all the messages sent to Bob. This is known as a replay attack since we are using old “valid” credentials to certify that the attacker has access.

2.2

Replay attacks cannot happen if you use timestamp metadata because you cannot use old certificates to validate an request anymore. The timestamped data would show that this is invalid and unless Alice doesn’t do this attack within a day, all of Bob’s are safe from this type of attack.

3.1

If a hacker compromises the red server (the one with all passwords including honeywords), then the hacker still cannot find the correct password. They would have to guess which one would be the correct one. This is safer because it turns into a probabilistic attack, meaning there is a high probability that they guess wrong passwords and a low probability they will get in on the first tries.

If a hacker compromises the blue server (the one with indices), then there are not even passwords. They will have a list of indices but nothing to guess passwords with. Compromising the blue server brings nothing of value unless you have the red one.

3.2

Honeywords are close approximations of the real passwords with small changes. With this in mind the attacker, as mentioned before cannot distinguish the real from fake passwords effectively. The attacker then has to try all these different passwords in the red server. When an attacker tries to login with a honeyword, we can check in the red server if it exists. If it exists in the red server and is not a real password indicated by the blue server indices, then we know someone is either trying to impersonate a user or a user gravely misspelled their password. Since the user does not know the honeywords exist, the probability that it is a hacker would be very high and we can flag this.

3.3

A good honeyword for pa$$word5 would probably be a few things. If we wanted to change the special characters ($) into S that would be a good start. Or we could change the special characters into & would also be a good. Then the only thing is the 5 at the end which we can turn into any other digit. Some example honeywords would be: password10 and pa&&word3.

3.4

-itWb!%s453gMoI00286!*mooewTi409##21jUi : looking at this, there is no way a human would ever type or remember this, therefore I do not think it is the password. This is simply way to complicated.

-Blink-123 : This could be a possibility.

-Blink-182 : This is a band, between this and Blink-123 I would choose this as a password. This is because Blink-182 is a well known band and I could see myself using it as a password for convenience and remembering.

4.1

In theory RSA encryption uses some integer M that is raised to the power of *e* and modulo *n.* So that gives us the ciphertext C that is also an integer. In essence, that is what the RSA encryption algorithm is doing. Decryption is raising C to the power of *d* and modulo *n.*

In practice the computational issues come from two parts. The first is the exponentiation to encrypt it. This is solved easily however if you choose the right *e.* The only condition that needs to be satisfied is that *e* must be coprime to *n.* For decryption however, you have to calculate C^*d* mod *n*, which is pretty hard. We cannot choose d, therefore this computation is pretty hard, however there is a clever solution using the Chinese Remainder Theorem. The trick with this is that we can carry out easier modulus with smaller numbers to find the modulus of C^*d* mod *n.*

4.2

Please see attached python file.