CS 306 Homework 2

Theodore Jagodits

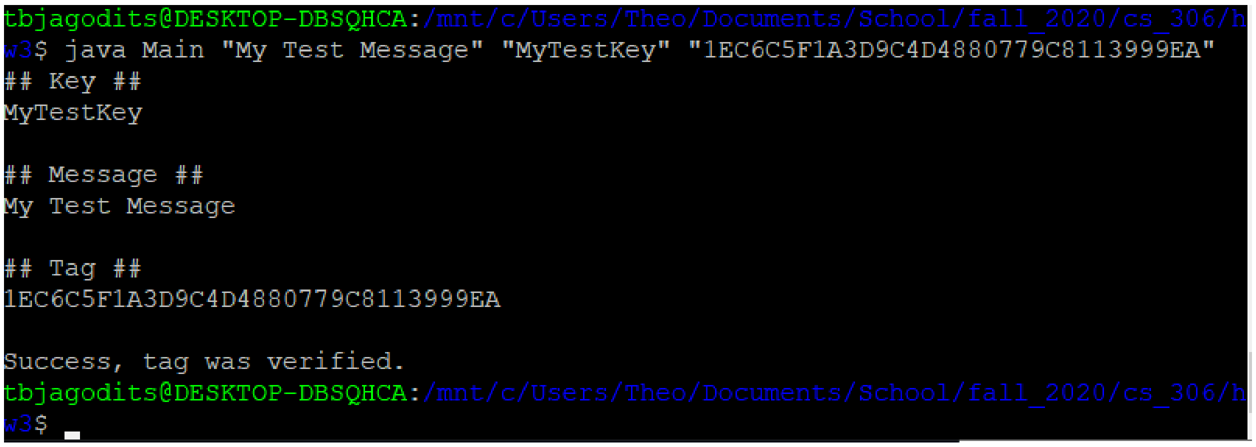
11/19/20

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 since it is easy to implement. I did this because it is very efficient and secure as long as fixed length messages are being sent or multiples of a certain n. This is more efficient than the domain extension MAC as described on the lecture because it only uses one blocksize array that we constantly update with new values rather than appending new things which would increase band-width efficiency.

1.4

According to the slides, this technique is very secure if the blocksize is communicated before and the FPK is secure it should be secure. Since we pad out the messages it will always be a multiple of blocksize() and there will not be any issues with that.

2.1

Case 1:

This is a replay attack. What happens is when Bob requests a new key pair because he has been hacked, Mallory holds on to that request indefinitely (never passing it on to the CA). Then Mallory also replays the old confirmation message of Bob’s current keys. So, Bob never gets his new keys and Eve can look at all of Bob’s messages using the key that she stole. If Bob sees the “new” key pair and realizes that it is the same key as before he can figure out that he has been hacked.

Case 2:

This is more of a man in the middle attack. When Bob requests a new key pair, Mallory holds on to that request again and does not pass it on to the CA. At this point Eve makes up a fake key pair (similar to the one being used) and modifies the old message that Bob had to get his keys. Mallory then replays the modified message. At this point Bob has a fake key pair he believes is the fake new key pair and Eve has the fake new key pair and Bob's current key. Every time Alice sends a message to Bob, Eve intercepts this message and decodes it with Bob's keys. Then she encodes the message with the fake new key pair and passes it along to Bob. This method would be a lot harder for Bob to figure out if he had been hacked.

2.2

This replay attack could not happen if the confirmation message of the old key pair was timestamped because Bob would reject the old message. Since we use metadata of the message we can assume that Eve cannot send a new message from Mallory to create a new fake key for case 2. We assume this message comes from the CA.

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.*

In essence we use modular exponentiation to speed up computation and save memory. By using modulo we restrict the values it can assume between itself and zero. If the exponent is multiplied e times, we can keep that result and multiply that instead of recomputing the entire thing. Then these numbers can be used with decryption as well. These numbers stored as a key pair that we can use to find the multiplicative inverse faster since we already have the numbers given.

4.2

Please see attached python file.