Homework 5: Cryptography II

Due March 24, 11:00 PM

---

1 Homework Overview

This homework is meant to introduce you to the concepts of using Public Key Cryptography, Ciphertext Malleability Attacks and Key Signing in a real-life program. You will get first hand experience in using cryptography libraries in Python to establish a 'secure' communication channel between a client and a bank. For simplicity, communication is done with text files. We've given you encrypt/decrypt function stubs that need to be completed. You will then attack your own implementation to modify the encrypted messages between the client and the bank, and produce unintended results. This will be followed by you writing a fix for this attack by signing your keys as one of the measures.

2 Getting Started

This assignment requires Python3 to successfully run the required libraries you may need, as well as your own code to communicate between the client and the bank.
You should have a Python 3 environment set up and running on your computer. You will also need certain external libraries, which can be installed using pip. We will be providing you with `requirements.txt` as part of the starter files, which contains a majority of the libraries you will need. Install them on your local machine using the command below[1]:

```
$ python3 -m pip install -r requirements.txt
```

2.1 Starter Files

The starter files are available here. The starter files include:

- program.py
- requirements.txt
- Bank.enc_pk
- bank.enc_sk
- accounts.json
- Alice.sig_pk
- Alice.sig_sk
- Eve.sigp_pk
- Eve.sig_pk
- sample.cipher
- sampleSigned.cipher

[1] A simple pip or pip3 install might work. However, on systems with multiple python interpreters (particularly macOS), pip may install for some version of python other than the one that runs when you type python in the terminal

3 Task 1: Public Key Crypto

Send and receive a basic encrypted deposit.

```
$ python3 program.py clientSendDeposit Alice Bob 50 Bank.enc_pk t1ctext
$ python3 program.py bankRecieveDeposit Bank.enc_sk t1ctext t1Recieved

Bank Received Deposit from Alice to Bob for $ 50
Decryption Sucessful!
```

We've given you starter code and an example ciphertext along with some sample key files.
First complete `clientSendDeposit()` in bank that uses the library function `PKEnc` to encrypt the input data correctly.
Then complete `bankRecieveDeposit()` to decrypt the encrypted output of the previous function using library function `PKDec`. Do not hard code keys. We have provided you with helper functions that you will use to achieve the encryption and decryption and even key generation.

Data is serialized to files with pickle. Certain objects, like keys, are converted to bytes first using the included helper functions You should test your program with more than just the set of keys we gave. You can generate more with the commands keygenSig and keygenEnc.

4 Task 2: Malleability Attack

It turns out, the above protocol is vulnerable. When Eve knows who the recipient is, she can steal the money.
Complete the stub function attack() in program.py that accepts the ciphertext output by `clientSendDeposit()` and the name of the intended recipient, and outputs a ciphertext that causes `bankRecieveDeposit()` to pay Eve.
Example:

```
$ program.py attack1 Bank.enc_pk t1ctext t2ctextTweaked

$ program.py bankRecieveDeposit Bank.enc_sk t2ctextTweaked t2Recieved

Bank Received Deposit from Alice to Eve for $50
Decryption Sucessful!
```

Note: do not use the bank's key to simply decrypt and re-encrypt the ciphertext. When we run the grading script, we will not give you the bank's key.

5 Task 3: Key Signing

Can signing the encrypted deposit prevent the attack? We've given you a stub function `ciphertextSampleSigned`. Implement it. You will have to use the signing keys (e.g., `Alice.sig_sk` to sign the encrypted message before and then attach the signature to the, along with the name of the signer (in our program the client is always "Alice.") to the encrypted message. Format of the deserialized ciphertext should be :

```
signedciphertext={"Signature":signature,
"Cipher":Cipher,
"Signer":"Alice"}
```

After that, the signed ciphertext needs to be taken as input by the `bankRecieveSignedDeposit()`. You need to complete this function. As additional input, this function already gets the contents of `accounts.json` mapping account names to signature public keys in a dictionary. You should use the appropriate public key to verify the signature and return the decrypted deposit object. If the signature fails, return none.

```
$ python3 program.py clientSendSignedDeposit Alice Bob 40
    Bank.enc_pk t3ctext Alice.sig_sk

$ python3 program.py bankRecieveDepositSigned
    Bank.enc_sk t3ctext t3Recieved accounts.json

 Bank Received Deposit from Alice to Bob for $50
 Signature Verified, Decryption Successful!
```

Again, we have given you stub methods to complete. Again, we have given you a sample ciphertext `sampleSigned.cipher` Again, you should test your program with more than just the set of keys we gave. You can generate more with the commands keygenSig and keygenEnc.

6 Task 4: Signature Stripping

Signing the ciphertext did not work; Eve was able to bypass the signature check. You notice Eve now has an account at the bank too. Can you reproduce the attack? Complete the stub function attackSignature() that takes the output of clientSendSignedDeposit() from task3 and Eve's secret signature key and strips the original signature and replaces it with Eve's signature.

```
$python3 program.py attack2 Bank.enc_pk  t3ctext
    task4AttackedCiphertext Eve.sig_sk
$python3 program.py bankRecieveDepositSigned Bank.enc_sk
    task4AttackedCiphertext t4Recieved accounts.json
Bank Received Deposit from Alice to Eve for $50
Signature Verified, Decryption Successful!
```

7 Submission

You will submit this homework via ELMS.

Submit the following files.

Required:

1. `program.py`

Submit `<directory-id>.tgz` to ELMS.

Note: Only the latest submission counts.

8 Resources

- Cryptography Documentation here