CS 355 Project

Code-Spec Paper

Project Description

Language and Libraries Used:

- **Language –** Python, ease of variable use
- **Libraries –** hashlib, hmac, os, binascii
  - **Hashlib –** this is a module that implements common interface to many different secure hash and message digest algorithms. It includes algorithms defined in FIPS 180-2, RFC 1321, and more.
  - **Hmac –** this module implements the HMAC algorithm which is described by RFC 2104.
  - **Binascii –** module that helps convert between binary and encoded data.
  - **Os –** module for system dependent functionality.

Glossary for the Repository:

- ***client.py*** – the client implementation for the TCP communication platform

- ***server.py*** – the server implementation for the TCP communication platform
- ***jsonDB.py*** – functions to create and maintain rudimentary json database
- ***password_file_generator.py*** – function to create a 4mb file full of random passwords of random lengths from 12 to 16 characters
- ***digester.py*** – generates a digest from the contents of a file. Used to generate a digest of the password file generated by the previous function
- ***elgamalChat.py* –** implementation of El Gamal for the server client interaction. It is different from the digest comparison function because of the types of messages that are sent. (Chat sends strings, digest is hexadecimal)
- ***elgamalDigestComparison.py* –** El Gamal implementation to demonstrate the goal of the project, comparing two password files without revealing their contents through homomorphic division. This uses hexadecimal inputs as values.
- ***elgamalHomomorphic.py* –** The simplest El Gamal implementation that uses simple integer inputs instead of strings or hexadecimal values to demonstrate homomorphic capabilities of El Gamal.
- ***database.json* –** a rudimentary file that acts as a database for public keys of clients for the server client exchange.

Security Goals:

Our security goal was to create a secure communication channel for Alice and Bob. This would include public key distribution for asymmetric encryption. An encryption scheme to encrypt the traffic. A method for authenticating messages sent through the network. A method to easily compare two files without revealing any of their contents to the other party. Along with a communication network over TCP using socket programming in python the following protocol was designed and implemented:

Public Key Distribution:

To achieve our security goals, there must be a way to distribute public keys for asymmetric encryption. This is implemented with Diffie-Hellman public key exchange protocol that is bundled with the El Gamal encryption scheme. The security of the DH problem is assumed by the discrete logarithm problem that was discussed in class. (s. 46+, Lecture 11)

Encryption Scheme:

For the encryption scheme for secure communication El Gamal was chosen for this protocol. It is implemented with the help of RFC 3526 standard that describes MODP Diffie-Hellman groups for IKE. In the implementation the values for *p* and *g* are fixed to those that are suggested in this standard. The implementation uses the 2048-bit MODP group. The security of El Gamal is based on the DDH problem discussed in class. (s. 18+, Lecture 12)

Authenticity:

To determine the authenticity of messages that are sent through the network and to mitigate any sort of MITM attacks, we decided to use an HMAC that is implemented using the hmac module for python. This HMAC uses the shared secret key from El Gamal encryption scheme to generate a signature for each message that is sent through which gets validated on the receivers’ end. Since only the communicating parties can create the shared secret, this provides authenticity for messages and prevents tampering from any third party.

Communication Network:

The communication network is implemented with socket programming that runs on localhost. It also implements a rudimentary approach for public database where the public keys for each party are stored. The implementation is very limited and can break very easily if not initiated properly. Proper execution will be discussed in a later segment. Between the communication networks, the ciphertext arrays are sent that contain the ciphertext pair from El Gamal (c_1, c_2) and the signature. When this message is received, the recipient party can decrypt this cyphertext to retrieve the original message.

Comparison of the Password Files:

The password files need to be compared between the two parties without revealing the contents of the file. Initially we thought implementing a hashing algorithm like SHA3 would be enough but soon realized that this way the parties would have to compare the digest generated by the SHA3 algorithm by hand. After deliberating for a bit, the homomorphic properties of El Gamal came in handy. Our comparison utilizes these homomorphic properties, namely division, to determine if the files are equivalent by not even revealing the SHA3 digest of the password file. This is achieved through the following: both parties generate their respective SHA3 digest of the password file. Bob encrypts his digest with his own public key and sends it to Alice. Alice then encrypts her own digest with Bob's public key and performs the homomorphic division between the two ciphertexts. She then sends the result to Bob. Bob can now decrypt the result and determine if the files are equivalent by checking if the decrypted message is equal to 1. Otherwise, the files are not equivalent. The same process is repeated for Alice for her to determine equivalence as well.

Testing:

There are several different files that are present in the implementation. Due to problems that arise with converting hexadecimal values to integers and back, and then converting them to strings, it was very time consuming to create a universal method of doing El Gamal, so there are two. One of them works with Unicode characters, which is implemented to demonstrate the chat feature. This can be found in the *elgamalChat.py* file. The other two are *elgamalDigestComparison.py* that demonstrates the capability of homomorphic division comparison with the digest of a test password file. The test password files can be generated with a function present in *password_file_generator.py*. *elgamalHomomorphic.py* implements the same algorithm but more generic inputs are present to demonstrate the idea of homomorphic division. Signature validation is implemented with the chat feature.

Running the Chat Functionality:

To run the chat functionality, we must strictly follow these steps:

- Run *server.py*
- Open two other terminal windows and run *client.py* in each. Make sure to input your nickname for BOTH and only then proceed with the prompt of connecting.
- This allows for the creation of *database.json* where public keys for the two clients are present. Not following these steps will likely end in not being able to retrieve one of the public keys and crashing.
- Do not run more than 2 clients as they will not be able to decrypt the messages and will just prompt with message tampering.
- The algorithm for picking "the other clients" public key just works by picking the one that is not yours. It is very rudimentary and handles only 2 clients.
- If everything is done properly, the two clients can chat between each other while seeing plaintext, as encryption and decryption are running in the background. Additional print statements can be added to inspect private and public keys as well as the cipher arrays and signatures.

Running the Password File Comparison:

This functionality is demonstrated with *elgamalDigestComparison.py.* It includes a main function at the end of the file. There are variables for Alice's' and Bobs values that are taken from a digest f a password file. These can be set to the same file to demonstrate the correctness of homomorphic division, or it can be set to different password files (both can be generated by the password generator function) to demonstrate otherwise.

Conclusion:

The security goals were met with the implementation of the protocol. Diffie-Hellman provides public key distribution for asymmetric encryption. El Gamal provides a sufficient encryption algorithm that is CPA secure. HMAC provides sufficient authentication method for the messages that are sent through the network, and SHA3 provides a collision-resistant hashing algorithm to generate digests for the password files that are later used to determine if the two are equivalent, using El Gamal’s homomorphic properties.