EECS 3482

Assignment 1: Implementation Report

John Ninan

Student ID: 218108522

## Introduction

For this assignment, the given task was to secure a client/server chat application used to send text messages and attached files to users. The specifications of securing this system were to secure the system through the utilization of the Diffie-Hellman key exchange method, password hashing, and appending a MAC message to messages sent through the channel. Additionally, this system was implement based on a provided skeleton of Python code utilizing the PyCryptodome library. This report will delve into the specific design decisions that were chosen, specifically the choice of Diffie-Hellman key exchange parameters, the choice of cypher, and the prevention of tampering of messages using MACs.

## Diffie-Hellman Key Exchange

As was previously stated, the Diffie-Hellman method was used in this specific implementation to ensure secure key exchange. The key-exchange parameters chosen for this system were the 1536-bit MODP group sourced from the RFC 3526 document. This Modular Exponential group was chosen for the implementation because the level of security required was fairly low-level and more importance was placed on the fast and smooth operation of the system as a whole, rather than the most robust security possible. While more secure MODP groups were available for use, this implementation was more focused on a Diffie-Hellman key exchange method that could run as efficiently as possible and on the widest variety of computers with varying levels of compute power. With this focus on efficient operation in mind, it was elected to use the “pow()” method in the use of the calculation of the Diffie-Hellman key, rather than simply writing the mathematical formula in python code. This speeds up our processes through the use of modular exponentiation while simply using the python formula would be much slower as the exponentiation is performed first and the modulus calculation is performed after.

## The ARC4 Stream Cipher

For this implementation, the cipher that was used was ARC4 (Alleged Rivest Cipher 4), which is a stream cipher. A stream cipher is a cipher that encrypts plaintext into ciphertext, byte by byte. The benefits of stream ciphers are that they are much faster than others (namely block ciphers). Additionally, they have a low complexity so they are easy to implement into modern programs and complex hardware is not required at all. They are also serial in nature and because they are a symmetric encryption tool, they are comparatively easier to use than those that deal with public and private keys. As mentioned above, the specific stream cipher that was used was ARC4. ARC4 was used in this implementation because it is the most widely used stream cipher and is robustly supported by PyCryptoDome. Other reasons are because ARC4 is simple to apply and works quickly, even on very large pieces of data. However, ARC4 has some security deficiencies, namely that if it is not used with a strong MAC, its encryption is vulnerable to bit-flipping attacks. Additionally, ARC4 does not provide any authentication functionalities. To compensate for these deficiencies, a MAC is appended to each message in the channel and authentication is conducted through the use of a bcrypt hash, as will be discussed below.

## Preserving Integrity

In order to ensure the integrity of messages, a MAC was appended to each message in the channel. A MAC (Message Authentication Code) is used to ensure that the source and content of a message are consistent and accurate to what is expected. MACs work using three algorithms: a key generation algorithm, a signing algorithm, and a verifying algorithm. First, the key generation algorithm generates a key at random. Then, the signing algorithm sends a tag given a key and a message. Finally, the verifying algorithm verifies that the message is accurate and authentic with the given key and tag. This is appended to each message in the system in order to ensure that the content of the messages is unadulterated.

## Authentication using bcrypt hashing

In the provided skeleton code, the authentication method was incredibly rudimentary and as such was also incredibly unsafe. No modifications were done to the passwords in transit such that any attacker could technically intercept them and they would then have access to the literal password. In order to solve this issue, passwords are hashed using the bcrypt hashing function. This ensure that the password is no longer in plaintext and is instead in a ciphertext that is undecipherable. 12 rounds of salting are done to ensure that the hashed password is completely different from the original password.

## Conclusion

To conclude, this system uses a variety of techniques and types of encryption in order to secure this message service.