1 Introduction

The objective of this project is to develop a secure chat program in C that provides authentication of correspondents, message secrecy through encryption, and message integrity using message authentication codes (MACs). Given that the program processes formatted input from a network, it also emphasizes the importance of software security.

The key goals for this project are:

- Gaining familiarity with cryptographic libraries such as OpenSSL.
- Designing a robust cryptographic protocol.
- Understanding various issues in network programming.
- Avoiding common software security issues.

The project skeleton includes basic chat functionality, such as listening for connections or making one with another host. Our task is to extend this functionality by:

1. Writing a handshake protocol to set up ephemeral keys, ensuring perfect forward secrecy.
2. Implementing mutual authentication using public key cryptography.
3. Encrypting and tagging each message with a MAC, and taking measures to prevent replay attacks.

This document details the implementation of the Triple Diffie-Hellman (3DH) key exchange protocol, which is a crucial part of establishing secure communication between parties. It describes the functionality and security properties of the 3DH protocol, as well as how it is integrated into our chat program.

2 Triple Diffie-Hellman (3DH) Key Exchange Protocol

Triple Diffie-Hellman (3DH) is a cryptographic key exchange protocol that extends the classic Diffie-Hellman key exchange by involving three separate Diffie-Hellman key exchanges to provide stronger security properties. This protocol is especially useful in securing communication between two parties, ensuring the confidentiality and integrity of the exchanged messages, and providing mutual authentication.

The 3DH protocol involves the following steps:

1. **Generation of Long-term and Ephemeral Key Pairs**: Each party generates a long-term key pair (private and public keys) and an ephemeral key pair for each session. The long-term keys are used for authenticating the parties, while the ephemeral keys are used for ensuring forward secrecy.
2. **Exchange of Public Keys**: Each party shares their public keys (both long-term and ephemeral) with the other party.
3. **Computation of Shared Secrets**: Each party computes three shared secrets using their private keys and the received public keys:

  - One shared secret using their long-term private key and the other party's ephemeral public key.
  - Another shared secret using their ephemeral private key and the other party's long-term public key.
  - A third shared secret using their ephemeral private key and the other party's ephemeral public key.

4. **Derivation of Session Key**: These three shared secrets are then combined (e.g., using a hash function) to derive the final session key, which is used for encrypting and decrypting messages.

3 Perfect Forward Secrecy in 3DH

Perfect Forward Secrecy (PFS) ensures that session keys will not be compromised even if long-term keys are. This property is achieved in the 3DH protocol by using ephemeral key pairs that are unique to each session.

3.1 Key Generation

In 3DH, each party generates both long-term and ephemeral key pairs. These keys are generated using the following steps:

- **Long-term Key Pairs**:
  - Each party selects a private key: a for Alice and b for Bob.
  - They compute their respective public keys using the generator g and a large prime p:

$$a = g^a \mod p \quad \text{and} \quad b = g^b \mod p.$$

- **Ephemeral Key Pairs**:
  - For each session, Alice and Bob generate ephemeral private keys x and y, respectively.
  - They compute their ephemeral public keys:

$$x = g^x \mod p \quad \text{and} \quad y = g^y \mod p.$$

3.2 Computation of Shared Secrets

The 3DH protocol involves the following computations:

- Alice ($\mathrm{a} = g^a \mod p$) and Bob ($\mathrm{b} = g^b \mod p$) exchange ephemeral keys x and y, where $x = g^x \mod p$ and $y = g^y \mod p$.
- Alice computes the shared secret $\mathrm{K} = \mathrm{KDF}(\mathrm{Y}^a, \mathrm{B}^x, \mathrm{Y}^a, \mathrm{X}, \mathrm{Y}, \mathrm{A}, \mathrm{B})$.
- Bob computes the shared secret $\mathrm{K} = \mathrm{KDF}(\mathrm{X}^y, \mathrm{A}^y, \mathrm{X}^b, \mathrm{X}, \mathrm{Y}, \mathrm{A}, \mathrm{B})$.

These shared secrets are combined to derive the final session key, ensuring that the session key is unique and not derivable from long-term keys.

4 Assumptions and Security Considerations

4.1 Assumptions

- It is assumed that the communicating parties already have access to each other's public keys.
- The public keys are stored in publicly accessible text files and read by the parties during the key exchange process.

4.2 Security Claims

The implementation of the 3DH protocol aims to ensure the following security properties:

- **Confidentiality**: The session key derived from the 3DH exchange is used to encrypt and decrypt messages, ensuring that the communication remains confidential.
- **Integrity**: The use of cryptographic hashing and the 3DH exchange ensures that any tampering with the keys or messages can be detected.
- **Mutual Authentication**: Both parties authenticate each other using their long-term public keys, ensuring that they are communicating with the intended party.

4.3 Handling Malicious Communicating Parties

In the presence of a malicious communicating party, the following considerations are made:

- If a party runs a modified, malicious version of the chat program, it could attempt to impersonate the other party or intercept messages. The 3DH protocol's mutual authentication helps mitigate this risk.
- The worst-case scenario involves a man-in-the-middle attack, where the adversary intercepts and replaces public keys. The use of authenticated public key distribution (e.g., through a trusted third party or pre-shared keys) can help prevent such attacks.

5 Implementation in Code

Below is a discussion of how the 3DH key exchange is implemented in the provided code:

5.1 Key Generation and Storage

The code uses the `dhGen` function to generate long-term and ephemeral key pairs for both the listener and connector instances. These keys are stored in public text files to facilitate the exchange:

```
NEWZ(b); // Private long-term key for listener
NEWZ(B); // Public long-term key for listener
dhGen(b, B);
```

```
// Store public key B
char* B_str = mpz_get_str(NULL, 10, B);
writeFile("B.txt", B_str);

// Generate ephemeral keys for listener
NEWZ(y);
NEWZ(Y);
dhGen(y, Y);

// Store ephemeral key Y
char* Y_str = mpz_get_str(NULL, 10, Y);
writeFile("Y.txt", Y_str);
```

Similarly, the connector generates its own long-term and ephemeral key pairs:

```
NEWZ(a); // Private long-term key for connector
NEWZ(A); // Public long-term key for connector
dhGen(a, A);

// Store public key A
char* A_str = mpz_get_str(NULL, 10, A);
writeFile("A.txt", A_str);

// Generate ephemeral keys for connector
NEWZ(x);
NEWZ(X);
dhGen(x, X);

// Store ephemeral key X
char* X_str = mpz_get_str(NULL, 10, X);
writeFile("X.txt", X_str);
```

5.2 Key Exchange

The listener reads the public keys A and X from the connector, and the connector reads the public keys B and Y from the listener:

```
// Listener reads A and X
char* A_recv = readFile("A.txt");
mpz_t A;
mpz_init(A);
```

```
mpz_set_str(A, A_recv, 10);
free(A_recv);

char* X_recv = readFile("X.txt");
mpz_t X;
mpz_init(X);
mpz_set_str(X, X_recv, 10);
free(X_recv);

// Connector reads B and Y
char* B_recv = readFile("B.txt");
mpz_t B;
mpz_init(B);
mpz_set_str(B, B_recv, 10);
free(B_recv);

char* Y_recv = readFile("Y.txt");
mpz_t Y;
mpz_init(Y);
mpz_set_str(Y, Y_recv, 10);
free(Y_recv);
```

5.3 Computation of Shared Secrets and Session Key

Both the listener and the connector compute the shared secrets and derive the session key using the `dh3Final` function:

```
// Listener computes shared secrets and session key
unsigned char keyBuf[PATH_MAX];
size_t bufLen = PATH_MAX;
dh3Final(b, B, y, Y, A, X, keyBuf, bufLen);
printf("Listener's session key: %s\n", keyBuf);

// Connector computes shared secrets and session key
unsigned char keyBuf[PATH_MAX];
size_t bufLen = PATH_MAX;
dh3Final(a, A, x, X, B, Y, keyBuf, bufLen);
printf("Connector's session key: %s\n", keyBuf);
```

6 Assumptions and Security Considerations

6.1 Assumptions

- It is assumed that the communicating parties already have access to each other's public keys.
- The public keys are stored in publicly accessible text files and read by the parties during the key exchange process.

6.2 Security Claims

The implementation of the 3DH protocol aims to ensure the following security properties:

- **Confidentiality**: The session key derived from the 3DH exchange is used to encrypt and decrypt messages, ensuring that the communication remains confidential.
- **Integrity**: The use of cryptographic hashing and the 3DH exchange ensures that any tampering with the keys or messages can be detected.
- **Mutual Authentication**: Both parties authenticate each other using their long-term public keys, ensuring that they are communicating with the intended party.

6.3 Handling Malicious Communicating Parties

In the presence of a malicious communicating party, the following considerations are made:

- If a party runs a modified, malicious version of the chat program, it could attempt to impersonate the other party or intercept messages. The 3DH protocol's mutual authentication helps mitigate this risk.
- The worst-case scenario involves a man-in-the-middle attack, where the adversary intercepts and replaces public keys. The use of authenticated public key distribution (e.g., through a trusted third party or pre-shared keys) can help prevent such attacks.

7 Conclusion

The Triple Diffie-Hellman key exchange protocol enhances the security of communications by combining multiple Diffie-Hellman exchanges, providing robust protection against various attacks and ensuring the confidentiality, integrity, and authenticity of messages exchanged between parties.