TedChat - Phase 1

Andrew Kao
Cory Knapp

September 23, 2016

1 Introduction

This document describes the design and security properties of TedChat, a secure desktop chat application. TedChat is written by TeamTed, which is composed of Andrew Kao and Cory Knapp, as an exercise in implementing secure protocols. As a result, the product emphasises security over all else.

2 Application Properties

TedChat is designed to securely send messages composed of text between two client applications.

3 Client Design Overview

The client application is to be designed in c++ using the Qt framework

In figure 3 we see a mock up of the main client window and a corresponding table describing it's features in table 1. Likewise, a mock up of the dropdown menu can be seen in 3 and a corresponding table describing it's features in table 2.

Not pictured in a figure is the preferences interface, which as of now, consists as a single field for entering the user's ID.

Figure 1: Client Window

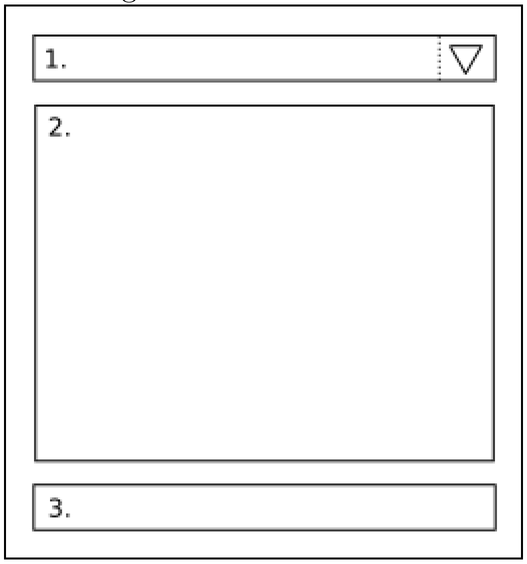


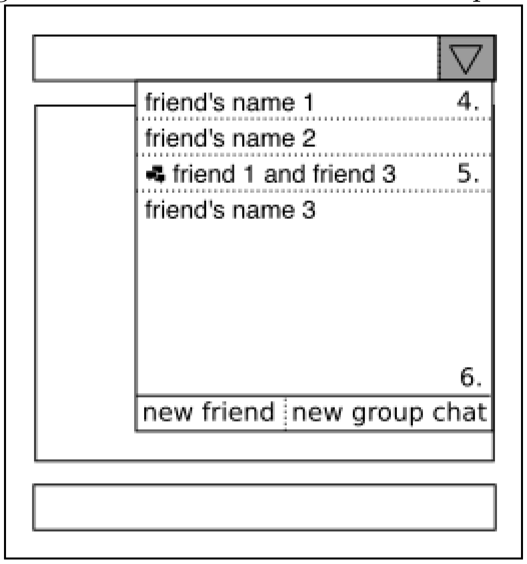
Table 1: Description of Client Window Features

1	Text field and drop down menu to display and change the active chat channel
2	Text field to show recent chat history
3	Text field to compose new messages

Table 2: Description of Client Window Drop Menu

4	Recent friends are stored in a list in the top of the menu
5	Group chats are shown as a list of names and denoted with an icon
6	the options for adding new friends, or iniilzing a new group chat appear at the bottom of the menu

Figure 2: Conversation Selection Dropdown



4 RESTful Server Design Overview

The server will be based on a SQL database. We can neglect to include a table to track user information; from the server's perspective, each user is represented by a single identification number. However, it is desirable (though not strictly necessary) to insure that users are assigned unique identification numbers. This will be implemented using a table consisting of a one column and a one row. We'll track the ID of the user most recently registered, and increment it when a new user ID is requested.

The message table will consist of four fields:

1. **Sender ID** - The self reported ID number of the user claiming to be the message sender.

2. **Recipient ID** - The ID number of the message's intended recipient.

3. **Timestamp** - Date and time that the message was posted to the server. This field is not provided by the client; the timestamp is generated by the server.

4. **Ciphertext** - Encrypted message, including message meta data.

5 Assets/Stakeholders

Because of the relatively limited scope of the product, there are few assets to protect. The primary assets are the messages sent between users. The primary goal is to protect the messages confidentiality and integrity. Secondly, we must maintain control of the server to ensure the availability of messages to the user. Thus, control of the server is also an Asset that requires protection.

The primary stakeholder in this scenario is the user who expects their messages to be sent securely, and without interruption.

6 Adversarial Model

The primary concern is to guard against a man in the middle attack. It is imperative that an attacker who is able to intercept any message find that message useless. By useless, we mean that the attacker should be unable to discern any meaning from the cipher text, nor should they be able to construct meaningful alternate cipher text as a substitute.

Notice that the server does no authentication of users at any time. User 1 may, with ill intentions, request a messages from the server which are intended for user 2. However, without user 2's private key, this message will be unreadable.

A second troubling scenario would be the case where a user sends a message posing as another user. Imagine that user 1, 2 and 3 are all friends, and of course each user has the other two's public keys. It would be easy for user 1 to post a message to the server with user 2 as the reported sender and user 3 as the recipient. However, user 2 and 3 generated a shared secret key through a Diffie-Hellman key transfer when they first connected. Any genuine message from user two will contain in it's metadata the shared secret, and others can be rejected as inauthentic.

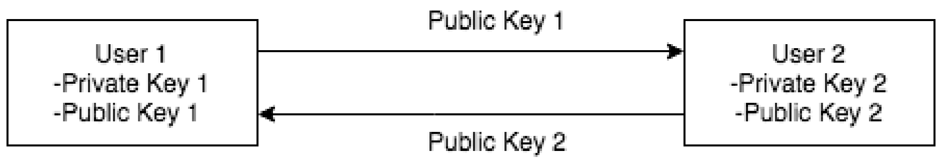
7 Previous work

1. WhatsApp Whitepaper
2. OpenSSL Documentation

3. PGP Documentation
4. TLS
5. LetsEncrypt
6. QT Documentation

7.1 Security Model

Figure 3: Handshake



1. Initialization
   (a) When user installs application, the program will generate their Curve25519 public-private key pair.
2. Connection to server
   (a) User registration
      i. Server will generate a UUID and assign it to user
3. User-user handshake
   (a) Passing Public Key (see figure 3)
      i. Users will physically pass their public key and user ID to users they wish to chat with.
4. Creating the chat
   (a) Shared secret (see figure 7)

    i. Both users will calculate a shared secret using Elliptic Curve Diffie-Hellman (ECDH) on their public-private key pairs.

(b) Generate session keys (see figure 6)

    i. Both users generate their respective initial root keys from the shared secret using HKDF.
    ii. Both users generate their respective initial chain keys from their respective root keys using HKDF.
    iii. Both users generate their respective initial message keys from their respective chain keys using HMAC-SHA256.

5. Encryption/Decryption

   (a) Encryption (see figure 5)

       i. When a user sends a message, a message key will be generated from the chain key The user generates an ephemeral Curve25519 public key.
       ii. The chain key will ratchet up by one.
       iii. The user encrypts the message key with the public key of the recipient with RSA.
       iv. The user appends the encrypted message key to the encrypted message with PGP.
       v. The user concatenates the encrypted message, encrypted key, and ephemeral key together and sends the whole string to the recipient.

   (b) Decryption (see figure 4)

       i. The user uses their private key to decrypt the message key.
       ii. The user uses the message key to decrypt the message.

6. Updating Session Keys (see figure 8)

   (a) A user will use the same root key until a response is received.
   (b) When a response is received, the user will generate a new ephemeral public key and generate an ephemeral secret by using ECDH on his new ephemeral public key and the other users ephemeral public key that was advertised in the response.
   (c) The user will then generate new root and chain keys from the ephemeral secret.

8 Analysis

8.1 Message Security

Because each message is encrypted by a key that is also encrypted with the recipients public key, an adversary cannot know the message key (and thus cannot decrypt the message) without knowing the recipients private key.

8.2 Forward Secrecy

Our system ensures that no message keys are recycled, therefore an adversary would not be able to use a compromised message key to decrypt past messages.

8.3 Server

OpenSSL/TLS comes with client certificate authentication for our server, so User 1 would not be able to make requests to the server for User 2's messages. This also means User 1 would not be able to send a message to User 3 posing as User 2. In the event that an adversary were to successfully pose as User 2 or for some reason SSL/TLS authentication is compromised, the impostor will not be able to decrypt User 2's messages without User 2's private key, and he does not know the shared secret of the actual User 2 and User 3, so User 3 would not accept any messages from the impostor.

Figure 4: Message Decryption

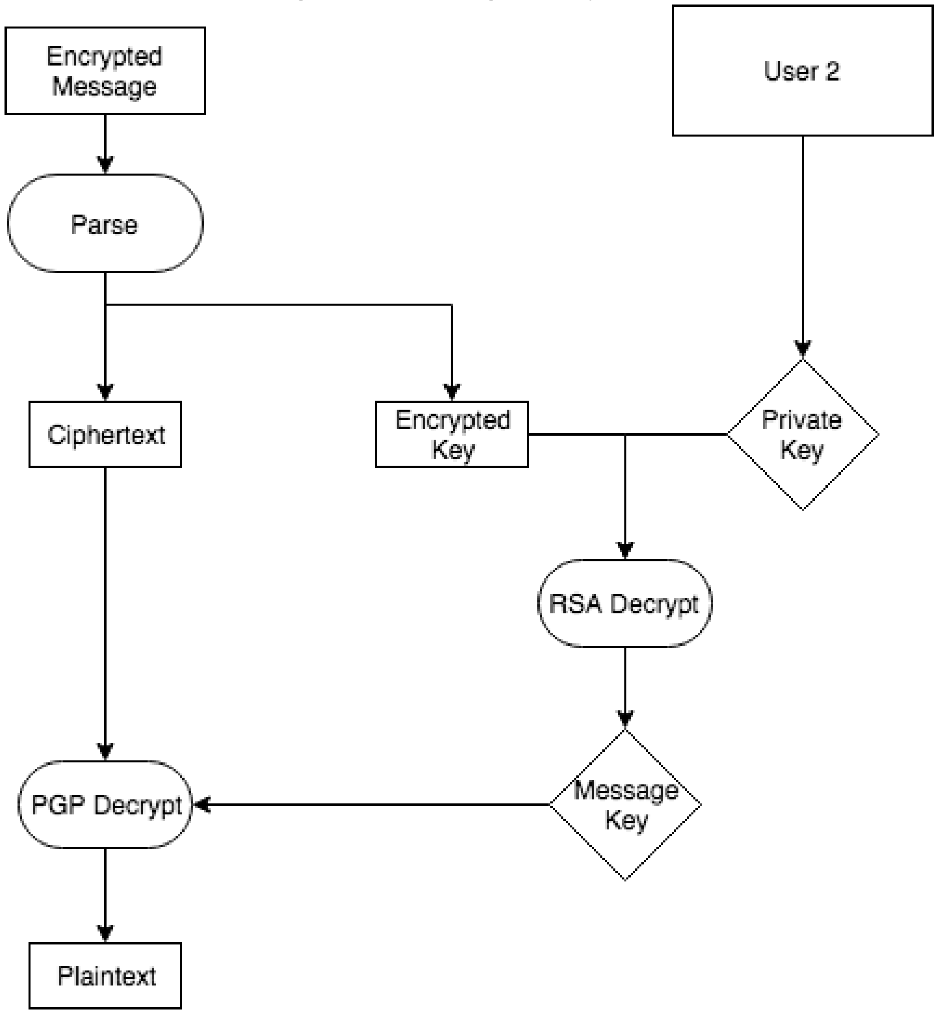


Figure 5: Message Encryption

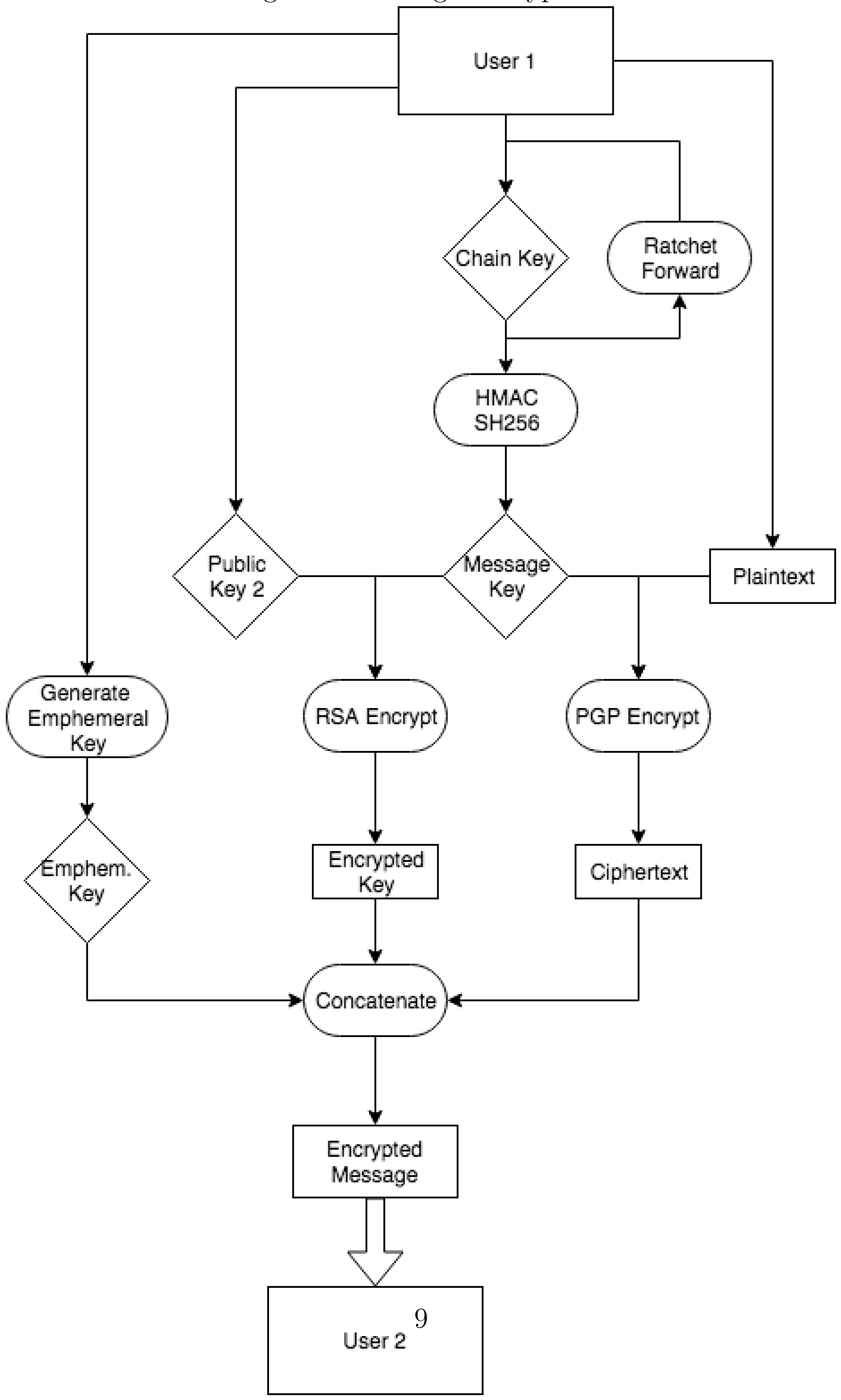


Figure 6: Session Keys

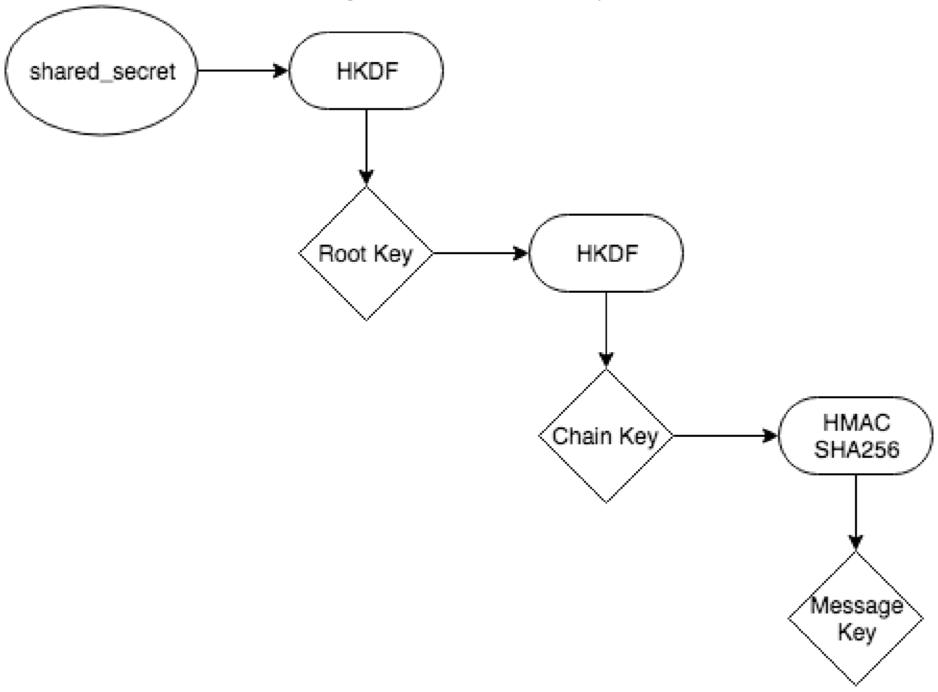


Figure 7: Shared Secret

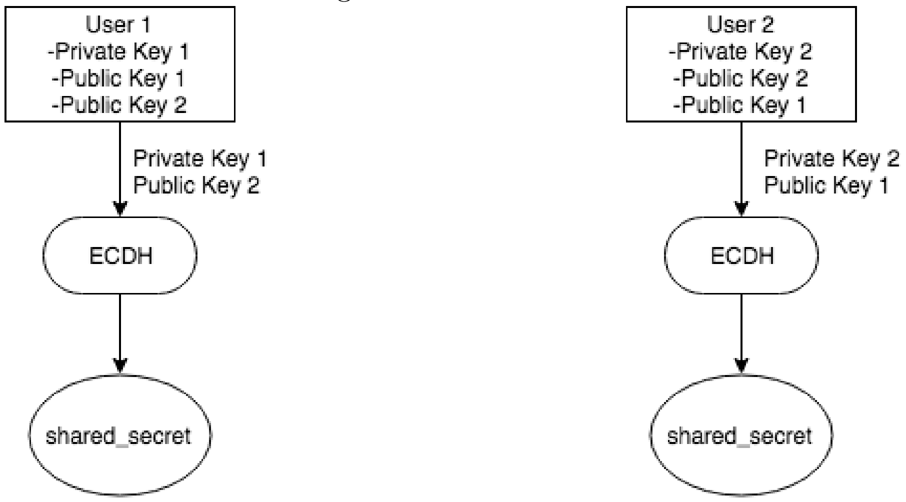
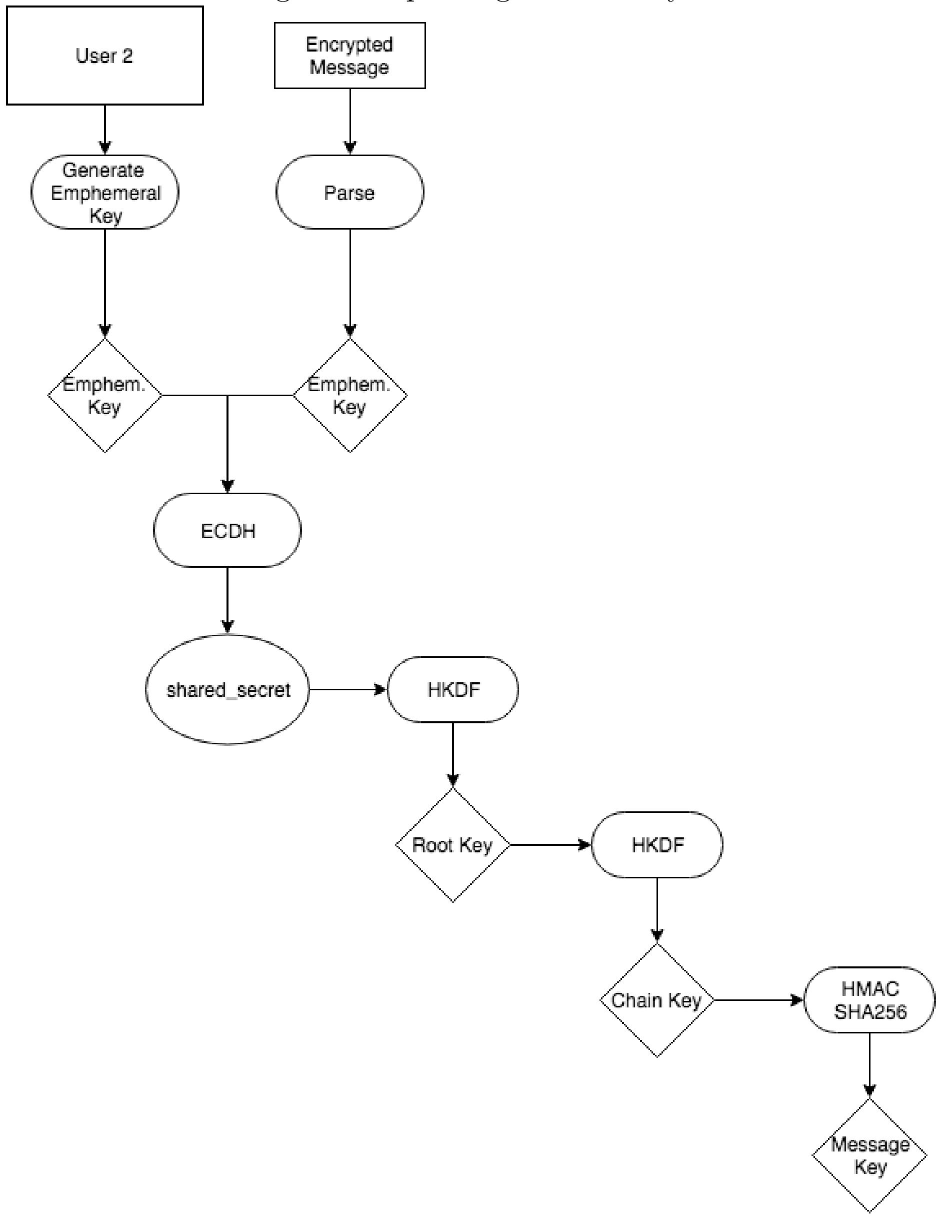


Figure 8: Updating Session Keys


User 2
Encrypted Message
Generate Emphemeral Key
Parse
Emphem. Key
Emphem. Key
ECDH
shared_secret
HKDF
Root Key
HKDF
Chain Key
HMAC SHA256
Message Key