SDD HSC Major Work - Messenger

Max Hamilton
35419837

Contents

1 Terminology

Throughout this report, the client who requests the application be built will be referred to as the customer. "The client" will refer to the application which "users of the client" use to send and receive messages, and "the server" will refer to the application which centrally stores messages, administrated by "users of the client", as in the client-server model.

2 Design Brief

2.1 Client and target audience

The target audience of this application is anyone who needs to securely communicate, without trusting the server they are communicating on. This could include governments, corporations cognizant of the possibility of hacking, or anyone renting servers or using a cloud provider. The target audience is one who does not mind giving up a small amount of convenience for security.

2.2 Definition of the problem

The customer requires a secure messaging application. They don't trust existing infrastructure, and existing messaging applications. They need all the code to be open source, and to be able to run their own servers. However, they don't fully trust their own sysadmins with the data, are conscious about the possibility they could be hacked, or might outsource the server hosting to a cloud provider, so need the server to know as little about the messages as possible. Ideally, an attacker could hack their system, control the server, and the worst thing they could do would be shut it down, with 0 possibility for sabotage and 0 possibility of accessing any data or metadata. The messaging service must be similar to email, with the ability to send messages to multiple people, to respond to specific messages, and to CC and BCC different accounts to messages. Users should only be able to read messages intended for them, and there should be a way for the recipient to authenticate a message came from a specific user, without the server knowing the author. The customer requires 2 applications, one client, and one server. The client application must have a GUI interface, but the server may have a CLI interface, as it will be run by system administrators and not ordinary users.

2.3 Description of the solution

2.3.1 Initialisation

The user of the server requests the server initialise a new user. The server will generate a random one-time initialisation code, and display it on-screen. The user of the client is informed of the IP address or domain name of the server by the user of the server, and inputs it to the client. A secure connection is established between the client and server via TLS over TCP/IP, and the client stores the server's IP address or domain name if dynamic DNS is used. The user will receive the initialisation code generated by the server for this initialisation, communicated to them via the user of the server, through existing contact channels. The client will then send that code back over the connection, to authenticate themselves. The client will generate a private and public key-pair, and store both. The user of the client will then input a username. The username and the public key are all sent to the server, which stores them together as an account. It is heavily recommended that the initialisation be done on a device used only for initialisations, or via a proxy, VPN, or over Tor, as otherwise the server could connect the username and public key to an IP address, which can be used to find the author or recipient of messages if they are not uploaded or downloaded over a proxy, VPN, or Tor.

Table 1: Security considerations

Possible threat	**Prevention**
Fake clients sending initialisation requests.	The server generates the initialisation code on request from the user of the server, so an attempt to initialise without prior planning will be ignored.
Code being sniffed on network and used to send a false request.	The code is sent over a secure TLS connection.
Username and password being sniffed or altered.	The username and password are sent over a secure TLS connection.
Initialisation code being sniffed through the initial communication.	No prevention. The system administrator must be careful to ensure the correct person receives the code, and no one else does. The code must be conveyed through another secure message channel, digital or physical.
Server building profiles of usernames and public keys to IP addresses.	Initialise via an IP not used for further message sending or receiving, such as on company device used for every initialisation, or via a proxy, VPN, or over Tor.

2.3.2 Access control and authentication

Users of the client do not log in in the traditional sense, the sole factor controlling their account is their private key. That key can be put onto a USB stick and deleted from the original computer for physical control, or left as a file on the computer. At all times, the server broadcasts 2 things using UDP. First, periodically, every x seconds or y messages, the server will generate a code, and encrypt it for multiple recipients with the public keys of every registered account. The encrypted code is then broadcasted, along with the hash of the original code. The server also broadcasts a list of public keys, and the usernames associated with them, encrypted many times in the same way as before, as well as the hash of the unencrypted list. It can be detected how many recipients a message was encrypted for, so the server would have a maximum number of clients, and add a certain amount of 'junk recipients', in order to prevent attackers from detecting the number of clients a server has. The possession of a private key imparts its user with 4 privileges: To decrypt the sending code broadcast by the server, allowing it to send messages to the server which will not be discarded, to decrypt the list of usernames and public keys broadcast by the server, allowing it to encrypt messages, and match public keys to username, to sign messages sent, and to decrypt messages received.

Table 2: Security considerations

Possible threat	**Prevention**
Hostile actors sniffing the broadcast sending code or list of public keys and usernames.	Both messages are sent encrypted, and the amount of recipients is obfuscated.
User leaks the sending code, enabling a hostile actor to overwhelm the server with spam	The sending code is refreshed after a certain time period has passed or a certain number of messages have been sent.
User leaks their private key	No prevention. If a private key is leaked, hostile actors can impersonate the owner, read the owner's messages, read the sending codes, and read the list of usernames and public keys. Users must not leak their private keys.
Server creates different sending codes or public key lists for each user, to track the author of messages or deceive users.	The server must send a hash of the plaintext version of everything they encrypt, and clients can check that the message they decrypt matches the hash. If the server tries to send different codes to different people, the deception can be detected by the non-matching of the hash.

2.3.3 Sending Messages

To send a message, the user of the client composes it, attaches any files, which are just concatenated onto the end of the message, inputs the usernames of any recipients, and requests the client to send. If the client does not want the server to know the length of the message, all messages padded to a previously arranged maximum size. The client then adds a header composing of the username of the user of the client, and a digital signature using the client's stored private key. The client then encrypts the message for multiple recipients, including itself, and every intended recipient. If the client does not want the server to know the number of recipients, they can generate junk recipients and randomly insert them into the list. The client then reads and decrypts the sending code broadcast by the server, and sends this header plus encrypted message and original header to the server, along with the sending code it received earlier. If uploading messages is done directly, servers can build a profile of which IP addresses sent which message. This can be prevented by sending messages via a proxy, VPN, or over Tor. The server verifies the code, and stores the message, as well as the time at which it was received. If the server receives a message with an incorrect code, it discards it, and logs the IP for potential blocking. The server refreshes the code being broadcast after a certain period of time, or after a certain number of messages have been sent.

Table 3: Security considerations

Possible threat	**Prevention**
Messages being forged or altered.	The messages are digitally signed by the author.
Hostile server	The server could potentially know: The timing of messages, the IP address which authored certain messages, the length of the message, and the number of recipients. To prevent the server knowing the IP address of the author, which could be used to build a profile, clients can upload via a proxy, VPN, or over Tor. By padding the message to an agreed upon maximum length, the length can be hidden, although the server would still know an upper bound on the message length. By adding junk keys, the number of recipients can be hidden, although the server would still know an upper bound on the number of recipients. The timing of messages can be concealed from the server by the sending of junk messages at random intervals, although that uses a lot of server storage space and the server still knows an upper bound on the number of messages sent.
Server being overwhelmed by fake messages from a hostile actor	Clients must log in with a username and password to establish a connection with the server and send messages.
Messages being sniffed	Messages are sent over a TLS connection, and additionally encrypted. Even if an attacker could decrypt the TLS encryption, they would know as much as the server, which is very little.

2.3.4 Receiving messages

Each client will store the most recent message that it has received, and periodically poll via TLS over TCP/IP the server for any messages sent past that time. If there are, the server will send the earliest message past the received time, and a bit indicating whether there are more messages available. If there are, the client will continue polling until there are no more messages available. When a client polls for a message, it sends the timestamp it wants the messages past, and appends the SR code. The server can then verify the code before sending a small snippet which the client can use to determine whether the message is intended for them. If it isn't, they move on, but if it is, they request the full message, with the same code. The code is again verified, the encrypted message is sent, and the client then locally decrypts it to display to the user. The client will keep a local log of past message titles, senders, timestamps, and a list of unique words, stored encrypted on the disk, to enable the user of the client to search and browse past messages. The full message is downloaded on demand by sending a request with the timestamp the message was sent to the server. If the client does not use a proxy, the server can build a profile of which IP addresses download which messages, even without names to match. To prevent this, clients can use a proxy or VPN, or connect over Tor.

Table 4: Security considerations

Possible threat	**Prevention**
Hostile actors polling the server.	Requests for messages must include the code, and the sent messages are completely encrypted.
Hostile actors sniffing the transmitted messages.	Messages are sent over TLS, with another layer of encryption.
Hostile clients attempting to read others messages.	Messages are encrypted with the private keys of the recipients, and can only be decrypted with their private keys. Hostile clients could download encrypted messages, but not decrypt them. They would know only what the server knows, which is very little.
Server sending fraudulent messages.	All messages are signed with the private key of the author, so it is impossible to fake or tamper with them.
Server not sending messages.	Users will be able to see if their message is not uploaded, as users must download their own messages to read them, so if a server is not sending certain messages, it can be quickly detected.
Server detecting who requests messages and building a profile.	While the server can see who requests messages, all they know is an IP address, so profile building can be circumvented with the use of a proxy, VPN, or connecting via Tor.

2.3.5 Storage

The server will store, in an SQL database:

- Encrypted messages.
- Timestamps of encrypted messages.
- List of usernames, and the public keys they are associated with.

3 Planning the software solution

3.1 Algorithms

3.2 Prototype

3.3 Data Dictionary

3.4 Modelling tools

3.5 Programming language

3.6 Logbook

4 Building the software solution

4.1 Source code

4.2 Feedback

5 Documenting to meet installation of software solution

5.1 Installation guide

5.2 User manual

5.3 Testing methods

6 Evaluate and suggest modifications

6.1 Evaluation

6.2 Logbook