A secure client-server mobile chat application implementing elliptic curve integrated encryption system (ECIES) and other security features.

Daniel Furnivall

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

1st April 2022

Abstract

abstract goes here

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name:	____________	Signature:	____________

Acknowledgements

acknowledgements go here

Contents

Chapter 1: Introduction

In an increasingly digital world, we are constantly producing data (and, of course, corresponding metadata). It's estimated that humanity created somewhere in the order of 2.5 exabytes of digital data per day in 2018[1] and the growth of digital data creation by individuals is constantly accelerating.

As storage costs decrease over time[19], economic and political incentives have begun to develop for nation state actors and major organisations to develop profiling systems using large-scale data collection and mining (the much vaunted "Big Data"). These systems are already being used for targeted advertising[4], market segmentation[13], criminal investigations[20] and sentencing[15]. In the political sphere, these models have been used for increasingly effective traditional campaigning, [7] as well as (alleged) psychological manipulation[2] and disinformation[17] campaigns.

The question of whether the average individual is able to enjoy the fruits of such data collection is less clear. Do these large actors have the best interests of the subjects of their data in mind? If not, methods for obfuscating or hiding sensitive data become valuable considerations.

In a pre-digital world, an individual could avoid eavesdropping or data collection by malicious entities by simply speaking in a hushed voice, shredding documents, moving quietly or looking over his or her shoulder. In the current landscape, it is much more difficult to avoid surveillance without eschewing technology entirely. Our mobile phones are constantly communicating our triangulated locations, and even if our web traffic is encrypted by default, browser or ISP metadata still contains useful profiling information.

Secure messaging applications aim to allow individuals to communicate with each other individuals around the globe while avoiding the potential for eavesdropping. In theory, this means the user can enjoy the benefits of a globally connected world while preserving their privacy. However, in practice there are of course many implementation difficulties to be considered.

1.1 Why are secure chat applications needed?

Secure messaging applications provide a means of communication between individuals or groups across a network of some kind. This can take the form of text, audio or video messaging, document or file sharing.

To begin to understand the users of secure chat applications, two important questions need to be answered:

1. Who desires secure messaging?
2. Who are they aiming to protect their data from?

There are many categories of potential users of such applications, and from wildly different settings. These can be mundane and innocuous, such as the organisation of a surprise party for a friend or family member.

However, another group of potential users are those who seek to hide criminal behaviour from law enforcement organisations. A high-profile example of this would be the EncroChat network of encrypted phones, predominantly used by organised crime, which was unveiled after a Europe-wide infiltration and investigation of the network by law enforcement groups (leading to several thousand arrests) [16].

Another group who may wish to evade police or law enforcement are political dissidents. In what has become known as the "Million Dollar Dissident" [9] case, a dissident in the UAE, Ahmed Mansoor, was targeted by the NSO group (an Israeli cybersecurity firm which produces spyware for government use) and subsequently had his passport confiscated as well as being beaten, having his car stolen and finally imprisoned by the UAE authorities within a week of posting anti-government posts online [12]. There are many parallels here with other groups who benefit from secure messaging - whistleblowers sharing information with journalists, and police informants who need to share data secretly with law enforcement.

In reality, there are plenty of reasons for *everyone* to use secure messaging such as protecting data in case of device theft, avoiding embarrassment, or limiting exposure to blackmail or government surveillance. Continuing the comparison to real-world communication - when speaking out loud, people don't tend to shout all the time, and tend to consciously limit and monitor who is listening to a conversation.

It should be clear that although there are many categories of potential secure messaging users, there are also many potential adversaries for these kinds of platforms, which means the development of such services is a complex undertaking. There are major tradeoffs which need to be made between usability and data security - for example, how can we preserve security of messages while also storing them on a mobile device?

1.2 Objectives

The primary development objective of this project was to develop an Android mobile application (and corresponding server) that used complex security features including an end-to-end encryption solution that uses both asymmetric (Elliptic Curve Diffie-Hellman) and symmetric (AES) encryption approaches, pseudonymous identity and self-destructing messages.

During the development journey of this application, the intrinsic motivation was to come to a greater understanding of the complexities, assumptions and tradeoffs involved in creating these kinds of platforms.

Chapter 2: Requirements and Analysis

¡Discussion should go here about Unger et al's paper and the three key challenges (Trust establishment, conversation security and transport privacy)[18]

2.1 Existing applications in this field

2.1.1 Telegram

2.1.2 Whatsapp

2.1.3 Signal

2.1.4 Comparison of existing applications

DRAFT: Insert table here comparing signal, whatsapp and telegram.

Possible fields: Encryption algorithm, Degree to which Open sourced, Self-destructing message implementation, group chats, video/audio chat, pseudonyms

2.2 Issues

2.2.1 Closed source

2.2.2 Tradeoffs between security and usability features

2.2.3 Nation state control

2.3 Features

2.3.1 Self-destructing messages

2.3.2 End-to-end encryption

DRAFT - potentially worth mentioning possible attacks on crypto chats and metadata e.g. message length attacks [3]

Chapter 3: Design and Implementation

3.1 Architectural Design

Some of the key primary architectural considerations on this project are highlighted below:

1. Client-side encryption to allow messages to flow through the (untrusted) server.
2. Client-side self-destructing messages according to client-defined storage duration parameter.
3. Persistence of user identity and handling of disconnection/reconnection events.

The figure below represents a very high level view of the system architecture, which gives an overarching perspective of how the system fits together and some of the technological choices made. However, it does not give a comprehensive picture of the complexity of the overall system.

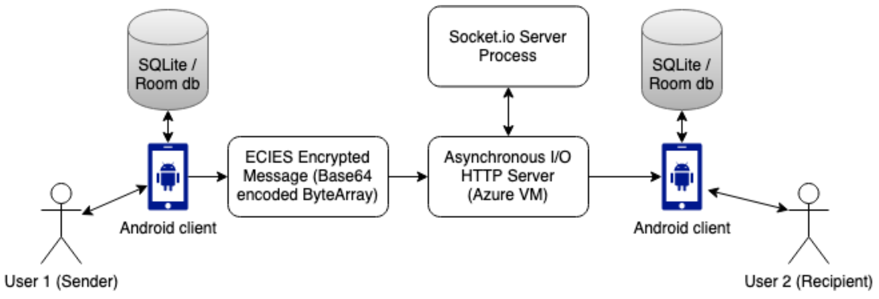

Figure 3.1: High level architecture of the system

3.2 WebSockets vs REST API

One of the major decisions required for this project was which transport protocol to use for propagating messages through the system. The two major options considered were REST APIs[11] and WebSockets[5].

REST (Representational State Transfer) is a set of design principles which allow us to develop web services which utilise a central server and is based on the twin concepts of request and response. A client will send a request to a server and receive a response based on the content of the request. REST APIs are heavily used across many industries and mostly (though not exclusively) use the HTTP protocol to communicate between client and server. The primary weakness of REST is that it is not optimal for circumstances where constant bi-directional communication is important between client and server. Indeed, the WebSocket Protocol Standards Track Document[5] states the problem as such:

```
Historically, creating web applications that need
bidirectional communication between a client and a server
(e.g. instant messaging and gaming applications) has required
an abuse of HTTP to poll the server for updates while
sending upstream notifications as distinct HTTP calls.
```

The WebSocket protocol is a newer system which excels in two-way communication between client and server systems. For chat applications like the subject of this project, WebSocket appears perfectly suited to the problem. Due to the comparative simplicity of bidirectional communication using WebSockets, the protocol uses significantly less energy[6] to maintain a client/server connection. This is especially relevant when considering that this is a mobile application which needs to preserve battery life.

During the literature review of this topic, it became apparent that there was a way to achieve the best of both worlds. Socket.io[14] is a cross-platform library which initially attempts to create a WebSocket connection and, if not possible, falls back to HTTP polling. It also provides helpful additional features like automatic reconnection. It was determined that both client and server implementations existed in the desired languages (Python/Kotlin) and subsequently included in the system design.

The figure below describes the entire journey of a message through the messaging system. The colour scheme for the various key stages is broadly represented by the following:

1. Green: Initial message composition, server querying and local storage.
2. Yellow: Asymmetric + symmetric encryption, encoding and send process
3. Red: Server routing for message
4. Purple: Receipt, decryption, display and storage of message.

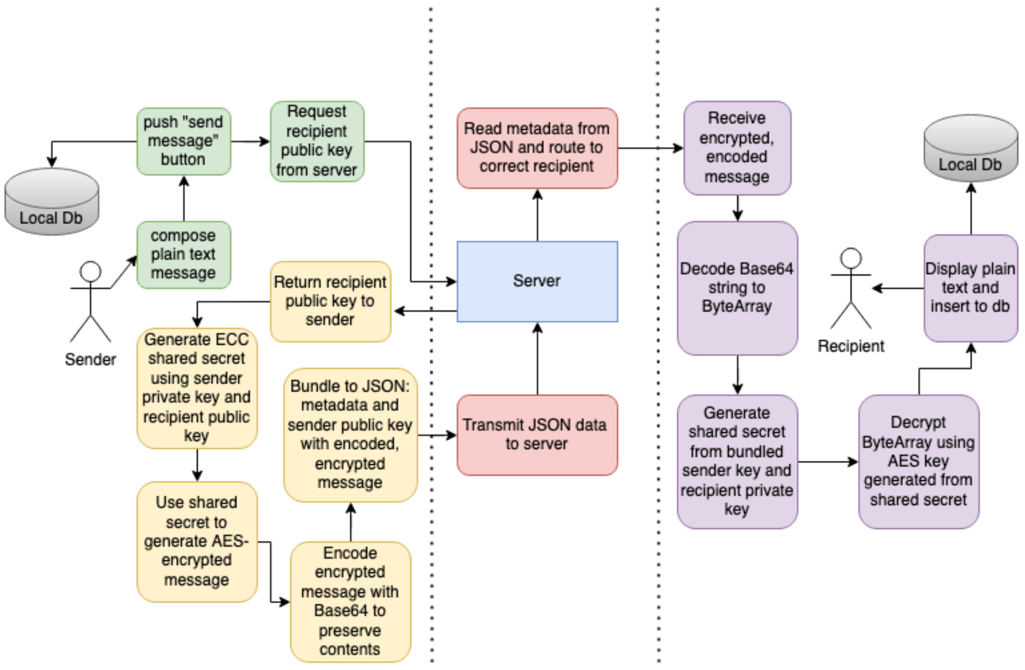
push "send message" button
Request recipient public key from server
Local Db
compose plain text message
Sender
Return recipient public key to sender
Generate ECC shared secret using sender private key and recipient public key
Bundle to JSON: metadata and sender public key with encoded, encrypted message
Use shared secret to generate AES-encrypted message
Encode encrypted message with Base64 to preserve contents
Read metadata from JSON and route to correct recipient
Server
Transmit JSON data to server
Receive encrypted, encoded message
Decode Base64 string to ByteArray
Generate shared secret from bundled sender key and recipient private key
Recipient
Local Db
Display plain text and insert to db
Decrypt ByteArray using AES key generated from shared secret

Figure 3.2: Flowchart with high level view of entire lifecycle of a message

3.3 Encryption Implementation

As mentioned previously, the system was designed with the prevailing goal of having two clients who could communicate with each other through an insecure, untrusted environment (i.e. the server) without needing to worry about message being vulnerable to man-in-the-middle attacks[8].

Two of the most prevalent encryption approaches used in the modern world are asymmetric and symmetric encryption. Symmetric encryption is the simpler of the two options - a secret key is used to alter the content of a piece of plain text in such a way that it's unreadable to anyone except someone who also has the secret key and can use it to decrypt the message.

[10]Here is a placeholder for a piece of text about ECIES.

3.3.1 Asymmetric component

3.3.2 Symmetric component

3.3.3 Difficulties

3.3.4 Storage of encryption keypair

3.4 Design Patterns

3.4.1 Dependency injection

3.4.2 Command pattern

3.4.3 Singleton pattern

3.4.4 Observer pattern

3.4.5 Adapter pattern

3.4.6 Data Access Object pattern

3.5 Persistence and storage considerations

3.5.1 Storage of self-destruct duration

3.5.2 Existing user persistence and reconnection flow

3.6 Development tooling

There are two primary components to the overall system - one or more Android mobile client applications which communicate with a central, deployed server.

3.6.1 Client

The final client application was written entirely in Kotlin (v1.6.10) using the Android Studio IDE.

Development and debugging process

Traditional Android software development involves using the qEmu device emulator built into Android Studio to imitate the experience of a real user on their own device.

Debugging code during the HushChat development process was a complex endeavour, as the development machine had fairly limited RAM. As the purpose of the application was communication between two clients, it was important to be able to emulate two devices at the same time so messages could be exchanged. This was initially possible on the development machine, but became unfeasible over time as the complexity of the system grew and memory capacity became strained.

The debugging environment eventually morphed into a single qEmu emulated device on the development machine combined with a physical android device

(Google Pixel 4). This methodology became possible with the introduction of Android version 11 and Android Studio 'Bumblebee' (v2021.1), which allows the developer to run Android debugging tools (i.e the Android Debug Bridge, or ADB) wirelessly. This feature was extremely helpful in the development process, and it allowed for the introduction of further feature complexity that would not have been possible otherwise due to the technical limitations of the development machine.

Running a WebSocket-based application locally also presented a minor challenge, as qEmu virtual devices use the traditional localhost/127.0.0.1 address to represent their own internal loopback interface rather than that of the host development machine. This meant that to communicate with the locally hosted server, the special debugging address 10.0.2.2 was used. After the development of the server side of the application was complete, this problem was solved by deploying the server to a static IP address (via a Azure Virtual Machine).

Key Dependencies

- Socket.io - v2.0.0 - the developers of socket.io provide a native Java implementation. Due to Java's seamless interoperability with Kotlin, this did not cause any implementation problems with the version implemented.

- BouncyCastle - v1.67 - BouncyCastle is the cryptography API that performs a lot of the cryptographic operations within the application, including generation of elliptic curve keypairs. There is a built-in version of BouncyCastle within the Android SDK, but this does not support ECC, which meant that the client application needed to replace the inbuilt library with a more updated version. This can be seen within the codebase with the following calls within the MainActivity and ChatWindow classes:

  ```
  Security.removeProvider("BC")
  Security.addProvider(BouncyCastleProvider())
  ```

- Room ORM - v2.4.0 - Room provides an object-relational mapping (ORM) over the SQLite database built into the Android SDK. When developing the application, the main data model to consider on the client side was that of an individual chat message, including metadata (e.g. recipient) and content. Working with Room allowed for a more streamlined development experience and reduction of boilerplate code, as it meant working directly with Kotlin (Java) entities instead of writing complex queries for inserting messages to our viewmodel. SQL queries were still used to capture relevant data to display to the user in chat windows.

3.6.2 Server

The server application was written in Python v3.9.1 in the PyCharm IDE and utilised several libraries which are highlighted below.

Key Dependencies

- python-socketio - v5.5.0 - a Python server implementation of socket.io, funded by the original socket.io developers.

- aiohttp - v3.8.1 - an asynchronous http server which (importantly) supports websockets. The socket.io process attaches itself to the aiohttp server which allows information to be transmitted and received from clients.

Deployment

To allow users to message other users, it was necessary to deploy the server application on a static IP. To do this, a 1GB/1CPU Virtual Machine instance was used, running Ubuntu 20.04 in the West Europe region of Microsoft Azure.

3.6.3 Final report

This dissertation was written entirely in LaTeX using Neovim v0.5.1 with the VimTex plugin.

3.6.4 Version Control

Git was the version control solution used throughout the project. Mild discomfort with the idea of storing the written report with a private company (Overleaf) while writing meant that it was worth taking the step of compiling the document locally. This had the fortunate benefit of meaning the entire project could be stored in a single git repository (as well as an opportunity to learn a lot of new things).

The host of choice for the upstream repository on this project was GitHub, and the entirety of the project can be found at the following link:

https://github.com/furnivall/SEng_Final_Project

Chapter 4: Evaluation & Testing

Chapter 5: Conclusion

Appendix A: First appendix

A.1 Section of first appendix

Appendix B: Second appendix

Bibliography

[1] R. Baeza-Yates and U. M. Fayyad. The attention economy and the impact of artificial intelligence. In *Perspectives on Digital Humanism*, pages 123–134. Springer, Cham, 2022.

[2] H. Berghel. Malice domestic: the cambridge analytica dystopia. *Computer*, 51(5):84–89, 2018.

[3] J. P. Degabriele. Hiding the lengths of encrypted messages via gaussian padding. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1549–1565, 2021.

[4] A. Farahat and M. C. Bailey. How effective is targeted advertising? In *Proceedings of the 21st international conference on World Wide Web*, pages 111–120, 2012.

[5] I. Fette and A. Melnikov. The websocket protocol, 2011.

[6] V. Herwig, R. Fischer, and P. Braun. Assessment of rest and websocket in regards to their energy consumption for mobile applications. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 342–347. IEEE, 2015.

[7] D. Kreiss and S. C. McGregor. The "arbiters of what our voters see": facebook and google's struggle with policy, process, and enforcement around political advertising. *Political Communication*, 36(4):499–522, 2019.

[8] A. Mallik. Man-in-the-middle-attack: understanding in simple words. *Cyberspace: Jurnal Pendidikan Teknologi Informasi*, 2(2):109–134, 2019.

[9] B. Marczak, J. Scott-Railton, S. McKune, B. Abdul Razzak, and R. Deibert. HIDE AND SEEK: Tracking NSO Group's Pegasus Spyware to operations in 45 countries. Technical report, 2018.

[10] V. G. Martınez, L. H. Encinas, et al. A comparison of the standardized versions of ecies. In *2010 Sixth International Conference on Information Assurance and Security*, pages 1–4. IEEE, 2010.

[11] M. Masse. *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.", 2011.

[12] M. Mazzetti, A. Goldman, R. Bergman, and N. Perlroth. A new age of warfare: how internet mercenaries do battle for authoritarian governments. *The New York Times*, 21:2019, 2019.

[13] K. Pantelis and L. Aija. Understanding the value of (big) data. In *2013 IEEE International Conference on Big Data*, pages 38–42. IEEE, 2013.

[14] R. Rai. *Socket. IO Real-time Web Application Development*. Packt Publishing Ltd, 2013.

[15] R. Simmons. Big data and procedural justice: legitimizing algorithms in the criminal justice system. *Ohio St. J. Crim. L.*, 15:573, 2017.

[16] P. Sommer. Evidence from hacking: a few tiresome problems. *Forensic Science International: Digital Investigation*, 40:301333, 2022.

[17] C. Stöcker. How facebook and google accidentally created a perfect ecosystem for targeted disinformation. In *Multidisciplinary International Symposium on Disinformation in Open Online Media*, pages 129–149. Springer, 2019.

[18] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. Sok: secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249. IEEE, 2015.

[19] C. Walter. Kryder's law. *Scientific American*, 293(2):32–33, 2005.

[20] S. Zawoad and R. Hasan. Digital forensics in the age of big data: challenges, approaches, and opportunities. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 1320–1325. IEEE, 2015.