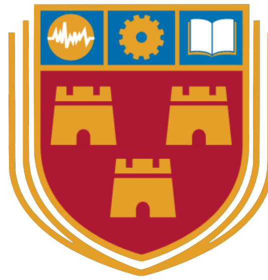
Secure Communication Platform Design Manual



Liliana O'Sullivan

C00227188

Submitted in partial satisfaction of the requirements for the degree of

BSc of Science (Hons) Software Development

Faculty of Science

Institute of Technology, Carlow

18th of December 2020

Abstract

This design manual discusses the design and decisions to create the proposed Secure Communication Platform. It will provide a high-level overview of the technical decisions.

Table of Contents

1 Introduction **1**

2 Project Plan **2**

3 System Architecture **3**

3.1 Password Hashing 4
3.2 Two-Person Chat 4
3.3 Group Chat 6

4 System Flow **7**

4.1 FastAPI request 7
4.2 Establishing a connection to a Web-Socket 8
4.3 Send Message 8

5 API Design **9**

5.1 JSON 9
5.2 OpenAPI 9
5.2.1 HTTP Request Methods 10

6 Database Design **11**

6.1 User Table 11
6.2 API Keys Table 12

List of Figures

2.1 Project Gantt Chart 2

3.1 System Architecture 4
3.2 Alice and Bob forward public keys 5
3.3 Alice Requests Bob Public Key 5
3.4 Server Forward Message 5

4.1 FastAPI Request-Flow 7
4.2 Connecting to a Web-Socket 8
4.3 Sending a message 8

6.1 Create Ilo keyspace 11
6.2 Create Users table 12
6.3 Create API table 12

List of Tables

6.1 Users Table 11
6.2 API Keys Table 12

Chapter 1

Introduction

This project aims to create a platform where users can communicate with encryption in place. The project will develop an Application Programming Interface (API) to enable developers to create a client to consume the service. All information will be encrypted end-to-end as users communicate with each other, specifically in transit and at rest.

The system will be split into two independent systems to accommodate best the features provided. A database will store users and API Key information

Chapter 2

Project Plan

In Figure 2.1 can be seen a Gantt chart of the project timeline. With project timelines of development.

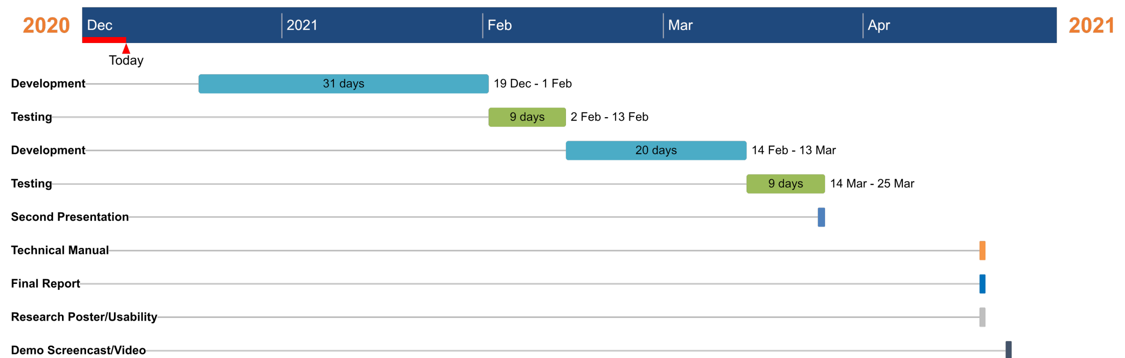


Figure 2.1: Project Gantt Chart

Chapter 3

System Architecture

The server architecture will be split into two systems. There will be an Elixir and a Python system. The system is to be architected in a form that best takes advantage of each technologies strengths.

Elixir will be used to process the user's conversations. Conversations are latency-sensitive applications and require rapid processing. Elixir is created to execute on the BEAM virtual machine, the backend for executing Erlang source code. This enables Elixir to inherit the concurrency and reliability capabilities of Erlang while introducing new semantics and having the ability to integrate existing Erlang libraries. As a result, Elixir is a powerful candidate for latency-sensitive, highly concurrent environments. It will be used to create a web-socket server that the client can communicate with to send messages.

Python will be used alongside the Elixir web-socket server. Python will serve as the central API server through the use of the FastAPI framework. Python is chosen because it provides acceptable performance while gaining many of it's advantages, such as providing robust tools, emphasis on code readability, having a large community base and much more. FastAPI is a web framework designed for building APIs. Its API-centric design provides many useful out-of-the-box features; for example, if an invalid request is sent to the API with invalid parameters, FastAPI, by default, responds in JSON with the error.

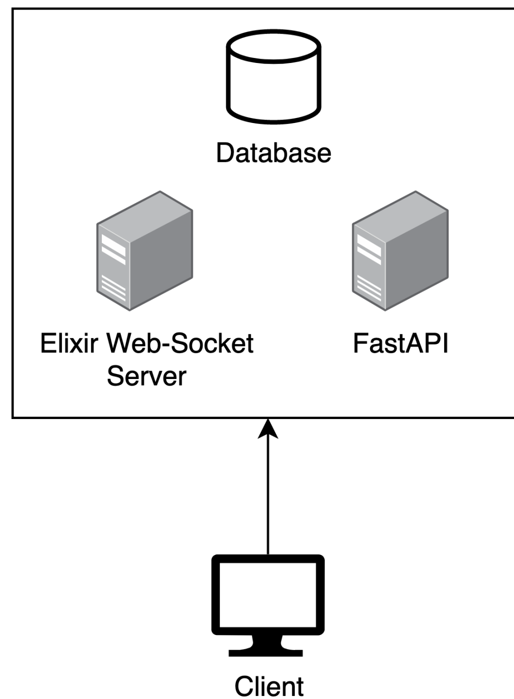


Figure 3.1: System Architecture

3.1 Password Hashing

Passwords will be hashed using Argon2id by using the Argon2id-cffi package provided by PyPi. Argon2id is chosen over Scrypt or Blake from the standard library, as it showed to be

3.2 Two-Person Chat

In a two-person conversation, encryption can be achieved by submitting the public key to the server, which will forward the key to the desired user. Each user of the conversation generates their keypairs (Private and public keys). They submit their keys to the server. The figures in this section will depict two actors, Alice and Bob, desiring to converse. In Figure 3.2, Alice and Bob each submit their public keys to the server.

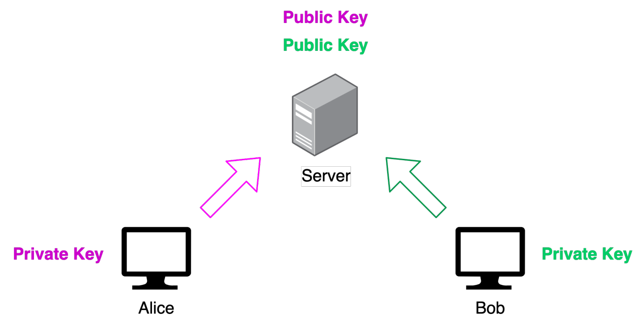


Figure 3.2: Alice and Bob forward public keys

Once Alice desires to send a message, a request for Bob's public key is submitted. Public keys are considered safe to openly share, as they can only be used to encrypt a message. This scenario is depicted in Figure 3.3.

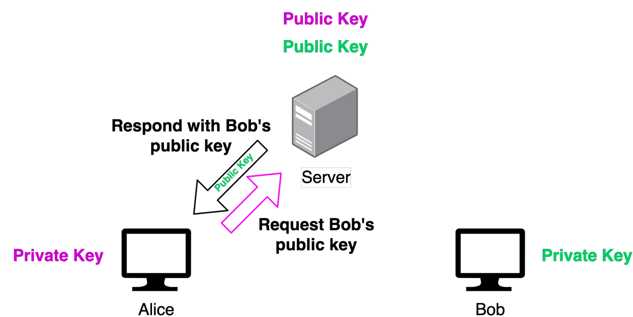


Figure 3.3: Alice Requests Bob Public Key

Once Bob's public key is obtained, the key can be used to encrypt the message. The encrypted message is sent to the server to be forwardedto Bob as depicted in Figure 3.3. This method is based on asymmetric encryption. This means the keys used to decrypt and encrypt are two separate keys.

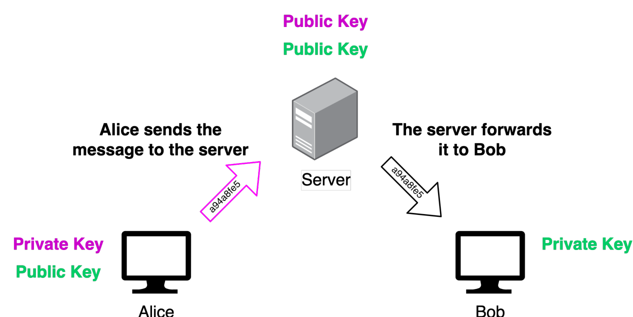


Figure 3.4: Server Forward Message

3.3 Group Chat

Group chat encryption has increased difficulty. It can create the situation if a participant desires to send a message; whose public key does Bob use to encrypt the message? This will be implemented using symmetric cryptography. This means the process of encrypting and decrypting is based on a single, shared key. Each participant will be granted access to this key. The sender will encrypt the message intended for all participants using this shared key and send the ciphertext to the server. All participants will use their shared key to read the message.

The creation of this group key is handled by the member who created the group. As they add members to a group, the secret key is sent to them privately using their public key.

Chapter 4

System Flow

This section demonstrates how a request is processed within the system. Three main scenarios' exist.

- Sending a request to FastAPI, such as the creation of a user
- Connecting to an Elixir Web-Socket
- Sending a message through Elixir's Web-Sockets

Not every request will use all parts of the system at one time.

4.1 FastAPI request

When a client creates a request to FastAPI, FastAPI will performed the required checks before querying the required database table. This is shown in figure 4.1.

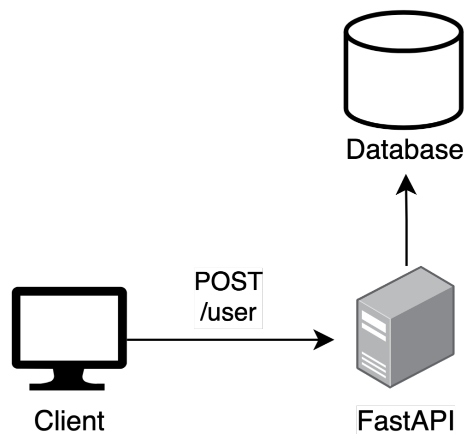


Figure 4.1: FastAPI Request-Flow

4.2 Establishing a connection to a Web-Socket

A client creates the web socket connection to the Elixir server directly. Elixir will verify the request by ensuring the requesting user is logged in. If the user is logged in, FastAPI responds with success. Assuming a successful verification, Elixir will continue with the creation of the Web-Socket. This dataflow can be seen in figure 4.2.

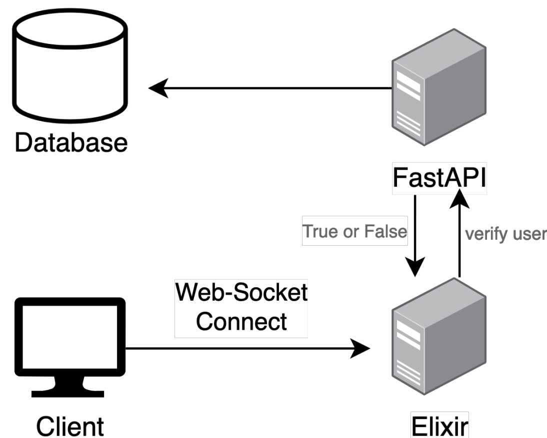


Figure 4.2: Connecting to a Web-Socket

4.3 Send Message

This is the elegance of the system, once a client wishes to send a message, the Elixir server will forward the message to other clients connected to the room without the need of communicating to FastAPI or a database. This is showcased in figure 4.3.

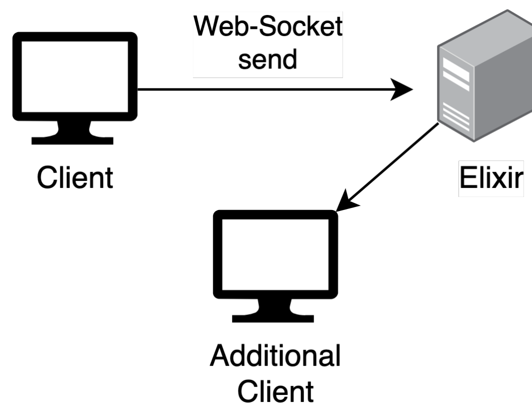


Figure 4.3: Sending a message

Chapter 5

API Design

The API design is architected fundamentally around two standards.

- JSON (JavaScript Object Notation)
- OpenAPI (Previously known as Swagger Specification)

5.1 JSON

JSON is a data-interchange format that has gained widespread support among a multitude of programming language. Additionally, it remains relatively human-readable while maintaining parsing overhead. For many dynamically typed languages such as Ruby, Python or JavaScript, JSON can become a natural extension of objects available; Python represents its' dictionaries with the brace notation to encapsulate objects using similar conventions to the JSON standard. As JSON is language independent and popular, it is an ideal choice for data interchange.

5.2 OpenAPI

The OpenAPI Specification is a specification for describing RESTful APIs. The OpenAPI initiative currently maintains it as part of the Linux Foundation; Initially, it was part of the Swagger framework until becoming a separate project. It is a JSON-based specification that states how Hypertext Transfer Protocol (HTTP) Request

Methods (Also sometimes known as HTTP Request Actions) should be used to interact with a RESTful API.

An OpenAPI specification can be used by developer tools such as Postman, Insomnia or Paw to test API methods, generate code, and be presented with a visual and interactive interface describing the interactions of the service.

5.2.1 HTTP Request Methods

The server utilises various HTTP request methods to conform to standards and simplify the usage of the API. In theory, any HTTP request method can perform the action of another if developed in that fashion; A POST method could be used to delete an entity, or a GET could be used to create an entity. While technically possible, this is not ideal behaviour, and as such, the system uses appropriate methods to perform different actions.

Chapter 6

Database Design

This is the database schema used within Cassandra. Included within each table is the command needed to create each table. Cassandra uses a keyspace when referencing a table, this can be loosely likened to a 'database' in SQL and MongoDB terms. This can be created using the code in figure 6.1

```
1  CREATE keyspace ilo WITH
   ↪  replication={'class':'SimpleStrategy','replication_factor':3};
```

Figure 6.1: Create Ilo keyspace

6.1 User Table

The table in 6.1 contains the information stored by a user.

user_id	username	api_key	logged_in	login_time	password	public_key
e75ab2f6..	NimbleIris	abxfsdg..	39.1.63.1	161.5	NkuCmvas..	MIICIj..
8b4fad26..	cookie12	a01babx..	12.143.11.6	150.123	Hy3wy4w..	gKCAg..

Table 6.1: Users Table

The structure shown in table 6.1 can be created using the command in figure 6.2. An index of the username and logged_in column must be created to allow the columns to be independently queries without obtaining the primary key.

```
1  CREATE TABLE ilo.users (
2      user_id uuid PRIMARY KEY,
3      api_key text,
4      logged_in text,
5      login_time double,
6      password text,
7      public_key text,
8      username text
9  );
10 CREATE INDEX users_logged_in_idx ON ilo.users (logged_in);
11 CREATE INDEX users_username_idx ON ilo.users (username);
```

Figure 6.2: Create Users table

6.2 API Keys Table

key_id	creation_epoch
8524-26ce051db2f6	1618242690.96891
996e-d20166fc223f	1608217341.16192

Table 6.2: API Keys Table

The structure shown in table 6.2 can be created using the command in figure 6.3

```
1  CREATE TABLE ilo.api_keys (
2      key_id uuid PRIMARY KEY,
3      creation_epoch double
4  );
```

Figure 6.3: Create API table