BLANK

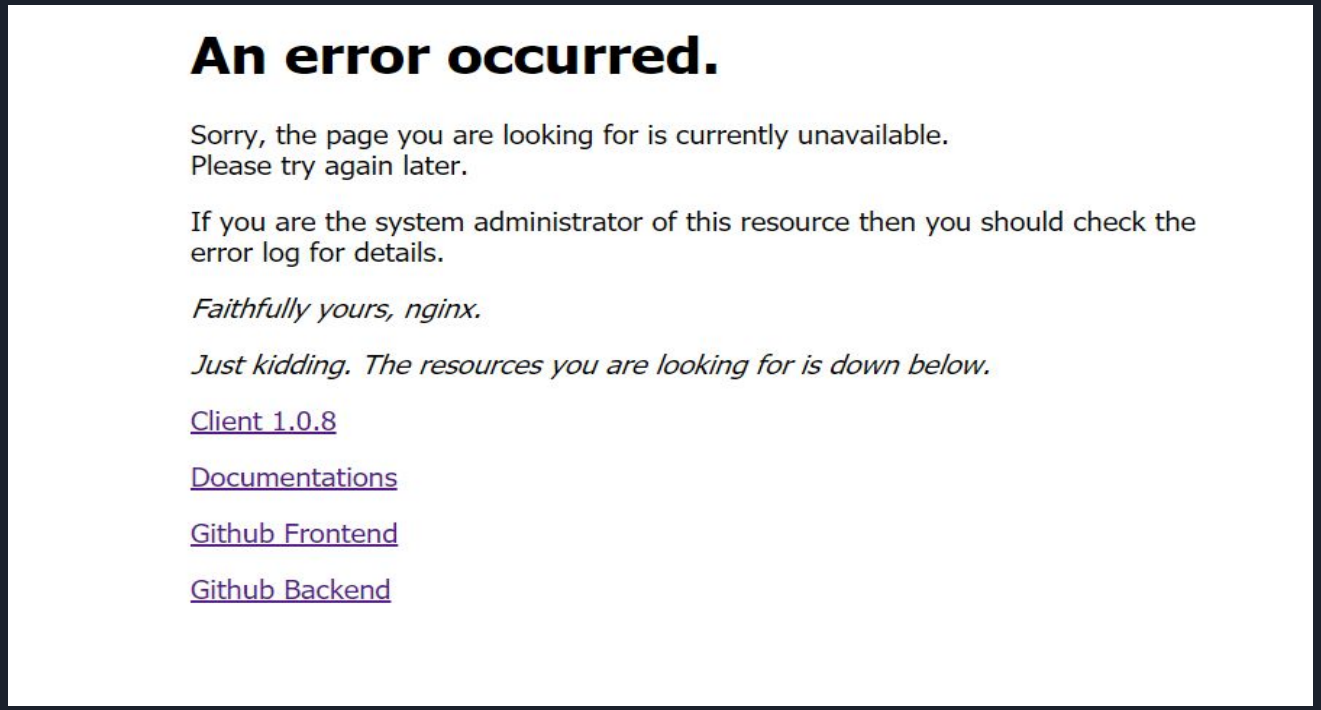
An Encrypted Chat Application

By : CyberGuard
Members:

- Sok Sotheanith
- Hayat Ahmed

Demo

URL: https://cgencryptedchat.me/



An error occurred.

Sorry, the page you are looking for is currently unavailable.
Please try again later.

If you are the system administrator of this resource then you should check the error log for details.

Faithfully yours, nginx.

Just kidding. The resources you are looking for is down below.

Client 1.0.8

Documentations

Github Frontend

Github Backend

Demo

URL: https://youtu.be/lzYZ23mC1yw

BLANK
An Encrypted Chat Application

The purpose of this project is to design a secure (hopefully) end-to-end encrypted messaging application.

Content

1. System Overview
2. Implementation
3. Vulnerabilities
4. Improvements/Future Features

System Overview

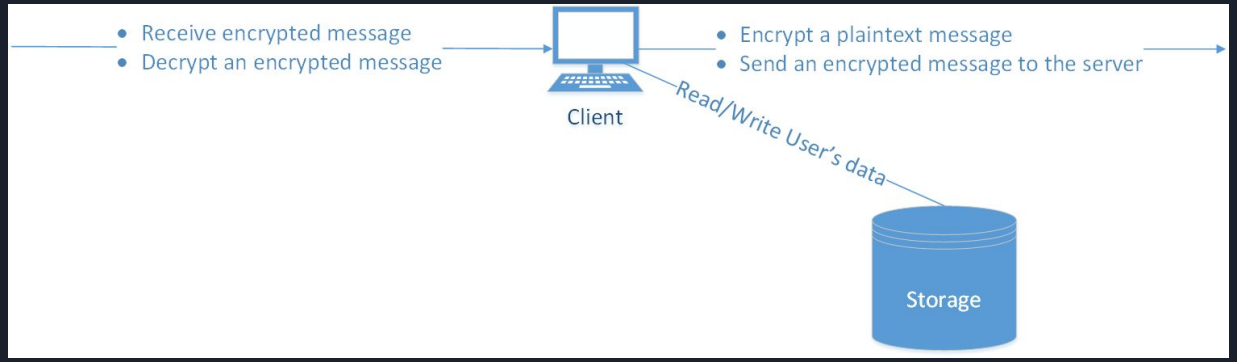
The system can be divided into three standalone components:

- Client
- Server
- Communication Channel

Client

Responsibilities:

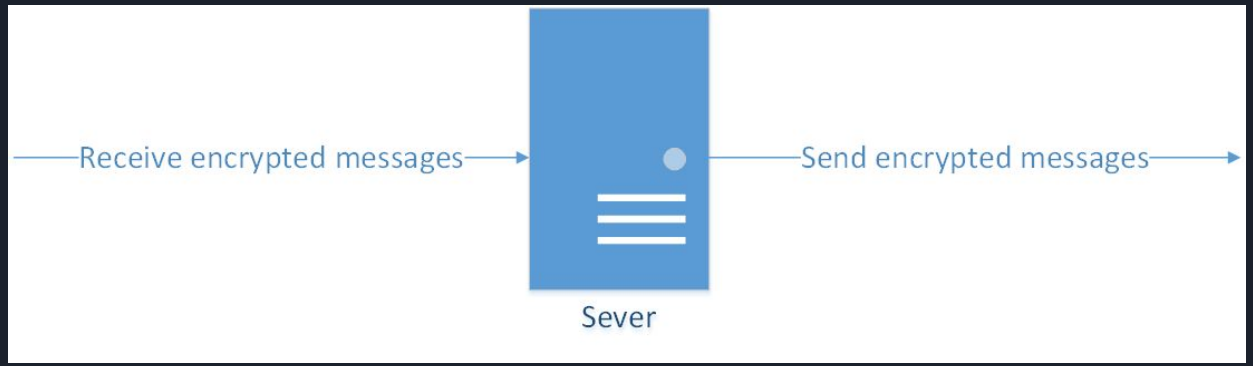
- Encrypt plaintext messages
- Decrypt encrypted messages
- Send encrypted message to the server
- Retrieve encrypted messages from the server
- Manage users' sensitive information such as RSA key pairs



Sever

Responsibilities:

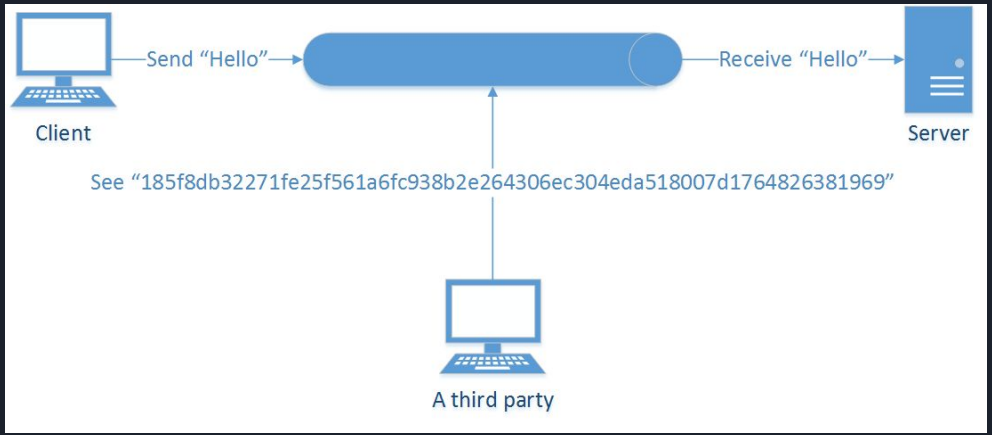
- Distributes encrypted messages to appropriate clients



Communication Channel

Responsibilities:

- Protect the transportation of data during transport so that only a sender and an intended receiver can see the data.



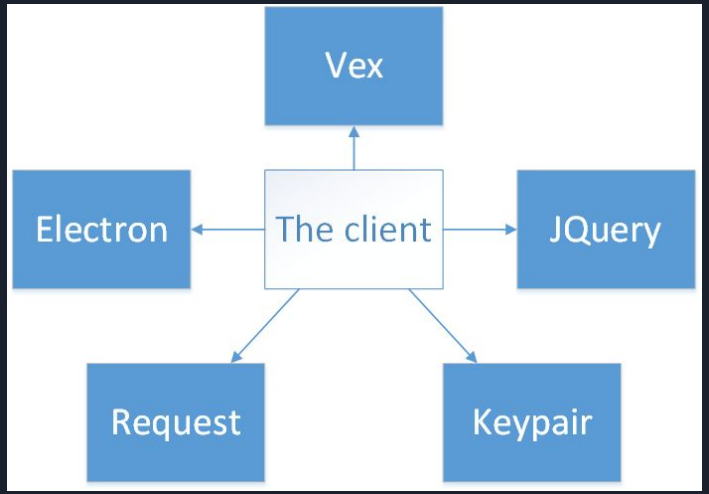
Implementation

Client

Client’s Dependencies

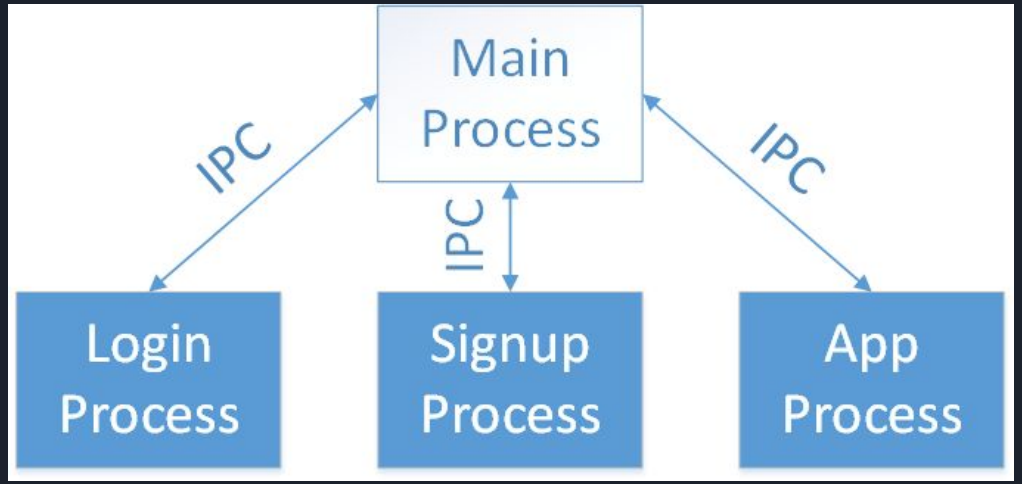
The current implementation of the client relies on five libraries.

- **Electron**: a framework allows for the development of desktop GUI application using Node.js for the backend and Chronum for the frontend.
- **JQuery**: a javascript library that simplifies HTML dom design.
- **Keypair**: a pure implementation of RSA key pair generation in javascript.
- **Request**: a javascript library that simplifies the way to make HTTPS call.
- **Vex**: a library that allows the creation of asynchronous dialogs.



Client’s Structures

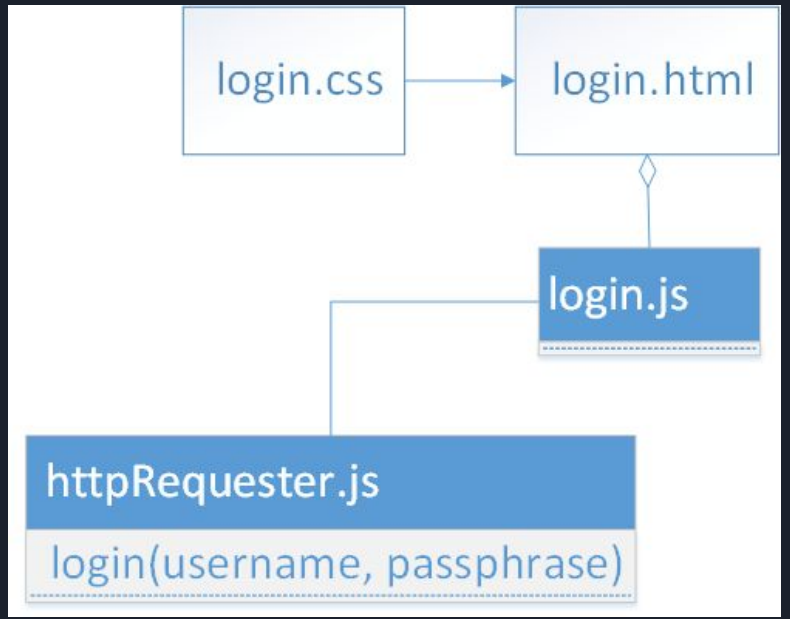
Since the client is built using an Electron framework, its structure is similar to an average chromium-based application where three is one main process and multiple rendering processes.



Login Process

Responsibilities:

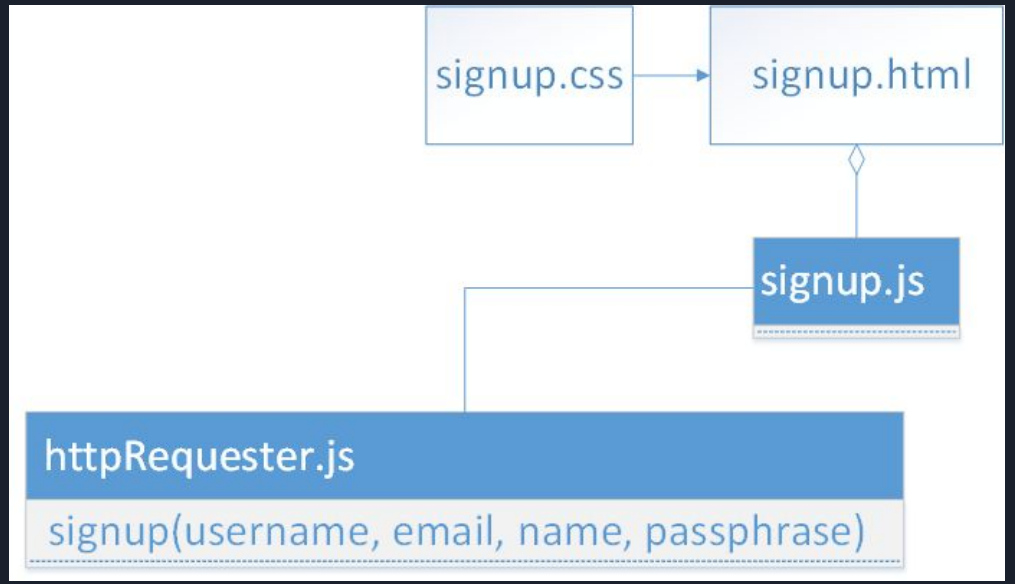
- Make a HTTPS login request to the server
- Receive JSON web token from the server
- Relay the JSON web token to the main process



Signup Process

Responsibilities:

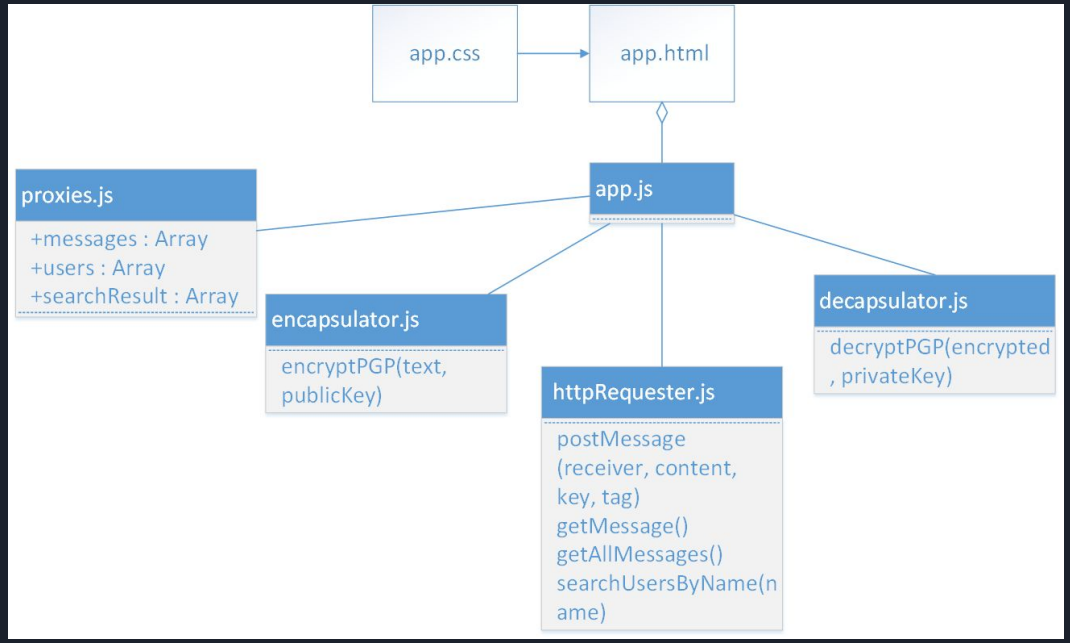
- Make a HTTPS signup request to the server



App Process

Responsibilities:

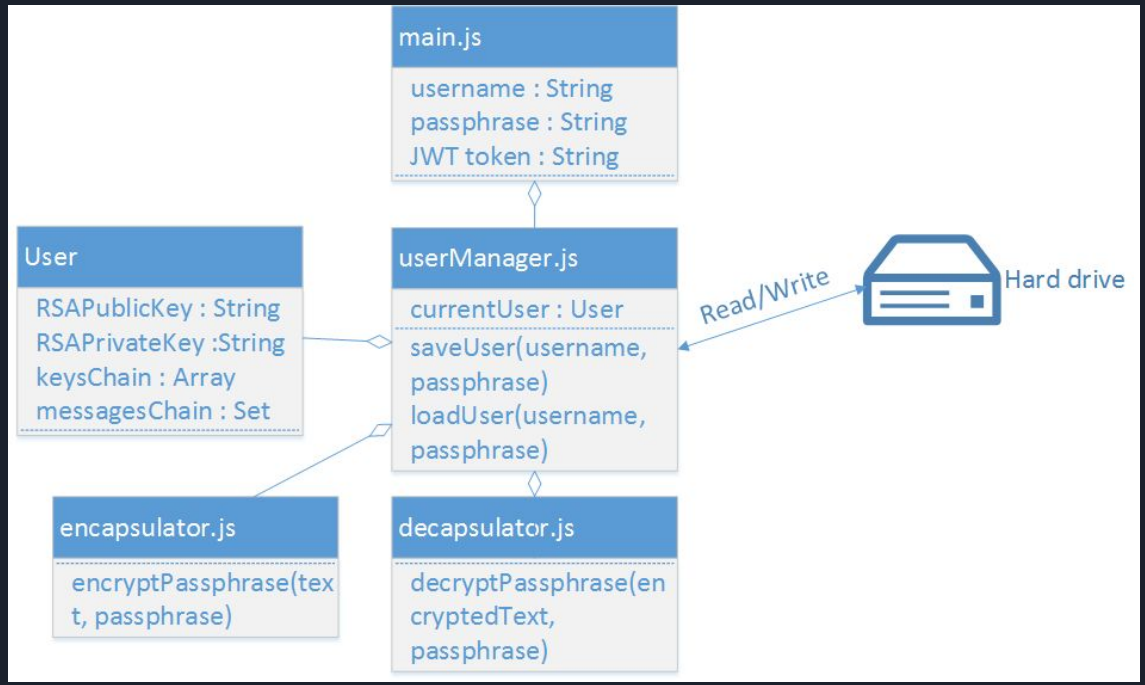
- Encrypt plaintext messages
- Decrypt encrypted messages
- Post encrypted messages to the server
- Retrieve encrypted messages from the server
- Search for existing users on the server



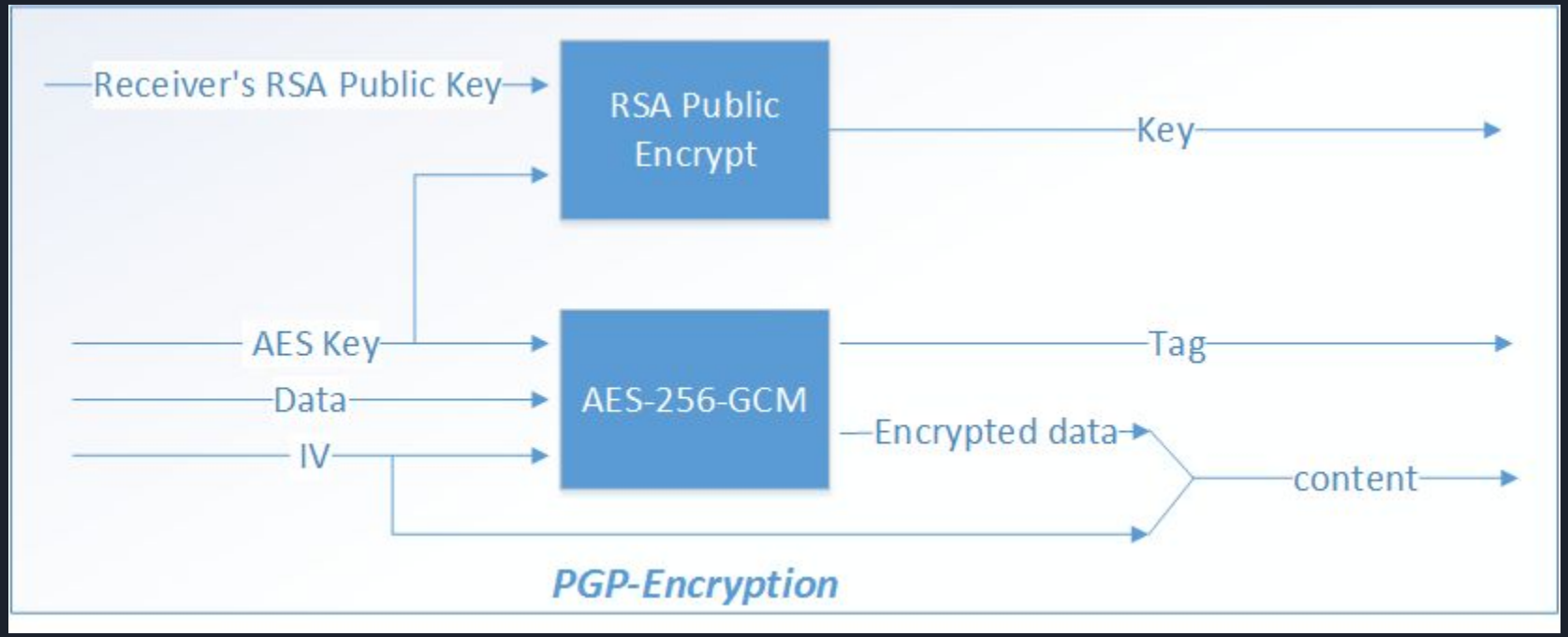
Main Process

Responsibilities:

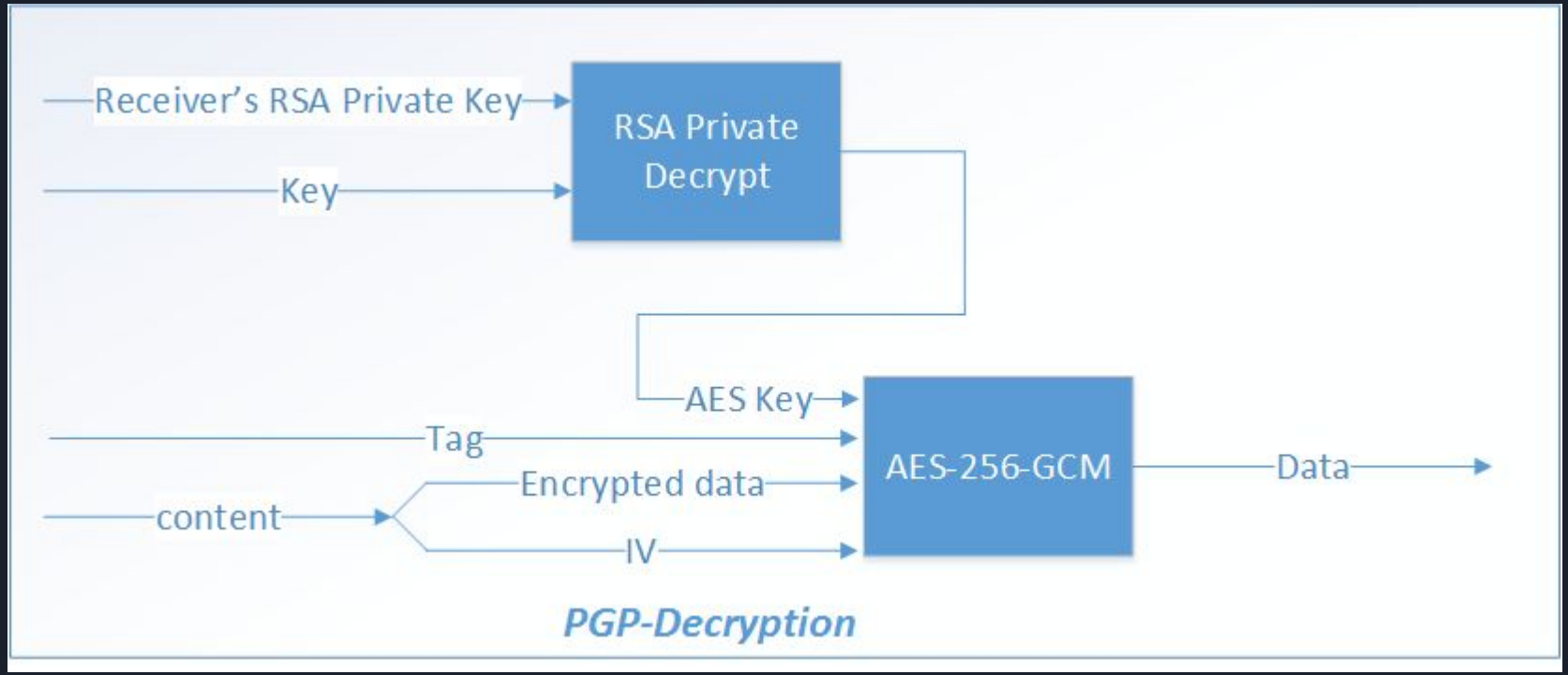
- Managing and protecting users' data including RSA key pair
- Distributing resources to rendering processes



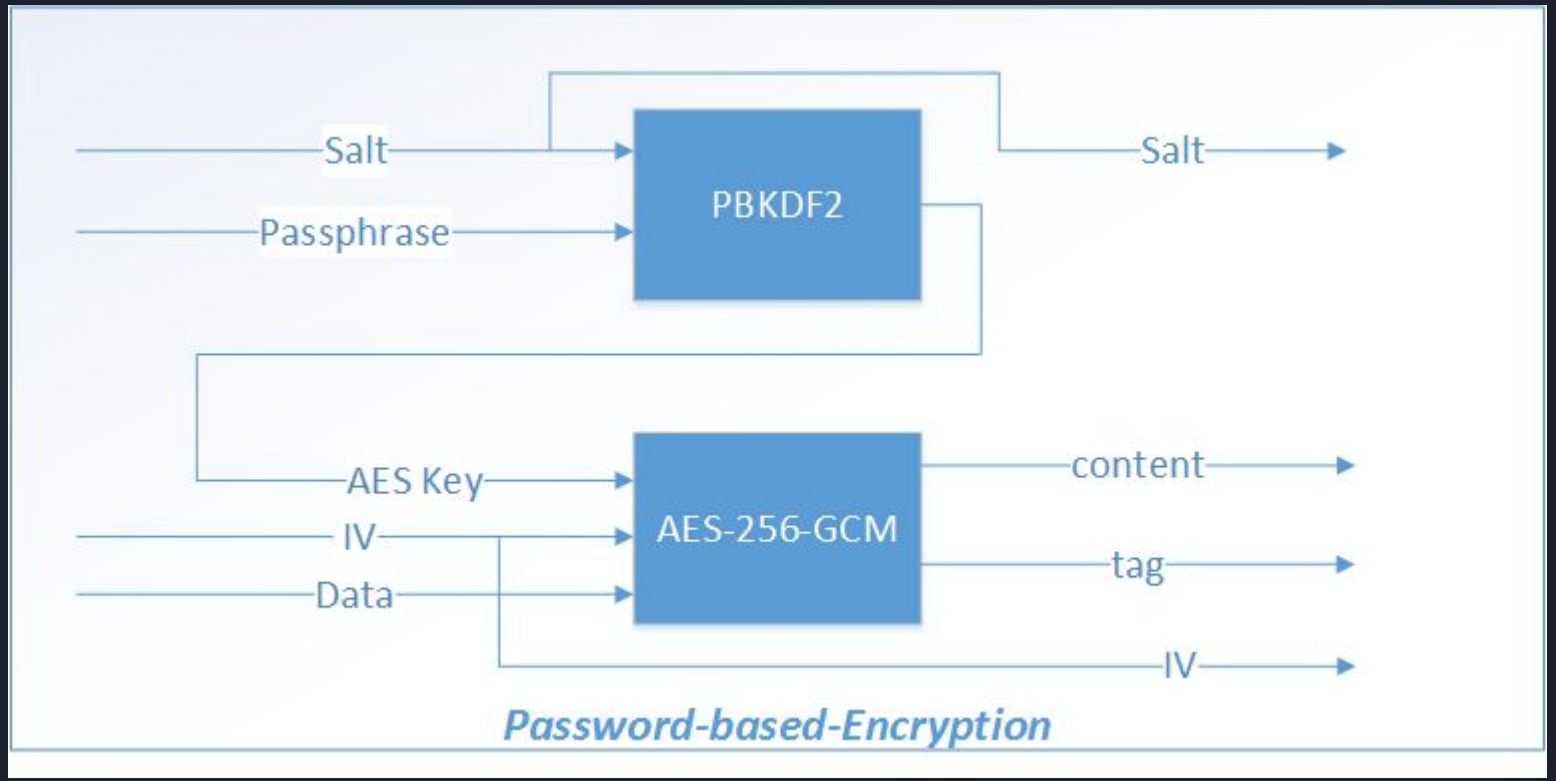
Encryption/Decryption Procedures



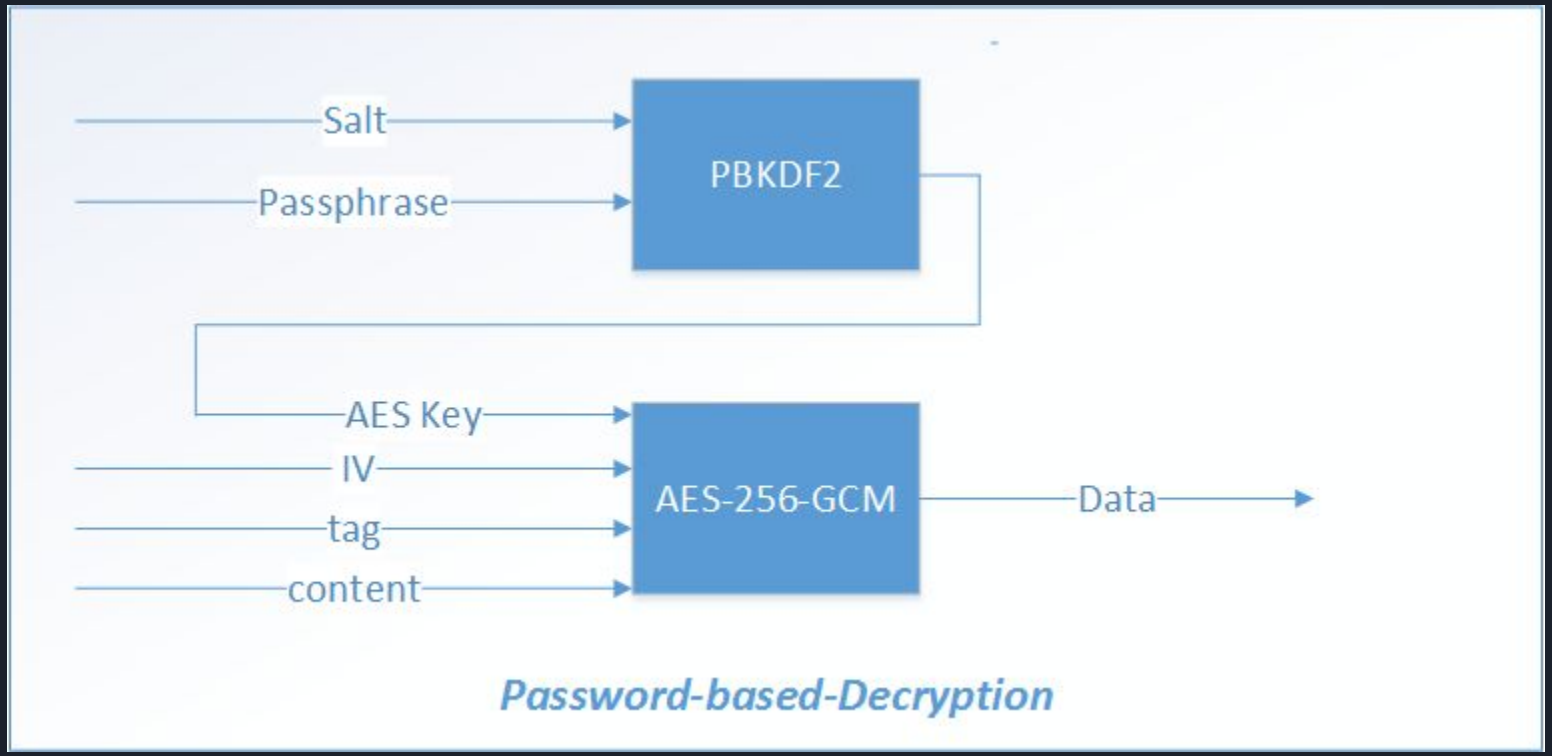
Encryption procedure for encrypting plaintext messages



Decryption procedure for decrypting encrypted messages



Encryption procedure for encrypting users' data based on their passphrase



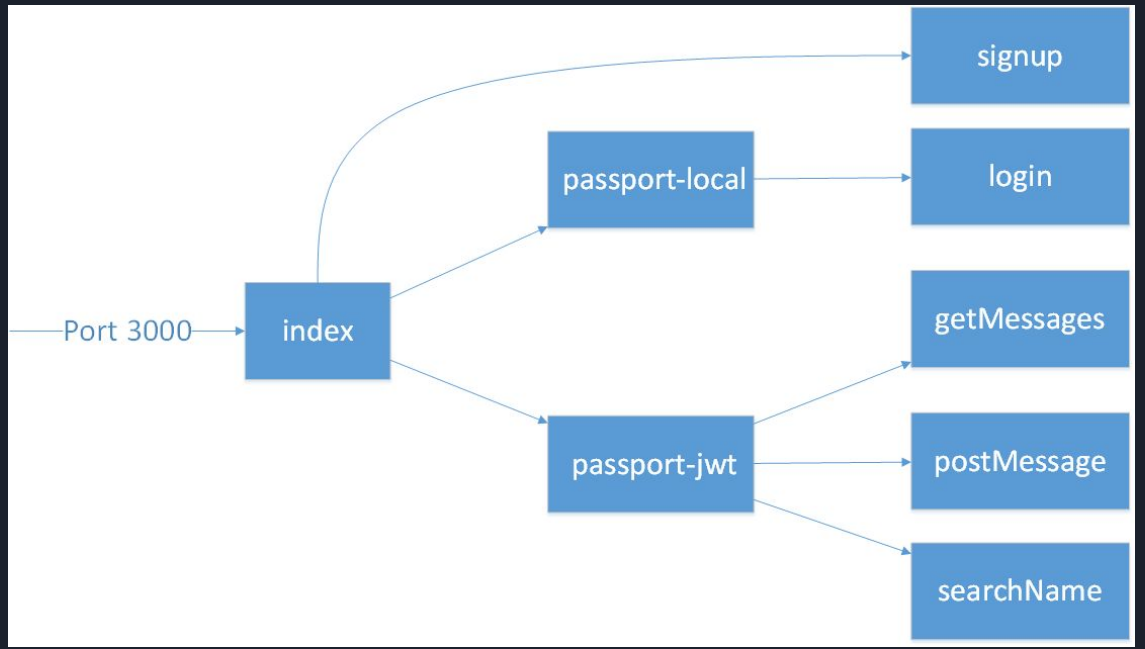
Decryption procedure for decrypting users' data based on their passphrase

Server

Sever's Routes

The server provides two types of routes

- Unsecured routes: ways in which users can authenticate themselves to the server
- Secured routes: ways for authorized users to access the resources provided by the server



Server’s Security

Users’ Passphrase:

- The passphrase of users is hashed with a random salt and stored on the server using bcrypt.

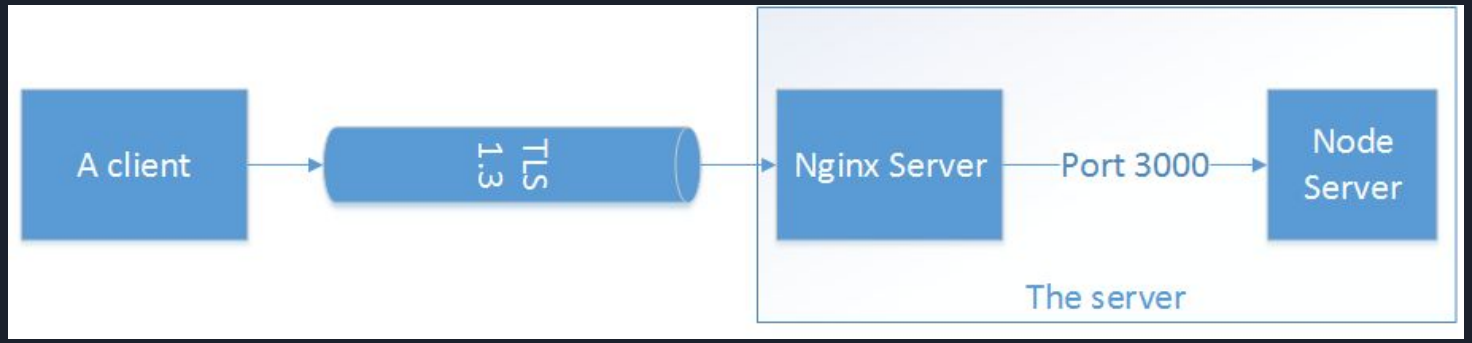
JSON Web Token:

- The server is using a symmetric key to sign and verify JSON web tokens.

Communication Channel

Communication Channel

The communication between a client and the server is secured using TLS 1.3 . On the initial connection, a client and the server will perform a handshake where they verify each other identity with a trusted third party and exchange a symmetric session key in which they use to encrypt their communication.

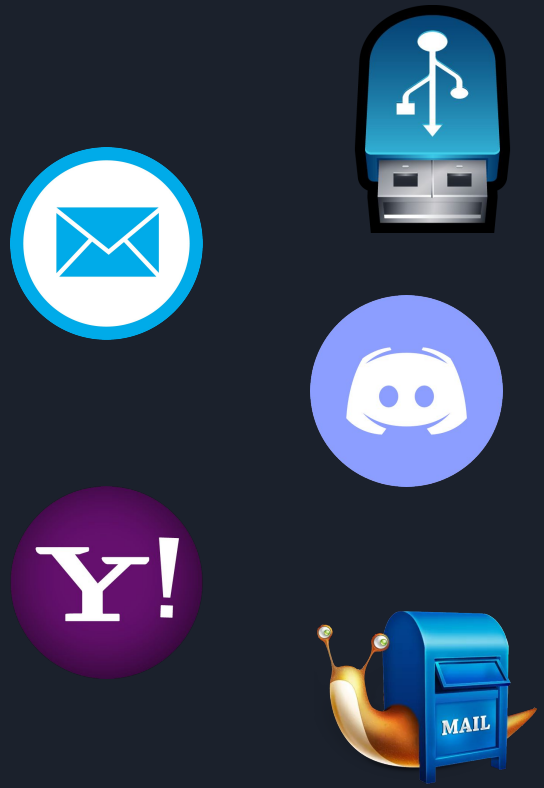


Key Exchange Infrastructure

Out-Of-Band

The current implementation of the system does not offer any official way in which users can exchange their public keys. Thus, it is up users to decide their own out-of-band ways to do the public key exchange.

Even though this decision will undoubtedly inconvenient users, it also increases the security of the system as each user has a choice to accept only public key of someone he or she already knows



Vulnerabilities

Client

Problems:

- Vulnerables to code injection
  - Built with web technology such as HTML and CSS
  - Complete node integration

Solutions:

- Turns off html parsing
- Performs html stripping
- Reduce node integration to only necessary libraries

Server

Problems:

- Susceptible to a denial-of-service-attack
- Server's signature key is semi-static
- Long-term viable of the secure-hash functions

Solutions:

- Request limit
- Email verification during signup process
- Automatic server's key rotation system.
- ???

Communication Channel

Problems:

- Users have to send their passphrase in plaintext to the server when they try to login.

Solutions:

- Implement zero-knowledge proof protocol such as Remote Login to eliminate the need for users to send their plaintext passphrase to the server when they try to login

Improvements/Future Features

It’s not a bug, it’s a features

- Perform HTML stripping or turn off HTML parsing in a client.
- Reduce node integration in a client. Implement a request rate limit in the server.
- Implement an email verification during signup process.
- Implement a signature key rotating system in the server Implement remote login protocol.
- Split the server into two independent servers.
- One server responsible for authenticating users and another server responsible for serving resources. Implement key exchange infrastructure.
- Introduce some form of encryption on data stored in the server based on such server’s software and hardware.
- Move to symmetric encryption and implement forward secrecy.
- Group messages and multimedia supports