CECS 478
Project Design and Documentation

ABConversation

www.abconversation.us

By Visal Hok and Jacob Parcell

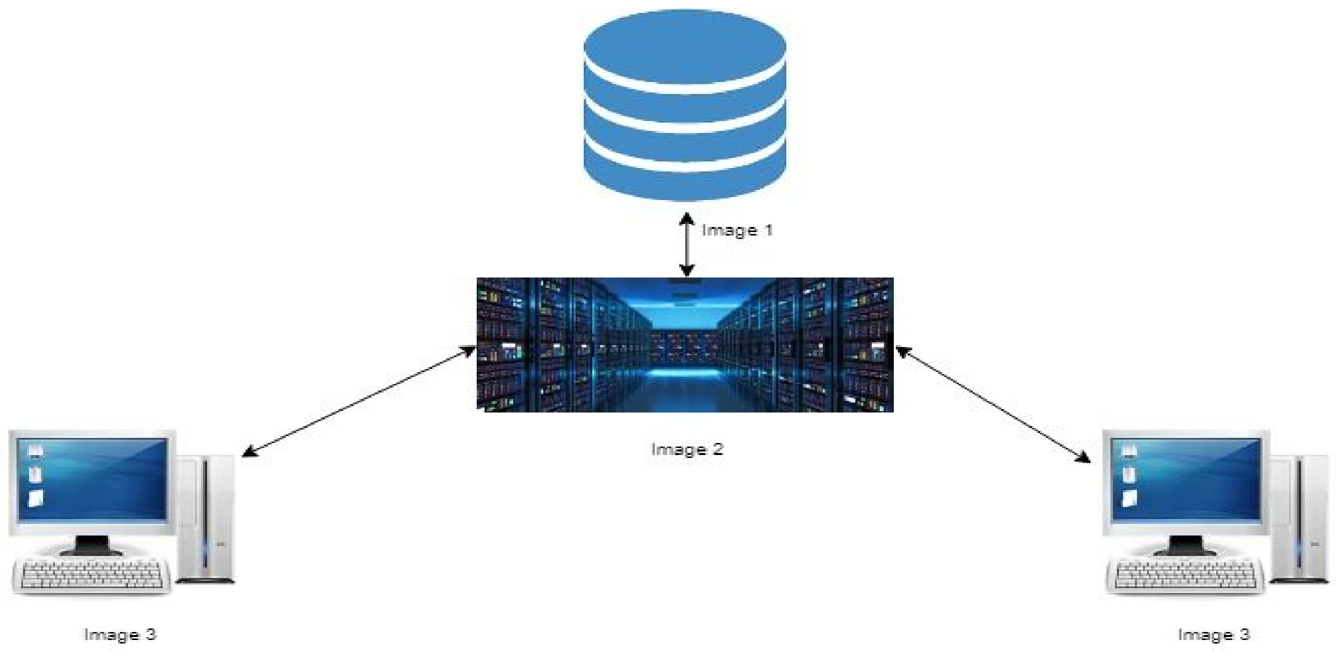
Table of Contents

1. Requirement

The purpose of this application is to provide end-to-end encryption for one-to-one messaging. In order for our application to be end-to-end, the application must be designed in a way that protects the user from the server as if the server was an adversary. This means we cannot trust the server with the distribution of keys and we also cannot allow the server to be able to understand messages going between users.

2. Overview

Our solution to the problem is to create client, server, and database. The purpose of the client is to encrypt and decrypt messages. The server will be use to transport messages from one client to another client. The database will be use to store users login credential and to temporary store messages until the client retrieve it.



3. Design

3.1. Assets

- Client:
  - Username
  - Password

    - Messages
    - RSA Private Key
    - RSA Public Key
    - AES Key
    - JSON Web Tokens
- Server:
    - User’s salt and hash password
    - User’s username
    - User’s encrypted messages
    - JSON Web Tokens Secret Key
    - Message Database
    - Users Database

3.2. Stakeholders

- User
- Application Owner

3.3. Adversaries

- Active Insider
- Active Outsider with Unlimited Resources
- Active Outsider with Limited Resources
- Passive Outsider

3.4. Attack Surfaces

- Active Outside
    - Keylogger
        - Stakeholder: User
        - Assets: username, password, messages
    - Brute force
        - Stakeholder: User
        - Assets: username, password, messages
- Active Insider
    - Server Access
        - Stakeholder: User
        - Assets: username, password, messages
- Passive Outside

- Man-In-Middle
  - Stakeholder: User
  - Assets: username, password, messages
- Shoulder Peeker
  - Stakeholder: User
  - Assets: username, password, messages

4. Solution

For users to send messages to each other they first must create an account. After account creation, the server will generate a JWT for the user, and the user will also generate their own RSA Public and Private key. Then each user will have to email each other their public key. To send a message the user will have to select the target user username. The message will get encrypted with AES. Then we will use HMAC to generate a tag on the encrypted message. Before sending the message to the server, both AES key and HMAC secret key get encrypted with the receiver public key. The message will then get send to the server as a JSON format to the server with the sender JWT. Then the server will verify that the sender JWT and if it valid it will save the message to the database. To retrieve the message the receiver will user its JWT to request the server if there any message. The server will verify the receiver JWT and if it's valid then the server will query the database if there a message for the user. If there a message for the user, the server will send the message back to the user.

To decrypt the message, the receiver will use their private key to decrypt the encrypted HMAC secret key and AES key. Then the receiver will use the HMAC secret key to generate an HMAC tag. Then the tag gets compare with the one that was send with the message. If the tag is not equal, then the message will be ignored. If it matches, then the message will be decrypted with the provided key.

5. Implementation

5.1. Encapsulation and Decapsulation

- Encapsulation: Encapsulation is important because it keeps the messages sent to and from the user from being read or altered by an adversary.

Before being sent to the server, the client-side application encrypts the message using the RSA public key of the receiver. The message that the user sends is encrypted using AES in CBC mode. The resulting ciphertext is then hashed with HMAC using SHA256 in order to create an integrity tag. After this step, the AES and HMAC keys are concatenated and encrypted using the RSA public key of the receiver. When these steps have been completed, the client side will send the RSA ciphertext, AES ciphertext, and HMAC integrity tag to the server for it to be sent to the intended receiver.

- Decapsulation: Decapsulation allows a user to read the messages sent to them. When the user receives a message that was stored in the server, the message is encrypted to preserve the sender and receiver's confidentiality. In order to read the message, the user must first use their private key to decrypt the RSA ciphertext containing the AES and HMAC keys. Using SHA256 and the HMAC key, the application re-generates the integrity tag. If the integrity tag and AES ciphertext match, then the user can be confident that the message has not been tampered with. If the integrity of the message has been proven, then the application proceeds to decrypt the message using the AES key. When this process finishes, the user will be left with the unaltered plaintext that was sent to them.

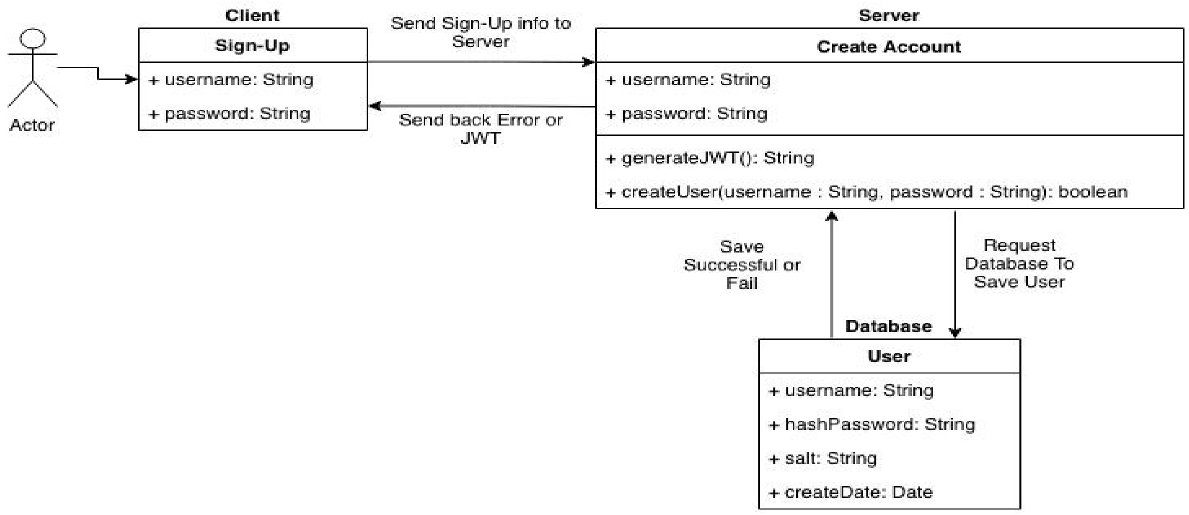
5.2. Key Exchange

Our application requires that users to send their RSA public key to each other through email. The reason why we choose to do it this way because we cannot trust the server to distributes the keys because we are treating it like an adversary. One issue with this is that it is inconvenient for the user because they must figure out the email address of the intended receiver and they must send the email containing the public key.

5.3. Client and Server

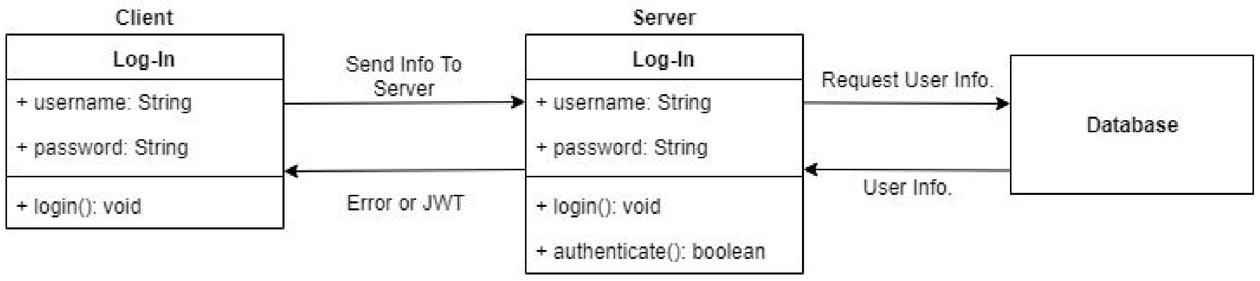
5.3.1. Sign-up

We ask the user to create an account with a username and password to keep the user's account secure. The client prompts the user to enter a username and password and that information is sent to the server via SSL. The server checks its database to see if the username is unique. If the username has already been registered, the server sends back a message requesting a unique username. When the user registers a unique username, the server saves the username and password and returns a confirmation message as well as a JWT to allow the user to send and receive messages from their new account.



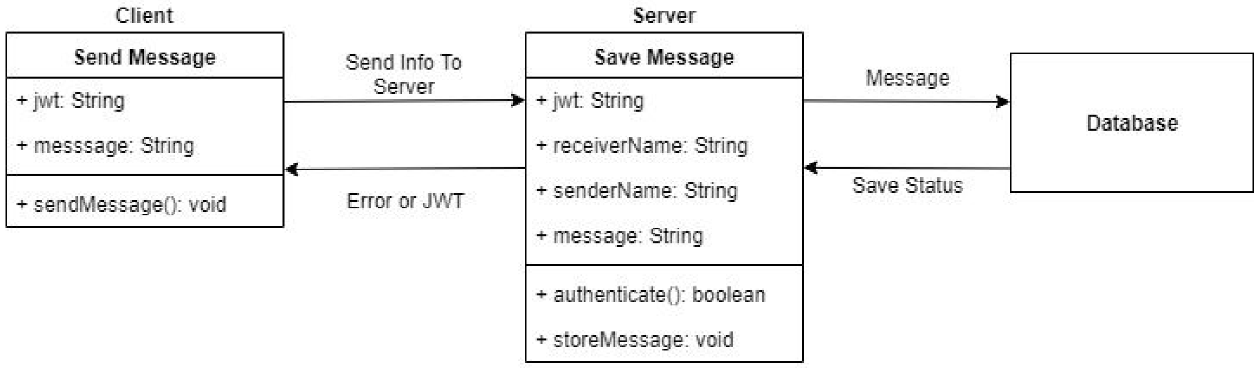
5.3.2. Login

Every time the application is started, a registered user must enter their username and password to gain access to their account. When the user enters their username and password, the client sends that information to the server and the server compares the user input with the information stored in the server's database. If the user input and the username/password combination do not match, the server sends back a message informing the user that their information is not accurate. When the user sends a username and password that matches the server database information, the server sends back a welcome message as well as a JWT to allow the user to send and receive messages from their account until they logout.



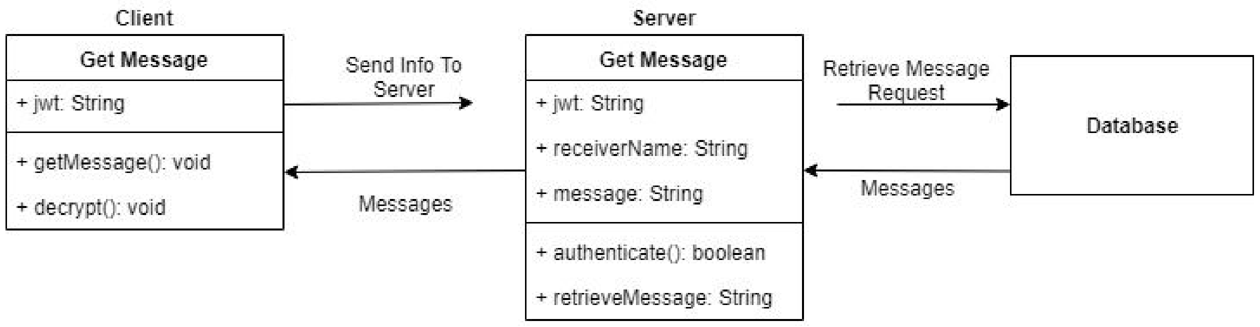
5.3.3. Send Message

Every time that a user sends a message, the application generates an AES key for encryption. Then the application uses the AES key to encrypt the message and CMC mode and IV. The IV will then be concatenate at the beginning of the encrypted message. Next the application will generate a hmac secret key, which will be use HMAC to sign the encrypted message. This will help us detect if the message got modify during transit and use to authenticate that the message was send from that user. Then the AES key get concatenate with the hmac secret key and will be encrypted with the receiver RSA public key. Finally, the client sends the encrypted message, encrypted HMAC and AES key, and hmac signature to the server in as a json format, receiver name, and sender JWT. The server will verify the sender JWT. If valid the server will store the message in database, if it's not valid then the server will send an error back to the client.



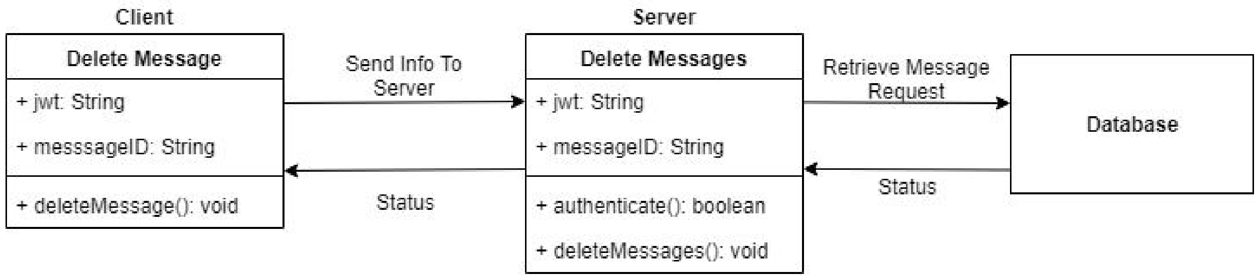
5.3.4. Receive Message

The application will make a get message request to the server every two seconds with their JWT. The server will check if the verify if the JWT is valid or not. If it's not, then the server will ask an error to the user. If it's valid then the server will check if there a message for the user in the database. If there no message, then it will do nothing. If there's message, then the server will send that back to the client. The client first step is to decrypt the encrypted AES key and hmac key with the user private key. Then run hmac to on the encrypted message. After the application will check if the result of the hmac that it ran match the hmac that was send with the message. If it matches, then the client will decrypt the message. If it does not, then the application will just ignore the message and move on. By checking the hmac signature first allows the application to save resources if the message was modified.



5.3.5. Delete Message

Every time the client receives a message from the server, it will ask the server to delete the message from the database. The client will send the server its JWT and the message ID number. The server will then check if the user has the proper authorization the delete the message. If the user does, it will delete the message from the database using the given messageID.



6. Weaknesses and Vulnerabilities

Most of the weaknesses in our system are based on the user-experience. Because our current implementation requires perfect key naming conventions, we are putting a lot of trust in the user and forcing them to create their own keys with correct names. We are also putting key exchange in the hands of the user. It can be inconvenient for the user to have to create their own keys as well as send them to their intended users.

While our implementation addresses a large amount of threats, there are some adversary methods that we currently cannot defend against. We currently do not have any way to defend or server against a DOS or DDOS attack. Also, while we are using 2048-bit keys for encryption and a 200-bit salt for hashing to combat against brute force attacks, we are unable to defend against a very high-powered adversary.

7. Future Work

- Implement Secure Remote Login
- Automate email process of public key exchange
- Group chat
- Perfect forward secrecy
- Better GUI interface
- Defense against impersonation attack
- Auto-generate key pair and personal keys folder when a user signs up
- Digital Signature

References:
Image 1: http://clipart-library.com/clipart/database-free-download-png.htm
Image 2: https://blogs.technet.microsoft.com/uktechnet/2017/01/17/how-to-boost-your-windows-server-2016-security/
Image 3: https://upload.wikimedia.org/wikipedia/commons/1/1a/Crystal_Project_computer.png