CECS 478 End-to-end Encrypted Chat

Team Skynet

Ben Koch (official team Art Designer) and Nhi Nguyen

Application Properties

Our chat application will allow two users to initiate an end-to-end encrypted chat. The users can add find other based on username and will be able to have a secure chat that only they can read. Users do not have to be signed in at the time a message is sent to them. They can simply request from the database any messages they have yet to receive.

The backend will be written in Ruby on Rails and hosted on Heroku. It will be a web application accessible from any device with a simple interface that allows users to login, add a user to chat with, open an active chat, and send messages in those chats. Ruby on Rails has many plugins that will help us create the application. We will use Ruby on Rails to create a RESTful web server that serves JSON to the client and allows users to insert, delete, and update messages in the database.

Assets / Stakeholders

Assets: User identity, chat messages and the information they contain.

Stakeholders: Application users.

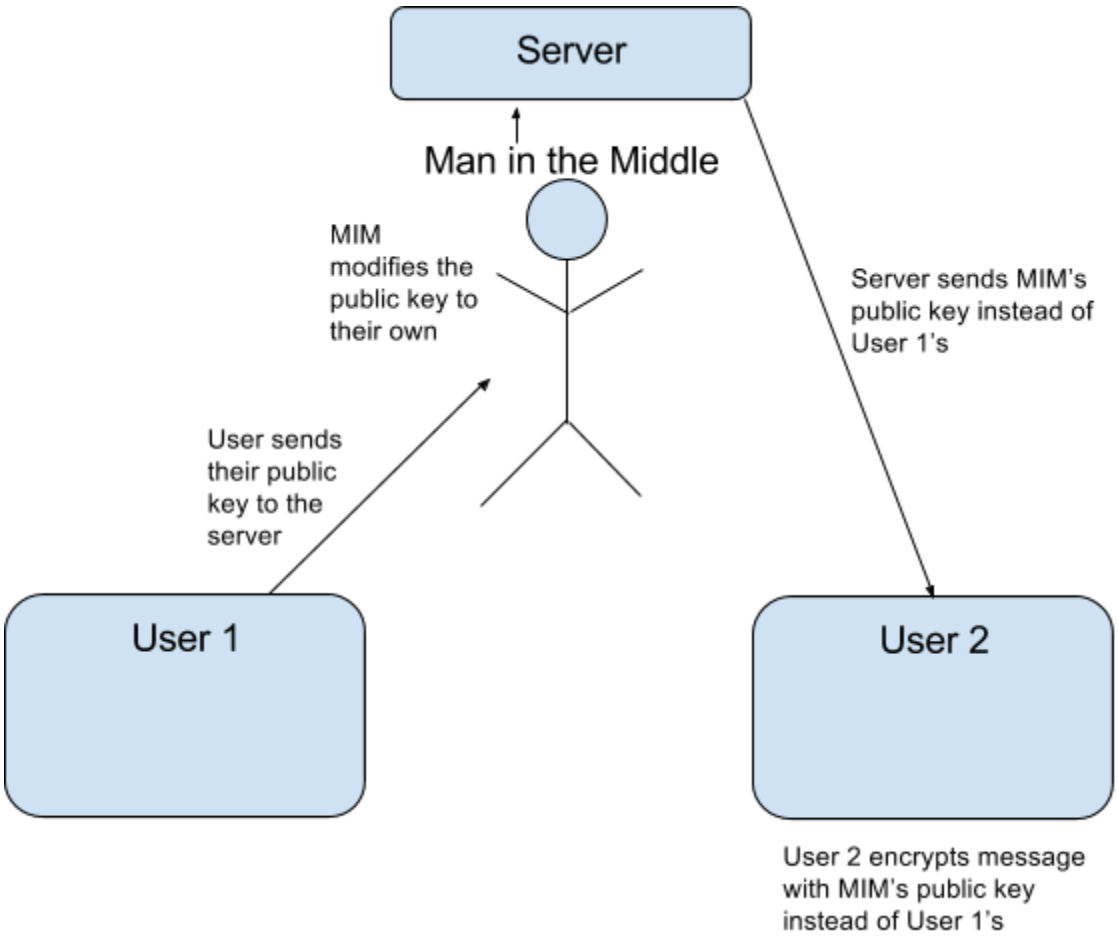
Adversarial Model

Outsiders: Outsiders can attack us in two ways: a Man-in-the-Middle attack from … the outside, or attempt to actually gain physical access to user devices (refer to Possible Vulnerabilities). Our assets should be protected from Man-in-the-Middle attacks so long as our PGP encryption holds up. Physical access attacks are simply impossible to address.

Insiders: Insiders will also attempt Man-in-the-Middle attacks. Should they gain access to user encrypted messages, they should not be able to decrypt them since the server itself is not privy to any user private keys.

Possible Vulnerabilities

Man-in-the-Middle Attacks: Our end-to-end encryption ensures that user data is securely encrypted and transferred between endpoints. The transfer process through the server is not completely secure, leaving the door open for eavesdroppers to impersonate either the recipient or sender through key exchanges or substituting user public keys with their own.

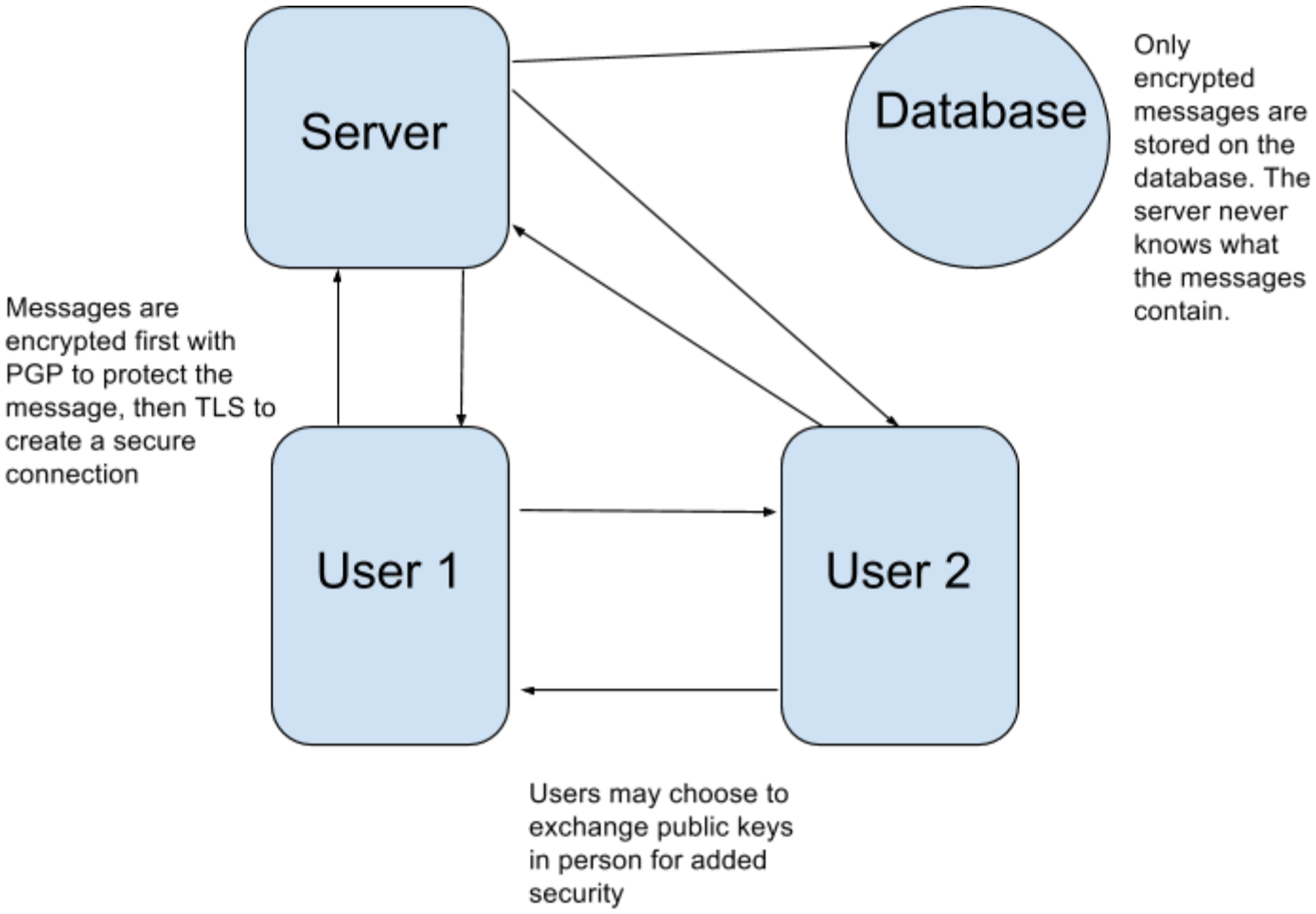


Physical Access: Our end-to-end encryption cannot directly address the risks of the communication endpoints (sender/receiver's devices) being hacked or their encryption keys stolen (or the decrypted messages simply read).

Solution / Analysis

- We will use the Devise plugin for rails which allows users to login with a username and encrypted password.
- We will use TLS 1.2 to encrypt all traffic between users and the server.
- We will implement PGP encryption to secure the messages from man-in-the-middle attacks. Each time a message is to be sent, we will generate two 256-bit AES keys. The first is to encrypt the message and the second is to ensure data integrity with HMAC.
- These two keys are concatenated together and encrypted using the receivers public key and sent to the user along with the encrypted message. The receiver will then decrypt the keys with their private key which they can then use to decrypt the message. The server will not be trusted at any time and will have no way of knowing what the users are saying.
- The keys are generated based on the user's username and password. Only the user will have the private key. The public key will be sent to the server.
- The public key may still be vulnerable to a man-in-the-middle attack if the server is compromised. Because of this, we will provide users the option to not have their public keys stored on the server, in which case they will have to exchange public keys in person or by another communication method and input them manually into the chat application.

Project Diagram



Use Case Diagram