ChatterBox

Jared Rathbun, Alexander Royer, and Camron Chrissis

What is ChatterBox?

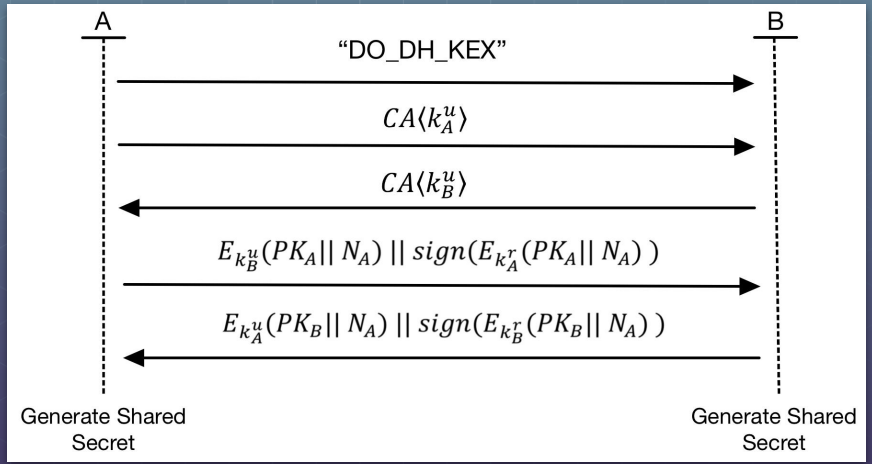
- ChatterBox is a group-chatting software, allowing multiple users to communicate simultaneously in a confidential and secure fashion.
- Users can create an account, reset their password, and login using a convenient and modern Java Swing GUI (Graphical User Interface).
- Each message is displayed in a neat fashion to every member of the group chat with a timestamp and their username.
- Requires 2-step authentication using HOTP (HMAC-Based One-Time-Password).

Our Security Protocol

- We decided to implement the use of hybrid encryption. Asymmetric Cryptography is used to transmit information such as keys and Symmetric Cryptography is used for the transmission of other information.
- In order to do this, we used RSA in ECB (Electronic Code Book) mode to serve as our Asymmetric Cipher, and AES with AEAD (Authenticated Encryption with Additional Data) as our Symmetric Cipher.
- To implement our AES Cipher, we used GCM (Galois Counter Mode), which allows us to verify the integrity of the message without including something like a signature or MAC tag.

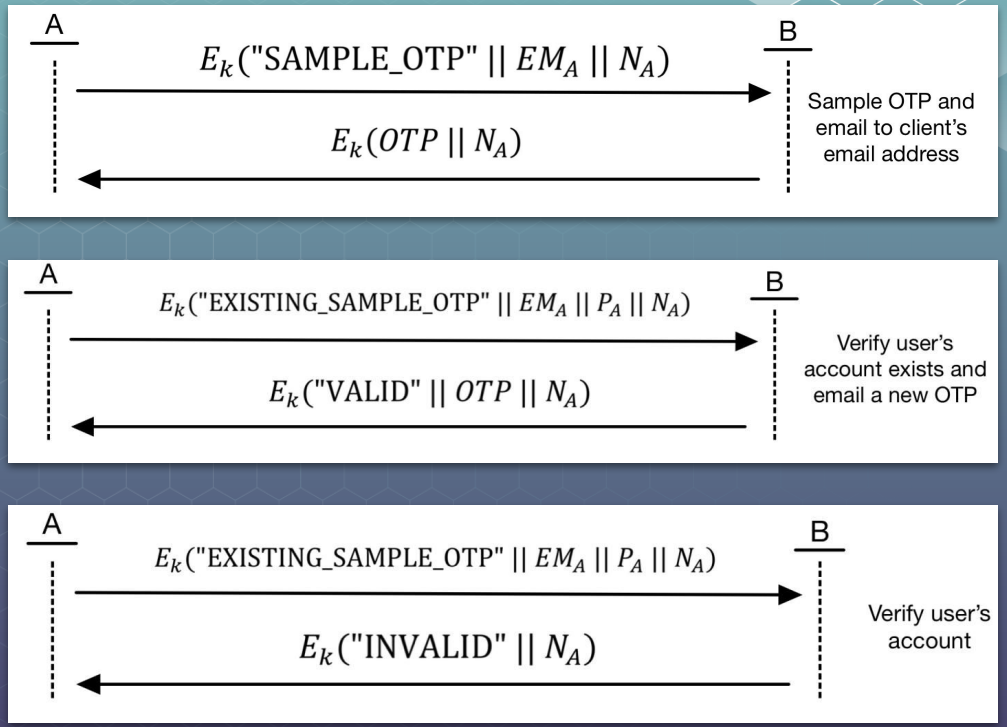
Client–Server Diffie–Hellman Key Exchange

- When a user connects to the server, the client sends a message notifying the server that it will be starting a Diffie-Hellman Key Exchange. An exchange of certificates is then completed. Specifically, it will be using Elliptic-Curve keys in the process.
- Using RSA and an encrypt-then-sign technique, the client would send it's public Elliptic-Curve key to the server and wait for the server's public Elliptic-Curve key. Once completed, both parties can compute the shared key and create an AES key from the keying material.
- Graphically, this would look like:



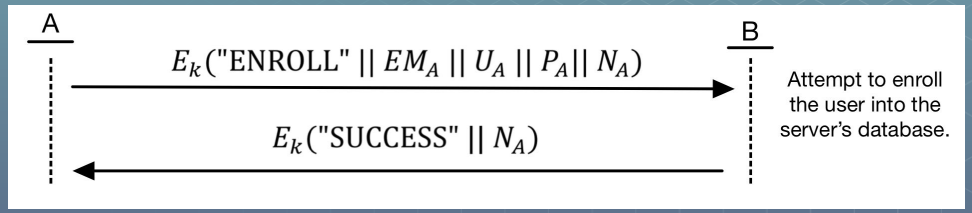
Sampling a One-Time Password

- When the client attempts to create an account, reset their password, or login the user will be prompted to enter an email address and is then prompted for an OTP.
- Depending on if they have an account or not, the message to the server will include an email address and password, or just an email address.

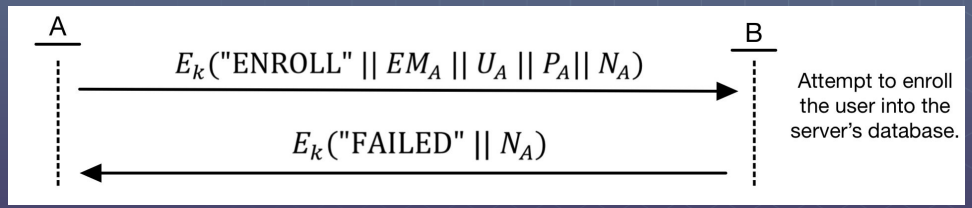


Enrolling a User into the Database

- The client will send “ENROLL” to the server along with an email address, username, password, and a nonce.
- If the server detects valid credentials from the database, the user will receive “SUCCESS” along with a nonce.

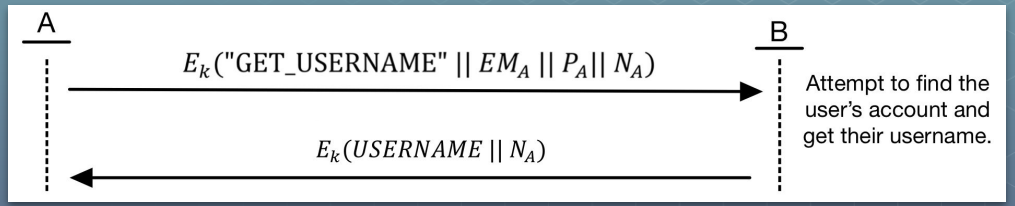


- If the server does not detect valid credentials from the database, the client will receive “FAILED” along with a nonce

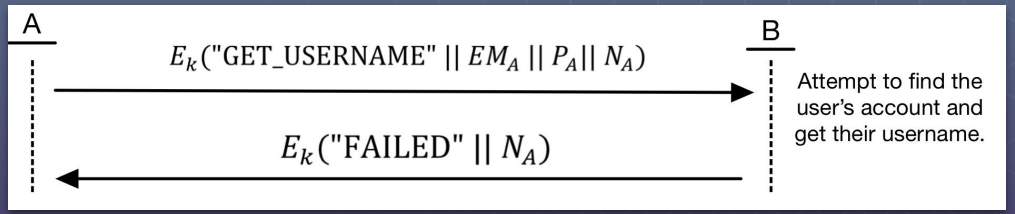


Requesting a Username

- The client will send “GET_USERNAME” to the server along with an email address, password, and a nonce.
- If the server detects valid credentials from the database, the user will receive the USERNAME along with a nonce.

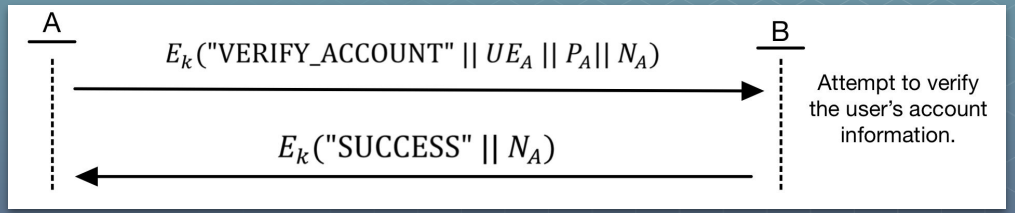


- If the server does not detect valid credentials from the database, the client will receive “FAILED” along with a nonce

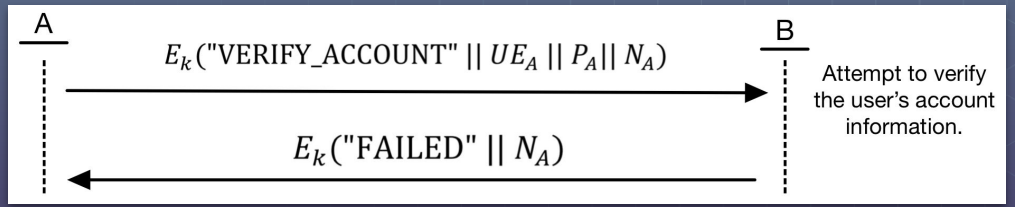


Verifying a User's Account Information

- The client will send "VERIFY_ACCOUNT" to the server along with an email address, password, and a nonce.
- If the server detects valid credentials from the database, the user will receive "SUCCESS" along with a nonce.

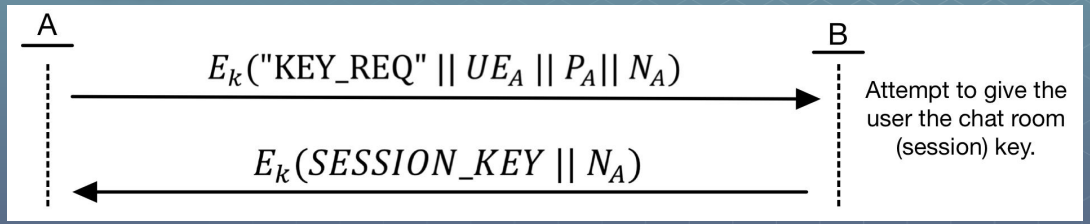


- If the server does not detect valid credentials from the database, the client will receive "FAILED" along with a nonce

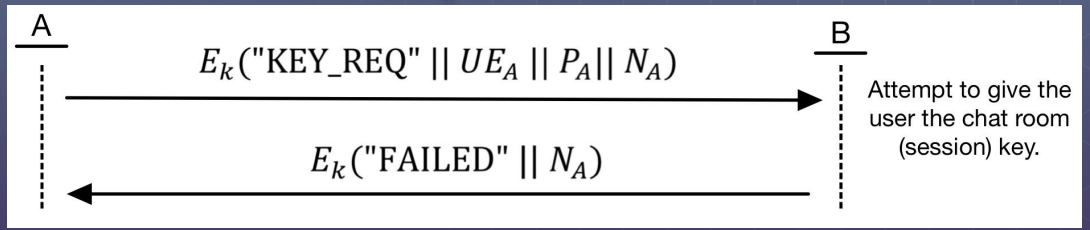


Requesting the Session Key

- The client will send “KEY_REQ” to the server along with a username or email address, password, and a nonce.
- If the server successfully authenticates the user’s credentials, the user will receive the chat room key “SESSION_KEY” along with a nonce.



- If the server cannot authenticate the user’s credentials, the client will receive “FAILED” along with a nonce



Drawbacks

- We did not implement a feature to refresh the Session Key after a certain number of uses. This could lead to something such as a Birthday Bound Attack.

- We did not implement a feature for the user to obtain or reset their username.

credit to https://vincentdnl.com/programming-memes/

Struggles we Came Across

- How should the server keep track of people's keys so that it can respond to the correct person efficiently?

  - SOLUTION: Assign each user a UUID (Unique User Identifier) and use a HashMap to map each UUID to the correct AES key.

- How will the server know when someone is in the chatroom or not so that the chat messages can be distributed to them?

  - SOLUTION: Using a thread pool, assign each thread a "state"; 0 standing for in communication with the server and not actively in the chatroom, and 1 being in the chatroom. (Thanks Prof. K!)

Parts of The JCA We Learned

- We utilized a class in the Java Cryptography Architecture known as a SealedObject. It's constructor takes two parameters, one being a Serializable object, and the other being a Cipher object. The SealedObject class encrypts the Serializable object for you.

- We created a class called SealedMessage which inherited SealedObject's properties. This allowed us to send encrypted ServerMessages or ChatMessages without having to worry about encrypting objects manually. This meant we could simply send the SealedMessage over an ObjectOutputStream or ObjectInputStream. The constructor looks like this:

```
public class SealedMessage extends SealedObject implements Serializable
{
    private final int UUID;
    private final byte[] IV;
    private static final long serialVersionUID = -6743567631108323096L;
    private final int TYPE;

    public SealedMessage(Serializable data, Cipher cipher, final int UUID,
            byte[] iv) throws IOException, IllegalBlockSizeException
```

DEMO!

THANKS!