Overview

An encryption library (such as CryptoKit), specifically tailored for quantum-resistant encryption algorithms.

Description

With the advent of Quantum Computers, the field of cryptography is undergoing a generational shift. Many traditional encryption algorithms can be easily cracked by quantum algorithms and this poses significant risk to systems (ranging from financial services to government data) around the world, once quantum computers are widespread. Thankfully, there exist a number of encryption algorithms designed to be “quantum-resistant”. These algorithms tend to be mathematically more involved than traditional encryption algorithms, and are thus a good candidate for leveraging the strengths of OCaml.

Our aim is to design a secure chatting application that uses one such Quantum-Resistant algorithm called Kyber, which was one of the first four algorithms that were the winners of NIST’s 2022, 6-year long competition.

Algorithm

Key Features:

Kyber relies on the Learning With Errors (LWE) problem, specifically a variant called **Module Learning With Errors (MLWE)**. The hardness of this problem forms the foundation of its security. It can be used for public-key encryption (to encrypt messages) and key encapsulation (to securely exchange keys). It is particularly well-suited for hybrid cryptographic protocols where quantum-safe key exchange is required. Kyber is designed to be efficient in both computational and communication aspects. It has fast key generation, making it well-suited for practical use, even on resource-constrained devices.

High-Level Workflow of Algorithm:

1. **Key Generation**: A pair of keys (public key and private key) is generated. The public key is derived using the **MLWE** problem, ensuring it is hard to reverse-engineer without the private key.
2. **Key Encapsulation**: The sender uses the recipient’s public key to encrypt a symmetric shared key, which is shared with the recipient. The encryption process involves introducing random noise based on the LWE problem, ensuring the ciphertext is indistinguishable from random data.
3. **Key Decapsulation**: The recipient uses their private key to decrypt the message and recover the encapsulated shared key.

4. **Message Encryption and Decryption**: All future communication between the sender and recipient is encrypted using the shared key that both are aware of, using a symmetric-key based algorithm, which in our case is AES-GCM, found in the MirageCrypto library.

For a more detailed description of the algorithm, refer to the research paper indexed **1** in our references.

Chat Application

User List Page

Acts as a landing page where users can see a list of available users to chat with.

- **Search Bar**:
  - Allows searching for users by their name.
  - Dynamically filters the user list as the search term is typed.
  - Supports case-insensitive matching.
- **User List**:
  - Displays all users with basic information
  - Clicking a user opens the chat interface for that user.
- **Error/Empty States**:
  - If no users are found matching the search term, display "No users found."
  - If there are no users in the list, show "No users available."

Messaging Interface

Enables users to chat with another selected user.

- **Chat Header**:
  - Displays the name of the user being messaged.
  - Includes a "Back" button to return to the user list page.
- **Message Display Area**:
  - Shows the current session's chat history in chronological order.
  - Displays messages with:
    - The sender’s name.
    - Message text.
    - Timestamp (maybe).
- **Message Input**:
  - A text input box for composing messages.
  - A "Send" button to submit the message.
  - Supports multiline messages
  - Automatically clears the input field after sending.
- **Temporary Chat**:

    - No message persistence: messages disappear when the app is closed or the user navigates away.
- **Error Handling**:
    - Prevent sending empty messages or exceeding a character limits

Components:

1. UserListComponent
Purpose: Display the list of users and allow searching for a user to start a chat.
Functions:

- fetch_users
- filter_users
- On_user_select

2. SearchBarComponent
Purpose: Handle user input for searching through the user list.
Functions: No standalone functions; it simply passes the search query back to UserListComponent.

3. ChatComponent
Purpose: Manage the chat session for the selected user, including sending/receiving messages.
Functions:

- establish_secure_connection
- load_chat_session
- Append_to_chat_session

4. MessageInputComponent
Purpose: Provide a text input area for typing and sending messages.
Functions:

- send_message
- encrypt_message

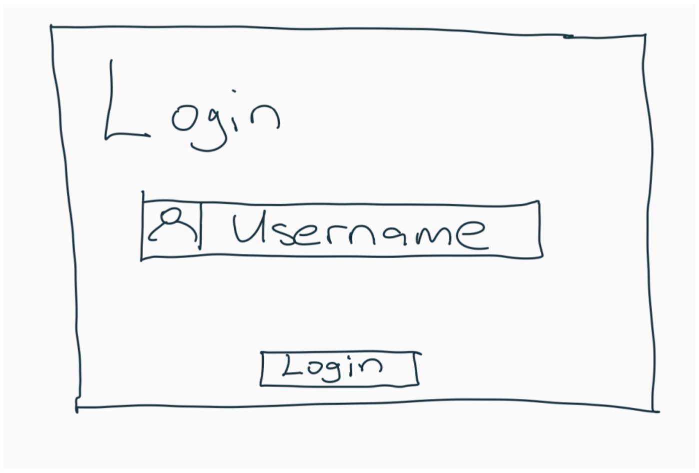
5. MessageListComponent
Purpose: Display the chat messages for the current session.
Functions:

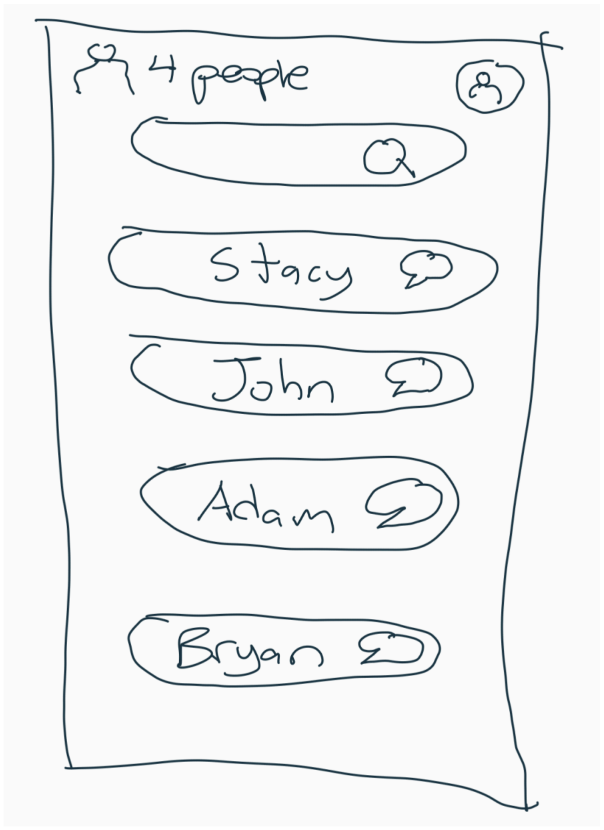
- decrypt_message
- render_message

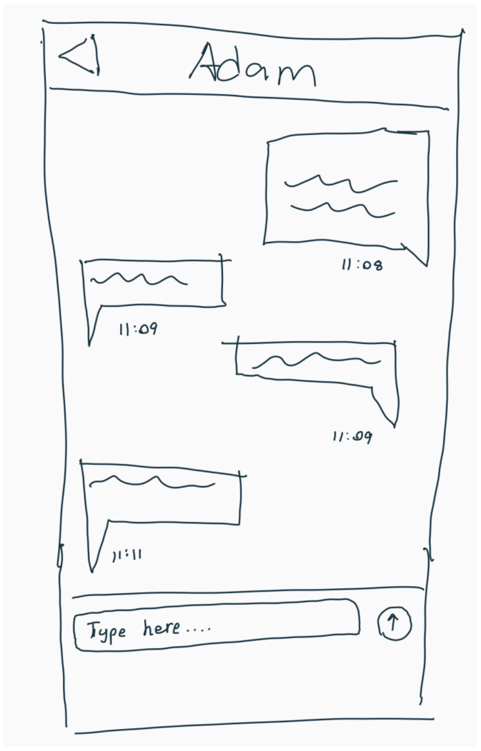
6. HeaderComponent
Purpose: Display the recipient's name and provide a "Back" button.

How a user can interact with our app:

1. The user will log in with a username and upon entering the app will be presented with a list of available users
2. There will be a search bar at the top that allows the user to filter the list by username.
3. The user can click on a username to start a chat that will establish a secure connection using our Kyber algorithm.
4. Once the connection is established, the chat interface will appear
5. The user can then type messages in the input box and send them where the messages will be encrypted with the shared secret and have to be decrypted. The messages will appear in the chat history.
6. Any received messages will also appear in the chat history
7. The user can press the back button to return to the user list and all messages will be cleared
8. The user can search for another user and start a new chat.







Libraries used

Core, Owl, Polynomial, ff (finite fields), Zarith, MirageCrypto, Rescript

Tentative .mli files

https://github.com/ameya-dehade/ocaml-quantum-encryption/tree/main/src/tentative-mli-files

Implementation Plan

Week 1:

1. Implement polynomial and polynomial vector functions
2. Implement randomness generators
3. Set up development environment for front end
4. Create a userlist component displaying static list of users
5. Implement search bar for filtering list

6. Set up basic navigation from user list to placeholder chat interface

Week 2:

1. Implement hash functions and param set up
2. Implement Kyber utilities and helper functions
3. Set up chat ui - chat component with messageListComponent and messageInput component
4. Implement chat functions
5. Temporary chat history functionality

Week 3:

1. Implement the actual Kyber algorithm using all the modules and functions created before
2. Key management functions
3. Integrate sendMessage and receiveMessage to encrypt and decrypt the messages
4. Implement startChatSession and endChatSession for key exchange and session cleanup

Week 4:

5. Test Kyber and the Frontend thoroughly
6. Debug!

Potential future features:

1. Sending files (would involve conversion of files of different formats, to and from byte representation
2. Persistent chats (would involve use of some sort of database)
3. Forwarding/Announcements (would involve advanced behavior on the frontend to allow mass creation of shared keys and aggregated delivery of a message)

References

1. https://eprint.iacr.org/2017/634.pdf