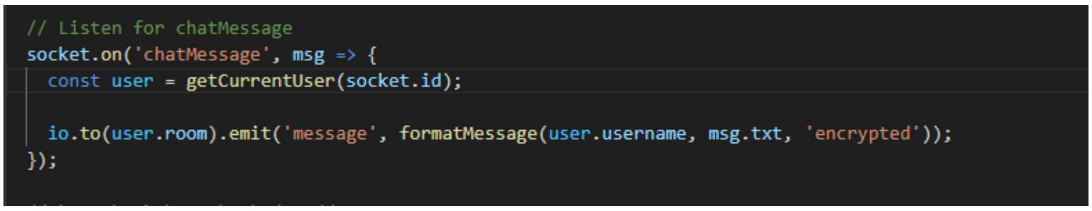
Róisín Ní Bhriain - 18326577

Secure Social Media Application

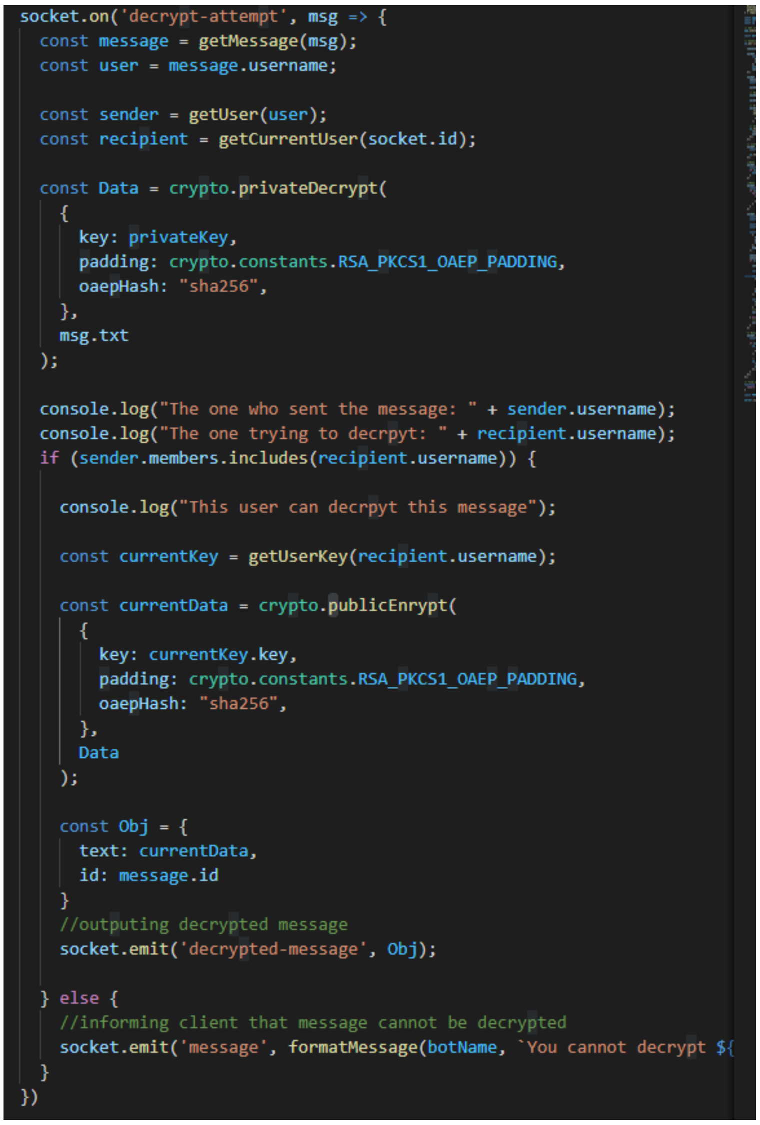
Encryption

The encryption library I used was the crypto library in javascript. I decided to use RSA encryption. I chose it as I wanted an asymmetric method of encryption as this is more secure than using a symmetric encryption.



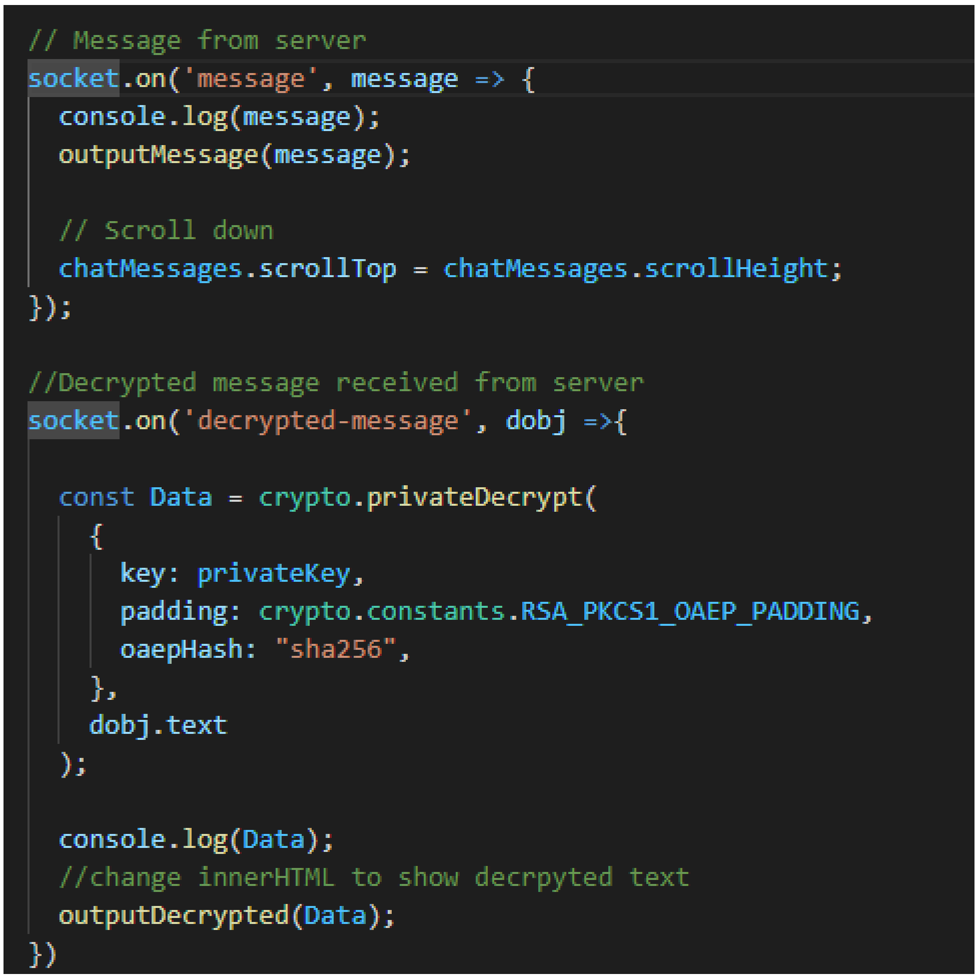
```
// Listen for chatMessage
socket.on('chatMessage', msg => {
  const user = getCurrentUser(socket.id);

  io.to(user.room).emit('message', formatMessage(user.username, msg.txt, 'encrypted'));
});
```

When the server receives a message it initially emits the message to all users in the room encrypted. This ensures that any user who is not in the secure group cannot decrypt the message as they do not have the correct key.



```
socket.on('decrypt-attempt', msg => {
  const message = getMessage(msg);
  const user = message.username;

  const sender = getUser(user);
  const recipient = getCurrentUser(socket.id);

  const Data = crypto.privateDecrypt(
    {
      key: privateKey,
      padding: crypto.constants.RSA_PKCS1_OAEP_PADDING,
      oaepHash: "sha256",
    },
    msg.txt
  );

  console.log("The one who sent the message: " + sender.username);
  console.log("The one trying to decrpyt: " + recipient.username);
  if (sender.members.includes(recipient.username)) {

    console.log("This user can decrpyt this message");

    const currentKey = getUserKey(recipient.username);

    const currentData = crypto.publicEnrypt(
      {
        key: currentKey.key,
        padding: crypto.constants.RSA_PKCS1_OAEP_PADDING,
        oaepHash: "sha256",
      },
      Data
    );

    const Obj = {
      text: currentData,
      id: message.id
    }
    //outputing decrypted message
    socket.emit('decrypted-message', Obj);

  } else {
    //informing client that message cannot be decrypted
    socket.emit('message', formatMessage(botName, `You cannot decrypt ${
  }
})
```

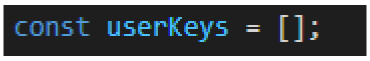
A user will attempt to decrypt a message and this will be sent to the server. The server will then decrypt the current message using its private key and then if the user is in the secure group it will encrypt the message using the public key of the user that requested the decrypted message.



```
// Message from server
socket.on('message', message => {
  console.log(message);
  outputMessage(message);

  // Scroll down
  chatMessages.scrollTop = chatMessages.scrollHeight;
});

//Decrypted message received from server
socket.on('decrypted-message', dobj =>{

  const Data = crypto.privateDecrypt(
    {
      key: privateKey,
      padding: crypto.constants.RSA_PKCS1_OAEP_PADDING,
      oaepHash: "sha256",
    },
    dobj.text
  );

  console.log(Data);
  //change innerHTML to show decrpyted text
  outputDecrypted(Data);
})
```

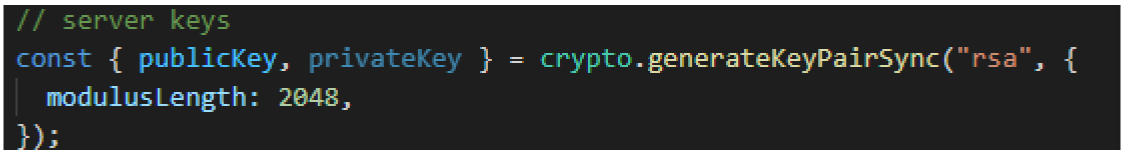
When the user receives a message it will display that encrypted message on the screen. If the user attempts to decrypt a message, and they are in the secure group the server will decrypt the message using its private key and then encrypt the message with the specific user's public key. Then the user can decrypt the message using its private key.

Key Management System



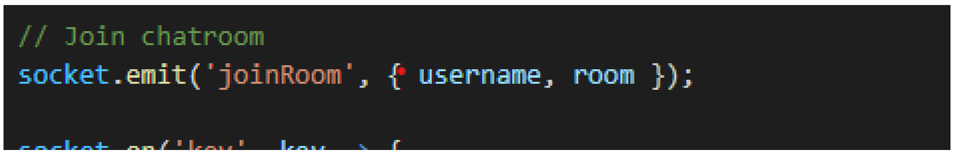
```
const userKeys = [];
```

The Key Management System I used was that of a list of public keys in the server.



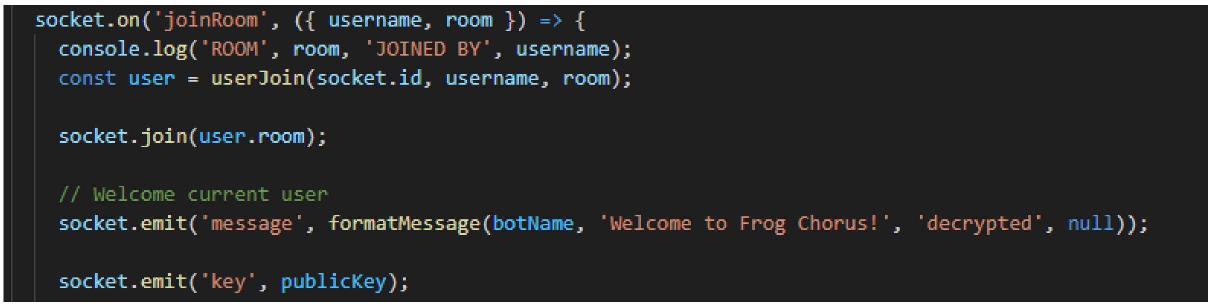
```
// server keys
const { publicKey, privateKey } = crypto.generateKeyPairSync("rsa", {
  modulusLength: 2048,
});
```

The server also had its own public key and private key generated upon initialisation. These keys were also generated in the same way in the user code.



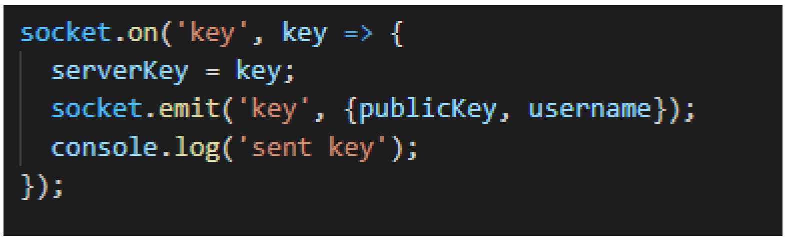
```
// Join chatroom
socket.emit('joinRoom', { username, room });
```

The user will join the room upon initialisation.



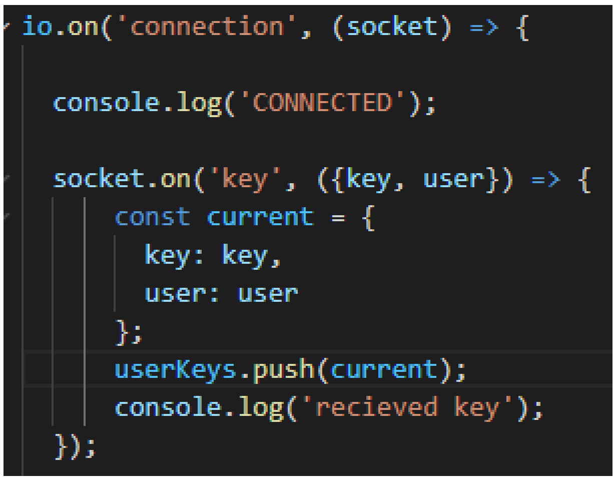
```
socket.on('joinRoom', ({ username, room }) => {
  console.log('ROOM', room, 'JOINED BY', username);
  const user = userJoin(socket.id, username, room);

  socket.join(user.room);

  // Welcome current user
  socket.emit('message', formatMessage(botName, 'Welcome to Frog Chorus!', 'decrypted', null));

  socket.emit('key', publicKey);
```

After a user joins the room the server will send the welcome message that everyone receives. Then the server sends its public key to the user.



```
socket.on('key', key => {
  serverKey = key;
  socket.emit('key', {publicKey, username});
  console.log('sent key');
});
```

The user will receive the key and store it as a local variable for use later. It then sends its own user name and own public key to the server.

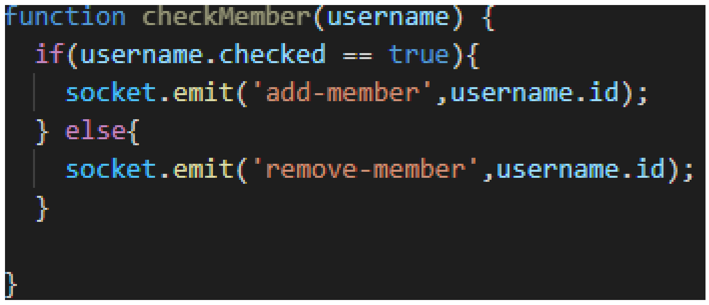


```
io.on('connection', (socket) => {

  console.log('CONNECTED');

  socket.on('key', ({key, user}) => {
    const current = {
      key: key,
      user: user
    };
    userKeys.push(current);
    console.log('recieved key');
  });
```

The server will receive the key and store it in a list with all the other public keys of the users.

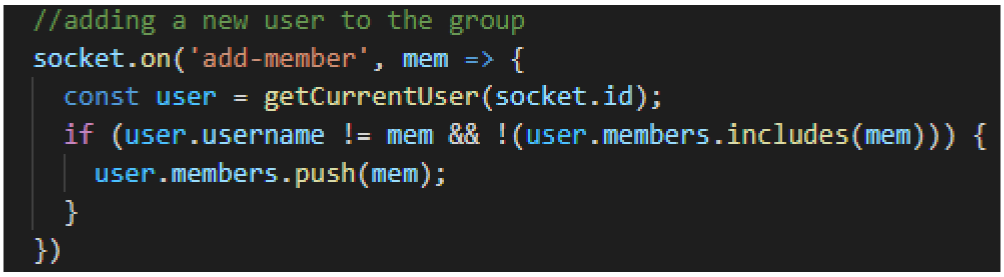
Adding Or Removing Members

There is a checkbox in the chat page for each user which allows them to add and remove members from the secure group in the room.



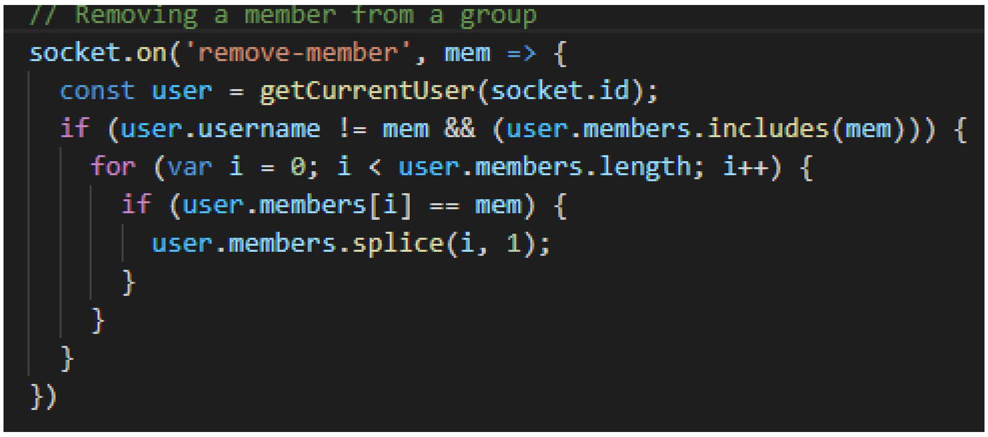
```
function checkMember(username) {
  if(username.checked == true){
    socket.emit('add-member',username.id);
  } else{
    socket.emit('remove-member',username.id);
  }

}
```

The user can either check or uncheck a box and if that happens then a user will send a message to the server to either add a member or remove a member from the list of members in a room of that user.



```
//adding a new user to the group
socket.on('add-member', mem => {
  const user = getCurrentUser(socket.id);
  if (user.username != mem && !(user.members.includes(mem))) {
    user.members.push(mem);
  }
})
```

If a new member is added by a user then it is added to the list of members for that user.

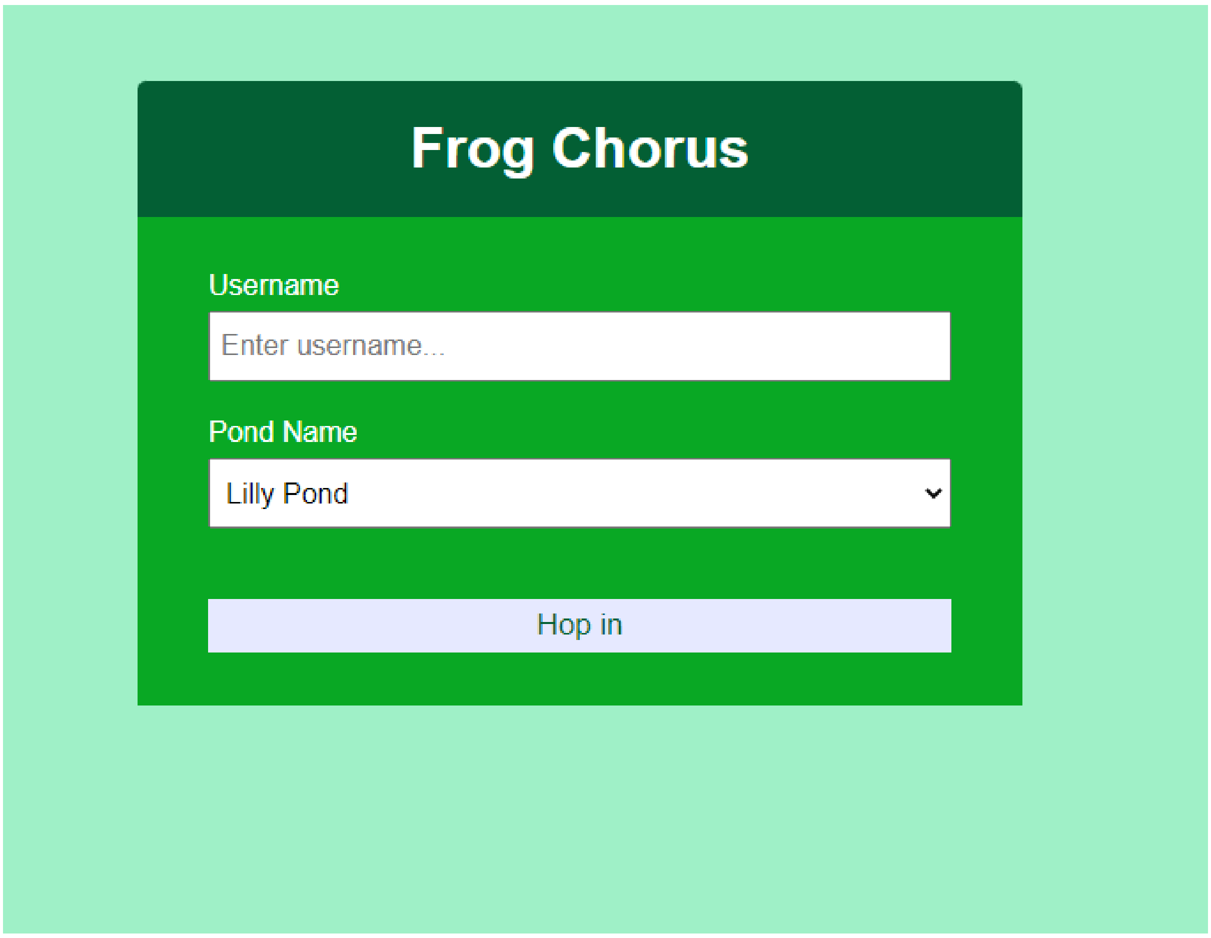


```
// Removing a member from a group
socket.on('remove-member', mem => {
  const user = getCurrentUser(socket.id);
  if (user.username != mem && (user.members.includes(mem))) {
    for (var i = 0; i < user.members.length; i++) {
      if (user.members[i] == mem) {
        user.members.splice(i, 1);
      }
    }
  }
})
```

If a member is removed from the secure group and the user is both in the group and not the user trying to remove the member. Then the server will search for that member and remove it from the list.

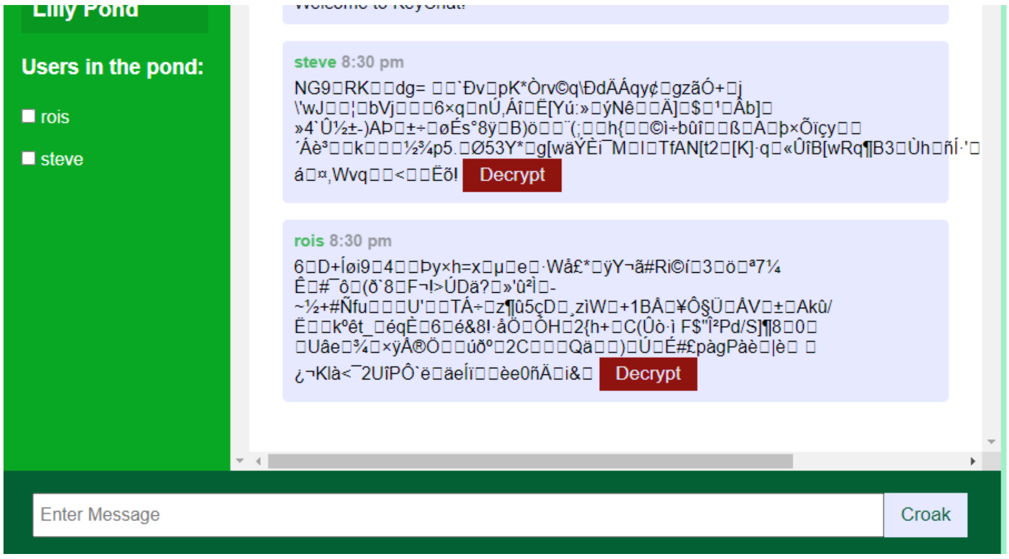
A Look At The Interface

The Login Page



The user must type a username and choose a room.

Encrypted Messages



steve 8:30 pm

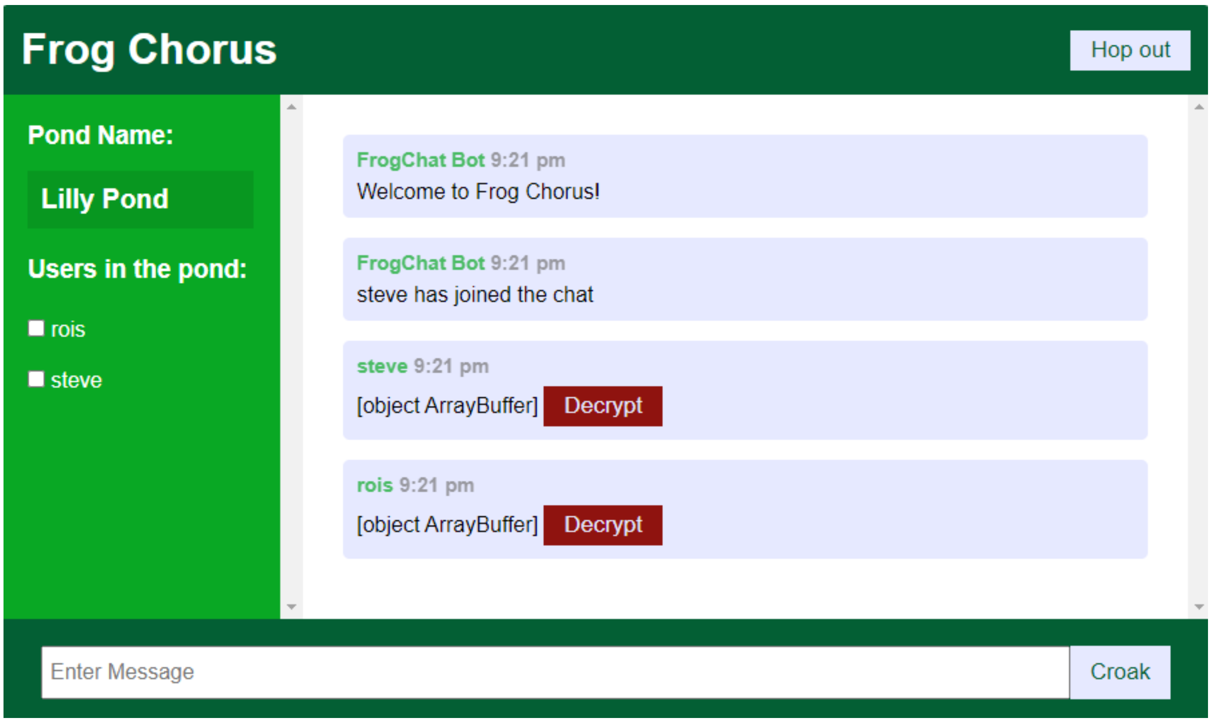
```
NG9□RK□□dg= □□`Ðv□pK*Òrv©q\ÐdÄÁqy¢□gzãÓ+□j
\'wJ□□¦□bVj□□□6×q□nÚ,Áî□Ë[Yú:»□ýNê□□Ä]□$□¹□Åb]□
»4`Û½±-)AÞ□±÷□øÉs°8ÿ□B)ö□□¨(;□□h{□□©ì÷bûî□□ß□A□þ×Õïçy□□
´Áè³□□k□□□½¾p5.□Ø53Y*□g[wäÝÈï¯M□I□TfAN[t2□[K]·q□«ÛîB[wRq¶B3□Ùh□ñÍ·'□
á□¤,Wvq□□<□□Ëõ!
```

rois 8:30 pm

```
6□D+Íøi9□4□□Þy×h=x□µ□e□·Wå£*□ÿY¬ã#Ri©í□3□ö□ª7¼
Ê□#¯ô□(ð`8□F¬!>ÚDä?□»'û²Ì□-
~½+#Ñfu□□□U'□□TÁ÷□z¶û5çD□¸zìW□+1BÅ□¥Ô§Ü□ÅV□±□Akû/
Ë□□kºêt_□éqÈ□6□é&8!·åÖ□ÒH□2{h+□C(Ûò·ì F$"Î²Pd/S]¶8□0□
□Uâe□¾□×ÿÅ®Ö□□úðº□2C□□□Qä□□)□Ú□É#£pàgPàè□|è□ □
¿¬Klà<¯2UîPÔ`ë□äeÍï□□èe0ñÄ□i&□
```

As you can see here nobody is in the secure group so the messages are all encrypted. These are in Array buffers which look like this.

Róisín Ní Bhriain - 18326577



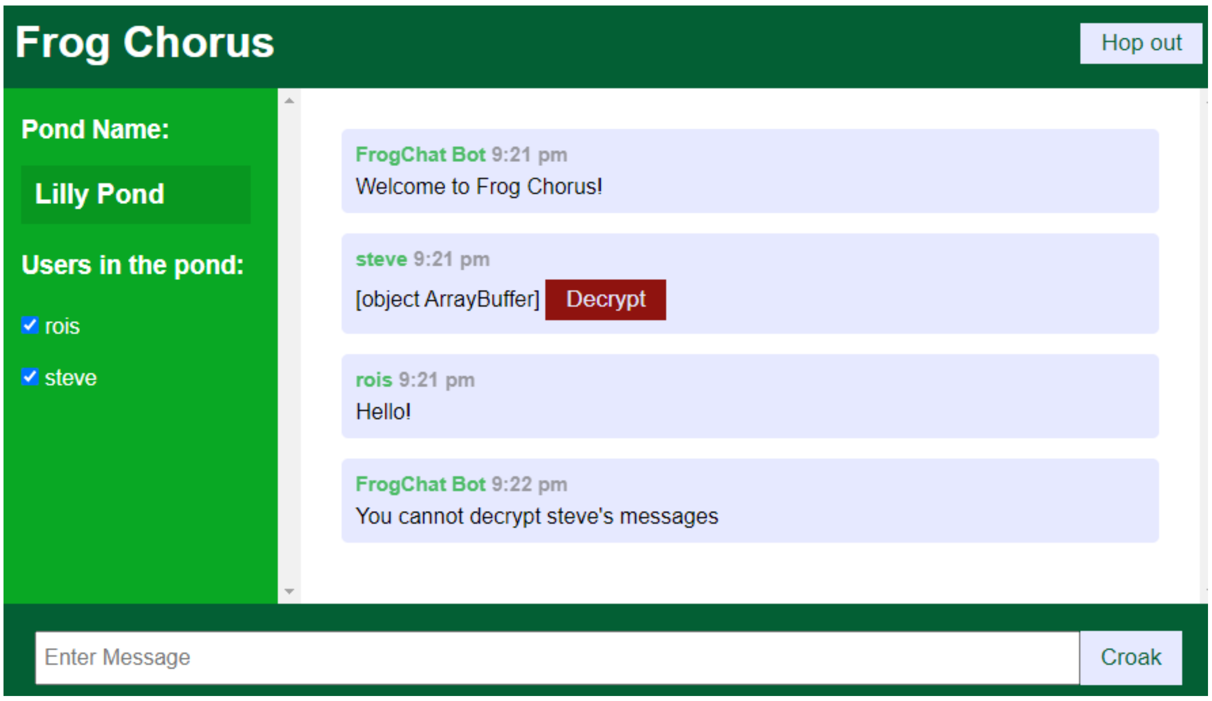
FrogChat Bot 9:21 pm
Welcome to Frog Chorus!

FrogChat Bot 9:21 pm
steve has joined the chat

steve 9:21 pm
[object ArrayBuffer]

rois 9:21 pm
[object ArrayBuffer]

Decrypted Messages



FrogChat Bot 9:21 pm
Welcome to Frog Chorus!

steve 9:21 pm
[object ArrayBuffer]

rois 9:21 pm
Hello!

FrogChat Bot 9:22 pm
You cannot decrypt steve's messages

This user Steve cannot decrypt his own messages but he can decrypt the other user rois as they are in the secure group. The checkboxes that add and remove to the secure group in the room are in blue.