Security and Privacy Project 2 Report

Name:

Student ID:

1 Introduction
2 Implementation
  2.1 User Management
    2.1.1 Register
    2.1.2 Login
    2.1.3 Token
    2.1.3 Logout
    2.1.4 Modify Username and Password
  2.2 Secure Communication
    2.2.1 Private Chat
    2.2.2 Group Chat
    2.2.3 File Transfer
3 User Interface
  3.1 Register, Login and Logout
  3.2 Modify Personal Information
  3.3 Private Chat
  3.4 Group Chat

1 Introduction

In this project, I built an end-to-end encrypted secure communication application named WebChat, which uses the RSA algorithm to achieve secure communication between two users, and uses the DES algorithm to achieve encrypted chats between users in groups. WebChat also has a user management system which allows users to register, log in, log out, modify user names and modify passwords.

It adopts a development model that separates the front and back ends. The front end uses the most popular framework Vue.js, and the back end uses the Python lightweight server framework flask. Websocket is choosed ad the front-end and back-end communication protocol, and it allows the front-end and back-end real-time two-way uninterrupted data transmission. At the same time, token is adopted to determine the user's status and permissions, which greatly improves the security of the system.

2 Implementation

2.1 User Management

The following code shows the user table in the database. WebChat will store the user's id, username, password, public_key, and a boolean value to determine the user's online status.

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(16), index=True, unique=True)
    password_hash = db.Column(db.String(128))
    public_key = db.Column(db.String(1024))
    online = db.Column(db.Boolean, default=False)
```

```
    def __repr__(self):
        return '<User {}>'.format(self.username)
```

2.1.1 Register

The `verifyUsername()` and `verifyPassword()` methods are defined on the front-end. They use regular expressions to determine the user's input username and password. If the format is incorrect, a prompt will be thrown and the registration process will be blocked.

In addition, when the user enters a username, the `verifyUserId()` method will be called. It will send a request to the backend to check whether the username has been registered and prompt the user.

After the execution of these three advanced methods is completed, a registration POST request will be sent to the back-end, which contains the username and password. Of course, restricting the user's input only on the front end will have a big security risk, and it is important to restrict it on the back-end as well.

```
this.$axios({
    method: 'post',
    url: this.$global.request("register"),
    headers:{'Content-Type':'application/x-www-form-urlencoded'},
    data: this.$qs.stringify({
      username: this.username,
      password: this.password,
    })
})
```

This is the code of register in the back-end. It can be seen that the back-end is similar to the front-end which will judge whether the data sent meets the requirements and whether the username exists in the database.

Then, the system uses the RSA algorithm to generate a pair of keys for each user, and stores the public key in the data table. For security reasons, the system will not retain the private key, but sends the private key back to the user with the data returned after the successful registration. The user is responsible for keeping it properly.

In addition, any password should not be saved in plain text. The `generate_password_hash()` method will be called to encrypt the user password and store it in the database.

```
def generate_key(self):
    rsa = RSA.generate(1024)
    self.__private_key = rsa.exportKey(pkcs=8, protection="scryptAndAES128-CBC").decode('UTF-8')
    self.__public_key = rsa.publickey().exportKey().decode('UTF-8')

def get_private_key(self):
    return self.__private_key

def get_public_key(self):
    return self.__public_key
```

```
@app.route("/register", methods=['POST'])
def register():
    if "username" in request.form and "password" in request.form:
        username = request.form["username"]
        password = request.form["password"]

        user_in_db = User.query.filter(User.username == username).first()
        if user_in_db:
            return jsonify({
                'code': 400,
                'msg': 'Username already exists'
```

```
            })

        if len(password) < 6 or len(password) > 18:
            return jsonify({
                'code': 400,
                'msg': 'Password length wrong'
            })

        if re.match('^[a-zA-Z]{1}[a-zA-Z0-9\_]{3,15}$', username) is None:
            return jsonify({
                'code': 400,
                'msg': 'Username format wrong'
            })

        en = Encrypt(username)
        en.generate_key()
        public_key = en.get_public_key()
        private_key = en.get_private_key()

        passwd_hash = generate_password_hash(password)

        user = User(username=username, password_hash=passwd_hash, public_key=public_key)
        db.session.add(user)
        db.session.commit()

        return jsonify({
            'code': 200,
            'msg': 'register success',
            'privateKey': private_key
        })

    else:
        return jsonify({
            'code': 400,
            'msg': 'register failure'
        })
```

2.1.2 Login

After the registration is successful, the front-end will call the routing method of Vue and automatically jump to the Login page. The process here is similar to registration, `verifyUsername()` and `verifyPassword()` are called to determine the input, and then send the data to the back-end processing.

After the back-end receives the data, it assigns the varable `online` the value `True` in the database, which marks the user as online. In addition, `check_password_hash()` will also be called to verify whether the user password is correct, and a unique token will be generated and returned to the user. The role of this token will be described in the next subsection.

After the front-end receives the data returned by the back-end, it stores the token locally in the browser, and then uses the routing method to jump to the ChatList page.

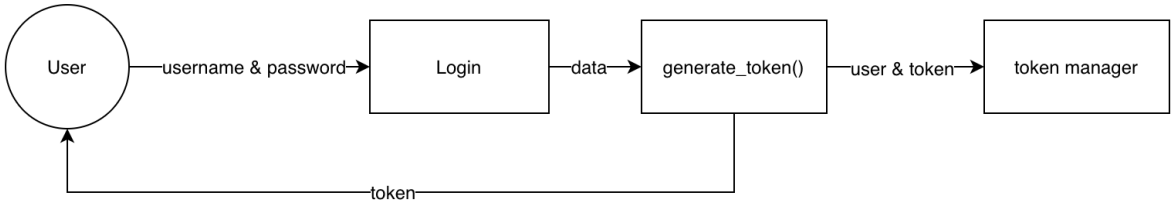
```
@app.route("/login", methods=['POST'])
def login():
    if "username" in request.form.keys() and "password" in request.form.keys():
        username = request.form["username"]
        password = request.form["password"]

        user_in_db = User.query.filter(User.username == username).first()
        if not user_in_db:
            return jsonify({
                "code": 400,
                "msg": "Invalid username or password"
            })
        if check_password_hash(user_in_db.password_hash, password):
            token = generate_token(user_in_db.username)

            # add online tag
            user_in_db.online = True
            db.session.commit()
```

```
            return jsonify({
                "code": 200,
                "msg": "Login success",
                "token": token
            })
        else:
            return jsonify({
                "code": 400,
                "msg": "Invalid username or password"
            })
    else:
        return jsonify({
                "code": 400,
                "msg": "Invalid data"
        })
```

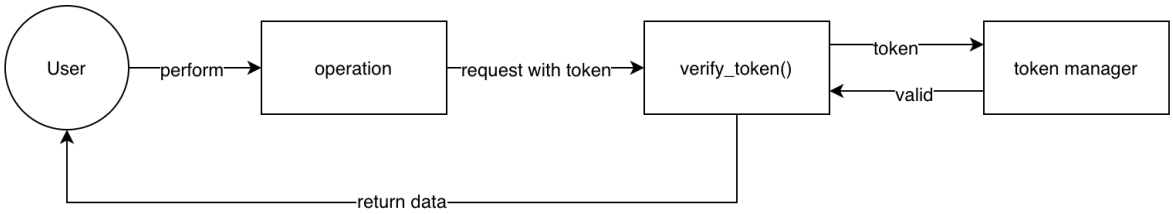
2.1.3 Token



The token is a string of unique keys generated by the server. When the user logs in for the first time, the `generate_token()` method will generate a Token and return it to the client. In the future, the client only needs to bring this Token to request data without having to bring the username and password again.

```
def generate_token(user):
    expiration = 3600
    serializer = Serializer(Config.SECRET_KEY, expires_in=expiration)
    token = serializer.dumps({"username": user}).decode("ascii")
    return token
```

The token is a good way to verify the user's identity and permissions. An expiration time can be set for each token. In this project, we set an expiration time of 60 minutes. In each following operation, the request header will always carry this token.



For each method that needs to be verified in the back-end, we use the decorator `@auth.login_required`, which means that the following method `verify_token(token)` will be executed before executing the method to verify whether the token is valid.

```
@auth.verify_token
def verify_token(token):
    g.user = None
    s = Serializer(Config.SECRET_KEY)
    try:
        data = s.loads(token)
```

```
    except SignatureExpired:
        return False
    except BadSignature:
        return False
    username = data["username"]
    if not username:
        return False
    else:
        g.user = User.query.filter(User.username == username).first()
        return True
```

2.1.3 Logout

Although the token timeout will cause the user to log out automatically, WebChat still provides a way to log out manually. This method clears the token stored locally in the browser on the front-end, redirects the page back to the login page, and then sends a request to rear-end.

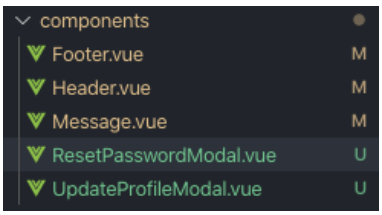
```
logout(){
      // request logout
      this.$axios({
        method: 'post',
        url: this.$global.request("logout"),
        headers:{
          'Content-Type':'application/x-www-form-urlencoded',
          "Authorization": "bearer " + this.$token.getToken()
        }
      })
      // remove local token
      this.$token.removeToken();
      // redirect
      if(this.fromPath==undefined) this.fromPath="/";
      this.$router.push(this.fromPath);
 },
```

In the back-end, this method invalidates the user's token and marks the user as offline.

```
@app.route("/logout", methods=['POST'])
@auth.login_required
def logout():
    user = g.user.username
    user_in_db = User.query.filter(User.username == user).first()
    # add offline tag
    user_in_db.online = False
    db.session.commit()
    g.user = None
    return jsonify({
            "code": 200,
            "msg": "Logout success"
        })
```

2.1.4 Modify Username and Password

For convenience, WebChat allows users to modify usernames and passwords. These two functions are very simple. The front end sends the data that needs to be modified to the back-end using PUT request, and the back-end updates the database after verifying the data. In the front-end, we also encapsulated these two components to facilitate the call: `ResetPasswordModal` and `UpdateProfileModal`.

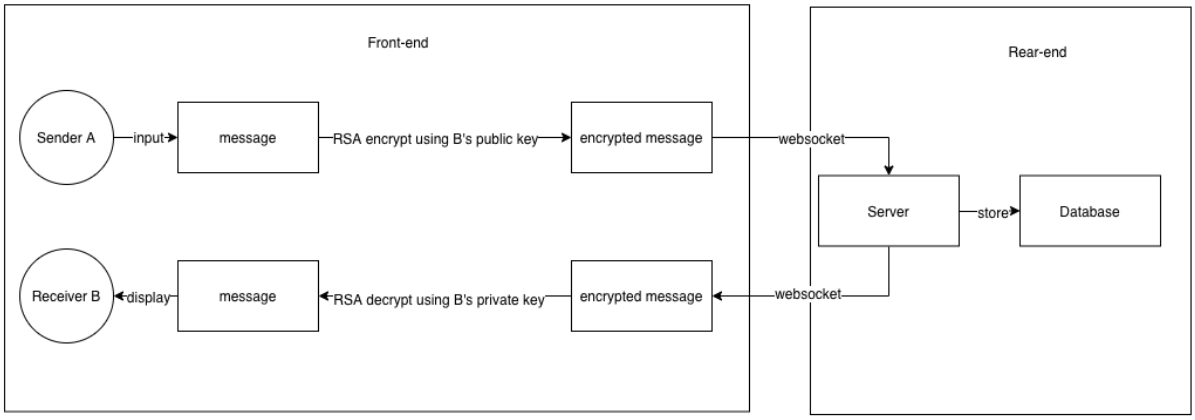


```python
@app.route("/changePassword", methods=['PUT'])
@auth.login_required
def change_password():
    user_in_db = ""
    if g.user is not None:
        user = g.user.username
        user_in_db = User.query.filter(User.username == user).first()
    else:
        return jsonify({"code": 400, "msg": "User type invalid"})
    if "prev_password" in request.form and "new_password" in request.form:
        password = request.form["prev_password"]
        new_password = request.form["new_password"]
        if len(new_password) > 5 and len(new_password) < 19:
            if check_password_hash(user_in_db.password_hash, password):
                user_in_db.password_hash = generate_password_hash(new_password)
                db.session.commit()
                return jsonify({"code": 200, "msg": "password changed"})
            else:
                return jsonify({"code": 400, "msg": "wrong password"})
        else:
            return jsonify({"code": 400, "msg": "Password length not match"})
    else:
        return jsonify({"code": 400, "msg": "Format error"})
```

2.2 Secure Communication

2.2.1 Private Chat

For private chat, WebChat adopts RSA algorithm. Except for the private key issued when the user registers, the entire information exchange process only exposes the user's public key. The sender uses the receiver's public key to encrypt, and the receiver uses her/his own private key to decrypt. The process of private key decryption is only performed locally. Messages in the chat process are all ciphertext, so as long as the user protects the private key, The entire communication process is safe.



When opening ChatList, the front-end will send a GET request to the server to obtain all user information including id, username, online status and public key, and store them in the list variable `userList`.

```javascript
getUserList(){
    let _this = this;

    this.$axios({
        method: 'get',
        url: this.$global.request("userList"),
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
            "Authorization": "bearer " + this.$token.getToken()
```

```
        }
    })
    .then(function (response) {
        if (response.data.code == 200) {
            _this.userList=response.data.data;
        }
        if (response.data.code == 400) {
            $('.toast').toast('show');
            _this.messageFailure=true;
            _this.hintTitle="Unknown error";
            _this.hintText=response.data.msg+" unknown error, please check console log";
        }
    })
}
```

Here, the `v-for` function provided by Vue is called to directly display the user list.

```
<li v-for="(u, index) in userList" :key="index" @click="routeToView(u)">
    <div class="d-flex flex-row">
        <div class="avatar" :style="{ background: 'url(' + identicon(u.username) + ')'}"></div>
        <div class="d-flex flex-column align-items-start justify-content-between ml-2">
            <span class="text-dark font-weight-bold username">{{u.username}}</span>
            <small class="text-secondary yourself" v-show="u.id==id">yourself :)</small>
        </div>
    </div>
    <small v-show="u.online" class="text-success">online</small>
    <small v-show="!u.online" class="text-secondary">offline</small>
</li>
```

On each user-item, a router method is bound so that clicking it will jump to the ChatView page, but it is worth noting that, in order to save resources and to distinguish users, except when performing routing jumps, We also need to define a query to transmit the selected user information along with the route.

```
routeToView(u){
    this.$router.push({
        name: 'ChatView',
        query:{ user: u, key: { public: this.publicKey }, id: this.id}
    });
},
```

In the Chatview, when the user enters the message and clicks send, the system will use the recipient's public key to encrypt the message and use WebSocket to send the message, time, and sender and recipient id to the server.

```
this.$socket.emit('webchat_message', {
      "sender": this.id,
      "receiver": this.user.id,
      "message": this.encrypt(this.messageText),
      "dateTime": new Date().Format("yyyy-MM-dd HH:mm")
  });
```

The server will monitor WebSocket and store the received encrypted information into the database, and broadcast it to all users.

```
@socketio.on('webchat_message')
def chat_message(form):
    sender_id = form["sender"]
    receiver_id = form["receiver"]
    content = form["message"]
    post_time = datetime.datetime.now()
    message = Message(sender_id=sender_id, receiver_id=receiver_id, content=content, post_time=post_time)
    db.session.add(message)
```

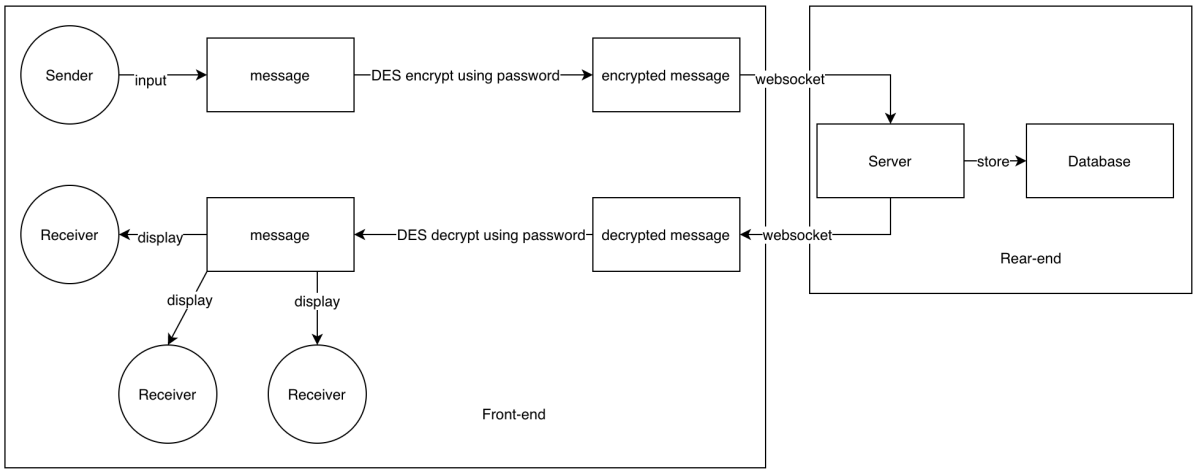
```
    db.session.commit()
    send(form, broadcast=True)
```

The user's front-end will also monitor the WebSocket from the server. Whenever a message is received, if the recipient is him/her, it will decrypt it with the private key stored locally. If the decryption is successful, a message will be displayed.

```
message: function(data){
    if(data.receiver==this.id && data.sender==this.user.id){
        let con = document.querySelector(".message-container");
        let msb = document.createElement("div");
        msb.innerHTML = `<div class="small text-secondary">${data.dateTime}</div>
            <div class="button-green">${this.decrypt(data.message)}</div>`;
        con.appendChild(msb);
    }
},
```

2.2.2 Group Chat

Since group chat involves multiple senders and receivers, RSA algorithm cannot be used, so here WebChat uses DES algorithm, that is, all members in each group chat with the same password to encrypt and decrypt messages. Similarly, the process will only exist locally, and any message exposed on the network is encrypted.



When a user enters a group, the system will prompt the user to enter the password of the group. This password will be stored locally for encryption and decryption. When the user enters the information and clicks to send, the front end will encrypt the secretary and send it to the server through the `emit()` method of WebSocket.

```
this.$socket.emit('webchat_group_message', {
    "sender": this.id,
    "name": this.user,
    "message": this.encrypt(),
    "dateTime": new Date().Format("yyyy-MM-dd HH:mm")
});
```

After the server receives the message, it will broadcast the data to the users in the group.

```
@socketio.on('webchat_group_message')
def chat_group(form):
```

```
    print(form)
    emit("webchat_group_message", form, broadcast=True)
```

The user's front-end will monitor WebSocket, and after receiving the message, it will call the DES decryption algorithm to decrypt the data and display it.

At the same time, it is important to pay attention to judge the sender's id and the user's own id, if the same proves that the recipient is the sender, then the user does not need the recipient of the message.

```
webchat_group_message: function(data){
  if(data.sender!=this.id){
      let con = document.querySelector(".message-container");
      let msb = document.createElement("div");
      msb.innerHTML = `<div class="small text-secondary">${data.dateTime}</div>
          <div class="small text-secondary mt-4 text-left">${data.name}</div>
          <div class="button-green">${this.decrypt(data.message)}</div>`;
      con.appendChild(msb);
  }
},
```

2.2.3 File Transfer

This WebChat also provides the function of sending files. Of course, in order to ensure security, files need to be encrypted during transmission as well. I won't repeat the process of sending and receiving files here because they are similar to communication. This subsection will focus on how files are read and encrypted.

In the front-end ChatView, we define two methods `sendFile()` and `receiveFile()` as sending files and receiving files respectively. When receiving the file uploaded by the user, we can call the `readAsDataURL()` method of FileReader to convert the file into base64 format, which allows us to use the RSA algorithm for encryption and decryption. The process after confidentiality is the same as that of sending documents.

```
let fileInput = document.querySelector("#fileInput");
let file = fileInput.files[0];
let reader = new FileReader();
reader.readAsDataURL(file);

reader.onload = function(e){
    let base64 = e.currentTarget.result;
    _this.$socket.emit('webchat_file', {
        "sender": _this.id,
        "receiver": _this.user.id,
        "file": _this.encrypt(base64),
        "fileName": file.name,
        "dateTime": new Date().Format("yyyy-MM-dd HH:mm")
    });
}
```

After receiving the file and decrypting it, the `receiveFile()` method will propose the file name and base64 value, and use regular expressions to segment the string and convert the file part into File form, and then call the `saveAs()` method of `FileSaver` to Download the file locally. The encryption and decryption process also only exists locally, and the group password will not be exposed during transmission.

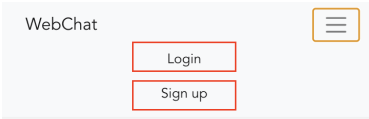
```
let arr = fileBase64.split(',')
let type = arr[0].match(/:(.*?);/)[1]
let fileExt = type.split('/')[1]
let bstr = atob(arr[1])
let n = bstr.length
let u8arr = new Uint8Array(n)
while (n--) {
```

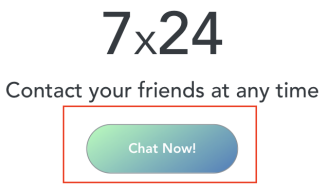
```
    u8arr[n] = bstr.charCodeAt(n);
}
let finalFile = new File([u8arr], filename+'.', {
    type: type
});
FileSaver.saveAs(finalFile);
```

3 User Interface

3.1 Register, Login and Logout

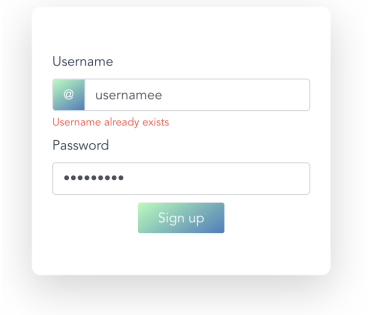
Click the button in the upper right corner of the homepage to jump to the login and registration pages respectively. If you are not currently logged in, clicking the "Chat Now!" button in the middle of the page will also automatically jump to the login page.





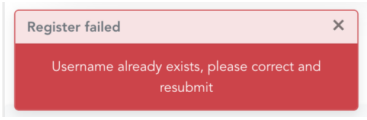
Index

In the registration page, the user is asked to enter a username and password, both of which need to meet a certain format. If the input is incorrect, the system will throw a prompt.



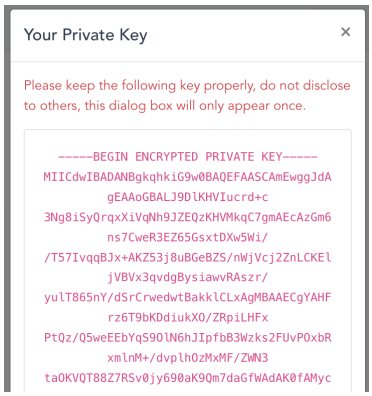
Sign up page

The error message shown in the figure below is a system encapsulated prompt component. Errors will be marked in red, and correct ones will be marked in green. This component will be called in any part of the system.



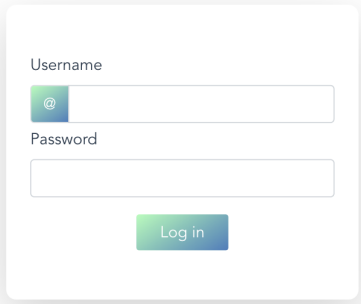
Error message

After the user is successfully registered, the user's private key will be displayed to the user through a prompt box. This prompt box will only appear once, so the user needs to keep his/her private key properly.



Private key

On the login page, the user needs to enter the user name and password to log in to the system.

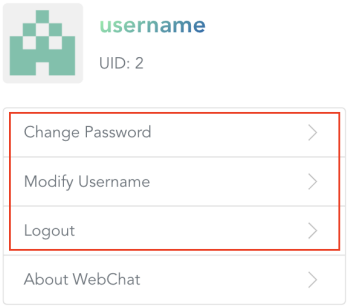


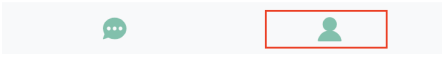
Login page

After successful login, the page will automatically jump to the chat list.

3.2 Modify Personal Information

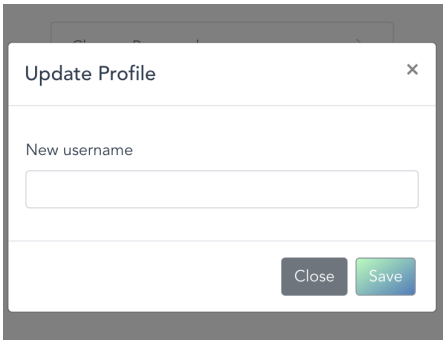
The second button at the bottom of the chat list will enter the personal information page, where users can view their personal information, modify their user name, modify password, and log out.

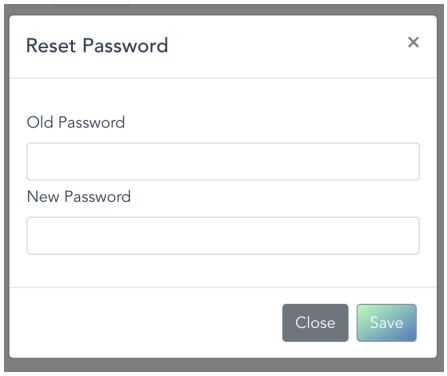




Profile

After clicking to modify the password or user name, an input box will pop up, enter the content to be modified, and click save. It should be noted that if the modification is successful, the system will automatically log out and redirect to the login page.

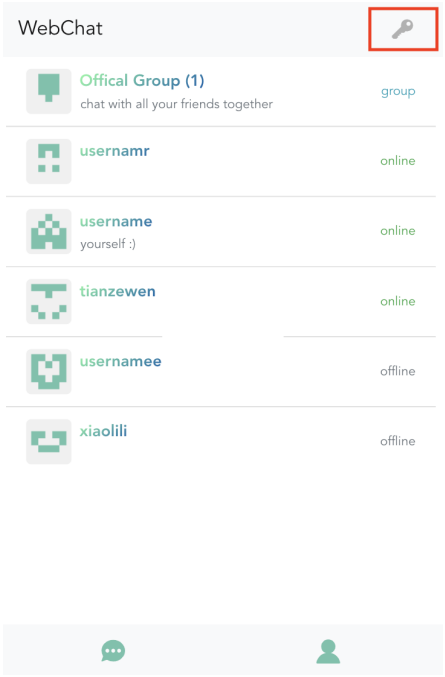




Reset password

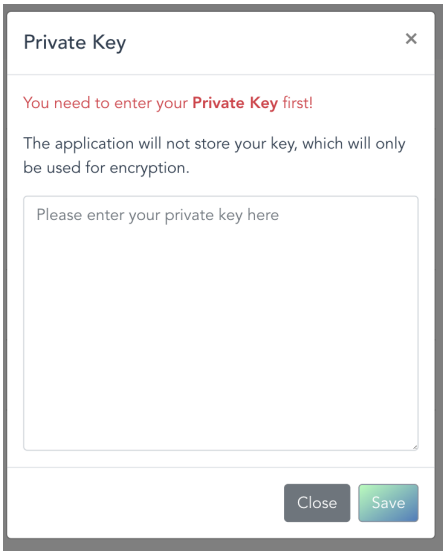
3.3 Private Chat

On the chat list page, all users and groups in the system are listed here. Before starting a chat, users need to click the key icon in the upper right corner of the page.

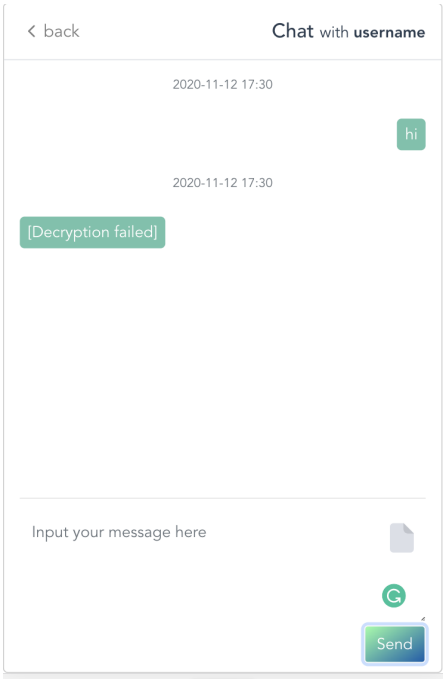


ChatList page

Enter the private key in the pop-up box. This private key will only be stored locally for encryption and decryption, and will be automatically deleted after the user logs out.

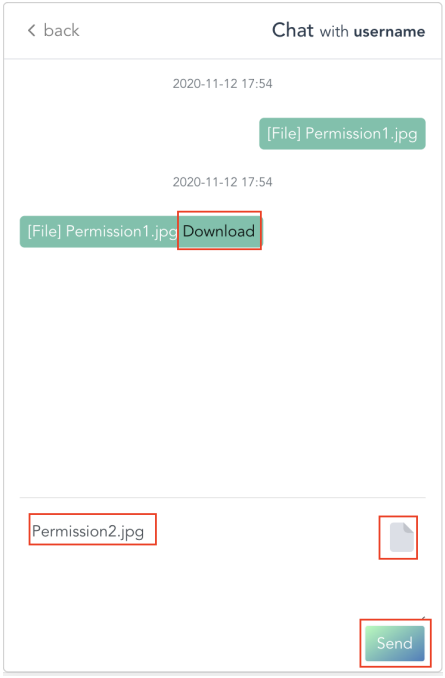


After that, click on any user in the chat list to chat with her/him. It should be noted that if you did not enter the key, or the key is incorrect, the following prompt will appear for decryption failure.



ChatView page

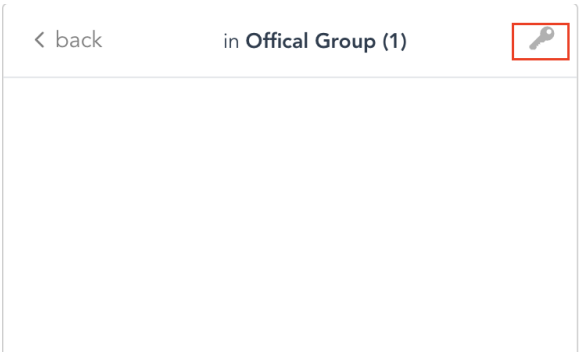
The user can also send a file to friends by clicking the file icon and upload a local file. The receiver can click the Download address to save the file received. File messages will be marked with [File] prefix.

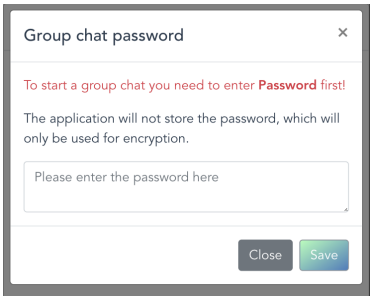


File transfer

3.4 Group Chat

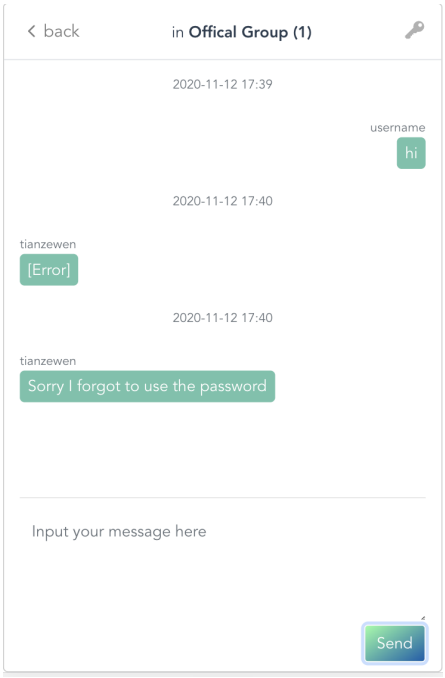
Click on any group to enter the group chat. All users in each group will be encrypted with the same password by default, so please make sure to agree a password with your friend before starting the chat, and click on the upper right corner Key icon, enter this password.





Group chat password

Then you can chat happily with your friends. If a user forgets to enter the password or enters an incorrect password, the error message shown below will appear.



GroupView page