Secure Instant Message Tool Design

Ruiyang Xu and B.H.Priyanka

1. SETUP

1.1. Assumptions

We assume that all the usernames and PDM information has already been stored in the server's database. So that no user needs to do account registration to the server (though they still have to do the login registration). Also the client will know the public key of the server.

1.2. Architecture

The basic architecture of our design has one server with multiple clients (Figure 1). The server is used for login, which will do authentication of the user and then register it's information which can later be used by other users. The server will maintain two lists: one is permanent in some database and the other is ephemeral in local memory. As mentioned in the requirement, nothing will be stored in a client machine. All the key pairs are generated dynamically during run-time and all the info are fetched from server when necessary. The client basically knows only about the server's IP and port number. Every client has at least two ports (Figure 2) for communication: one for server and one for the other clients.

2. PROTOCOLS

2.1. Authentication Protocol for Login

A PDM protocol is used during login authentication. As indicated in Figure 3, after being successfully authenticated with each other, the DH key will be used in all client-server communication. The client will also generate a RSA key pair for peer authentication purpose later. The public key will then be stored in the server. *Currently those global tables might not be thread-safe, we didn't have enough time to add locks for them. So this might cause some race condition issue on the server.*

2.2. Key Establishment Protocol And Peer Authentication

After the user types in the 'send' command, the source client will first ask the server for target user's information (i.e. IP, port number and stored public key, it might need to open a new port for later communication) and initiate a mutual public key authentication with the target using its public key. The target client receiving this authentication request will also fetch the server to find out the source's information (which include its public key). And then uses that to finish the mutual public key authentication. After the peer authentication, source and target will generate new RSA key pairs and exchange new public keys.

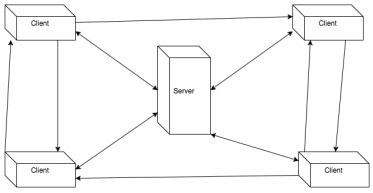


Fig. 1. The basic architecture.

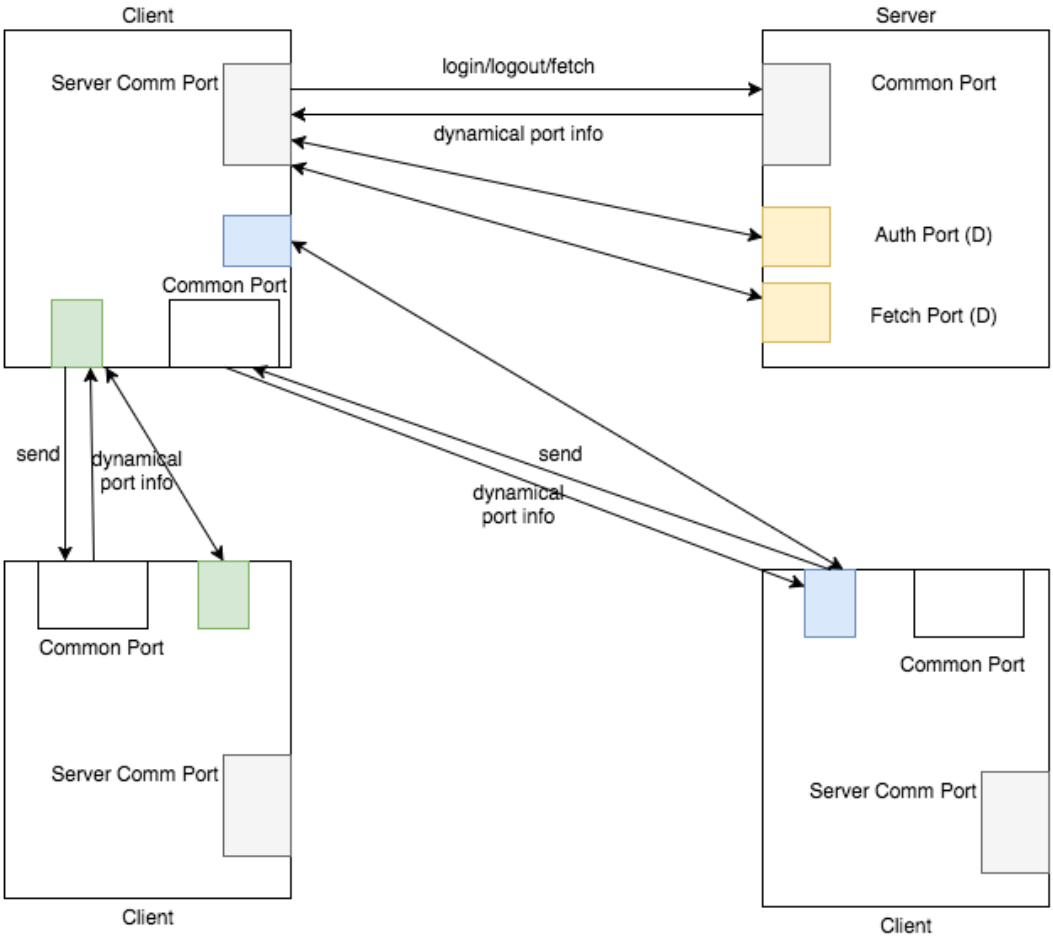


Fig. 2. Communication port design.

Table I. Permanent user info list

User	Moduli
Alice	$2^{W_1} mod\ p$
Bob	$2^{W_2} mod\ p$

Table II. Ephemeral user info list

User	IP	Port	RSA-Auth-Public
Alice	192.168.1.100	5505	*****
Bob	192.168.1.105	4567	*****

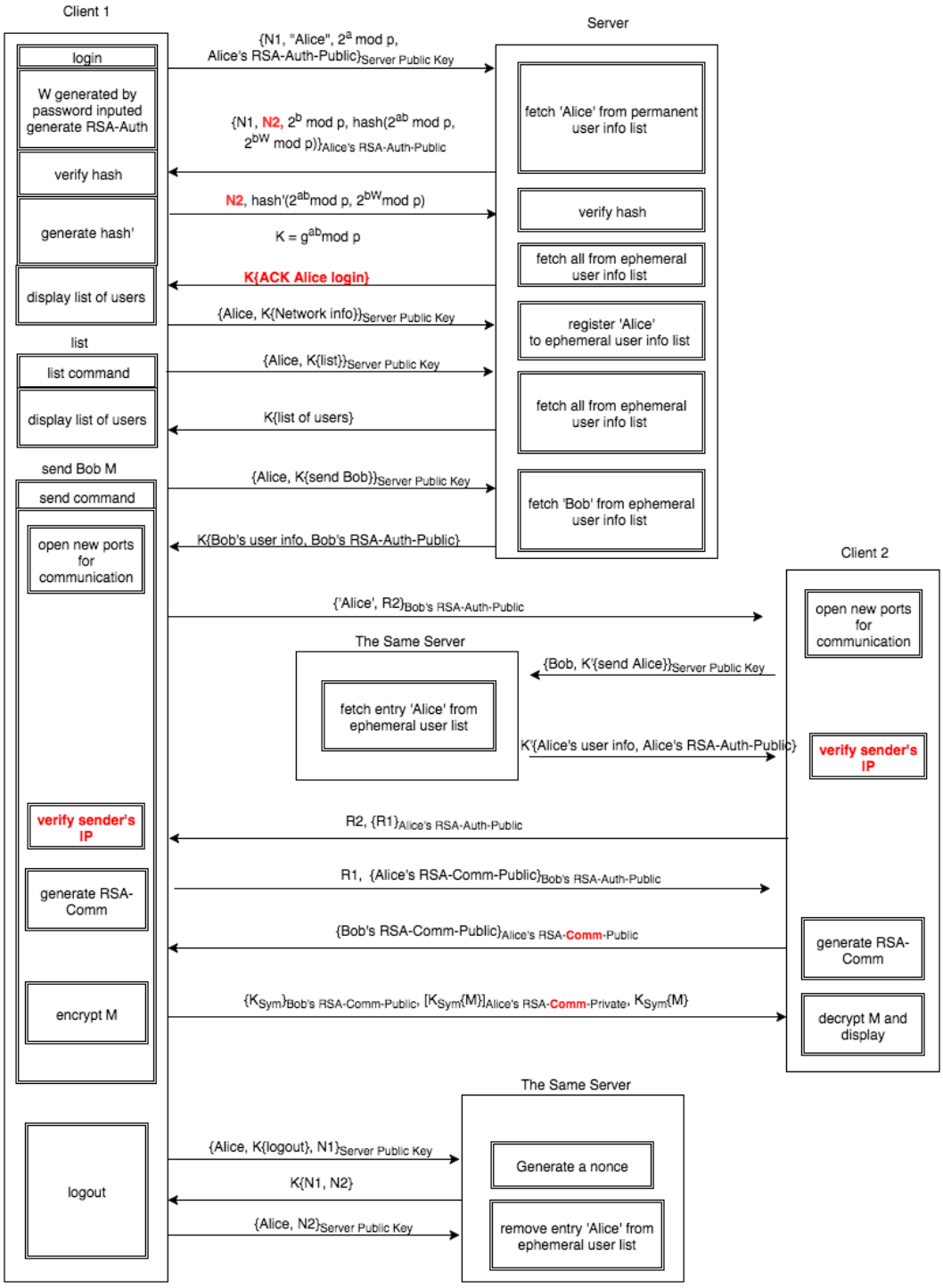


Fig. 3. The overall protocols. Modified parts are marked in red

2.3. Messaging Protocol

Source will generate a symmetric key to encrypt message, and encrypt the symmetric key with RSA public key. Also included a signature on the encrypted message for integrity verification.

2.4. Logout Protocol

After the user logs out (meaning to terminate the client program), a 'logout' command will be sent to the server. The server then simply removes the user's entry from ephemeral user info list. To prevent a replay attack, nonces are used. *Since UDP is being used, server will have no idea on logout if the client exits abnormally. If given more time, we would be able to implement a 'heartbeat' message mechanism*

3. SECURITY FEATURES AND FLAWS

3.1. Use of Weak Passwords

Since this protocol is a strong password protocol, use of weak password will be susceptible only if the attacker has broken into the server database and has successfully done the dictionary attack on the stored moduli.

3.2. Denial of Service Attack

Only if the attacker can somehow know a user name, then he can initiate a large amount of login requests. The server will then have to compute multiple times of $hash(2^{ab} mod\ p, 2^{bW} mod\ p)$. To foil this attempt, the server can use some delays for the same user's login request.

3.3. End-point Hiding And Perfect Forward Secrecy

This protocol does have end-point hiding. Every message is totally encrypted with a proper secret key which does not reveal the user identity. And it also provides Perfect Forward Secrecy, since RSA key pair is generated dynamically for every message communication.

3.4. Untrustable/Compromised Server

If a server is untrustable, then we assume it has no idea of users' PDM moduli information. Then the mutual authentication will be foiled. A compromised server will only leak PDM moduli, and the attacker can hardly derive the original password from that. *However, since we are not using a real random process to generate prime number from the password, this might be a flaw in current implementation and also leaves a chance for the attacker to crack the password. If we have enough time, a real PDM protocol can be implemented.*

3.5. Replay Attack, Reflection Attack and Impersonation

The design avoids replay attack since it uses randomly generated session tokens-NONCE on both the client and the server sides to establish authentication. It also avoids the reflection attack since every message is encrypted with either rsa public key or the shared DH key. It's also impossible to impersonate any other user.

3.6. Man-in-the-Middle Attack

Design is not vulnerable to a passive Man-in-the-Middle attack since there is no information that the attacker can obtain by eavesdropping or by any other means. Each message is secure. However, since we are using UDP, an active MITM attack might be harmful: since dynamic port info is being sent out with plain text, the attacker can get this info and send some garbage to the peer so that it foils the communication.*If*

given more time, we would rather not use UDP and also ensure that the port info was encrypted.

4. VULNERABILITIES FOUND IN OTHER TEAMS

4.1. Abhishek Sawarkar - Pratik Pande

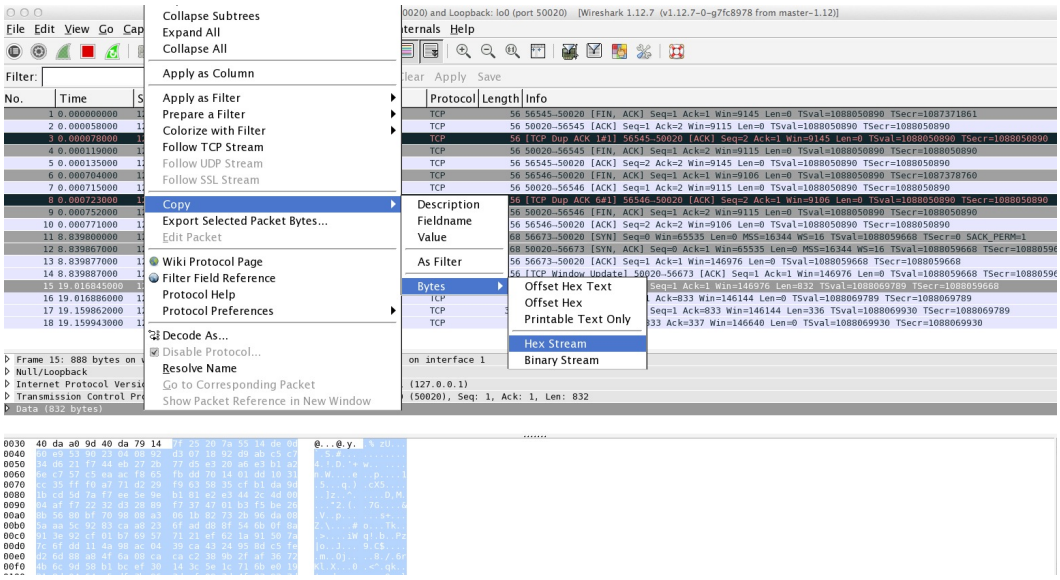
One of the requirement of the project was: Your protocols should satisfy the following constraints: The users only needs to remember a single password. The client application should not remember the users passwords.If you are using public/private keys, the client application cannot remember the private/public key of the users.You can however assume that the client application knows the public key of the server (if you are using public/private keys).*The design of this team has a serious flaw since each client stores the public keys of other users which is forbidden.*

4.2. Onyeka Chu Igabari - Vishal Rao

Though these are not serious issues, we would like to mention that the design of this team has not handled few exceptions like - sending message to a non-existing user which causes the program to exit. And also few print statements on server side are irrelevant.

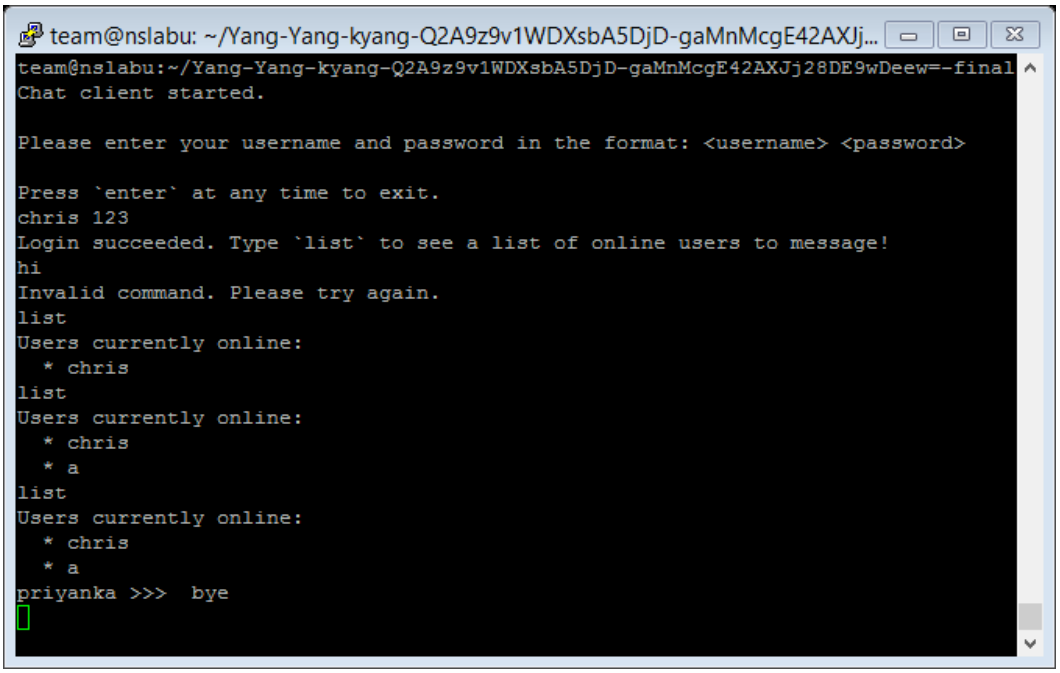
4.3. Christophe Leung - Yang Yang

a. This team has not implemented logout/exit functionality.
b. In the login module, the following first two messages as in the design document is not implemented in the code.
Client Authenticates Server
A to C: M_A , N_A , K_A , T_A K_C
C to A: N_A , T_C K_A
Only third and fourth messages are implemented for client-server authentication.
Exploits done:
1.
a. A wireshark capture was done for user authentication to the server for one client.
b. Hex data of length 832 bytes was obtained and converted it to bytearray.



c. Replaced this bytearray value in the client code line 190 as shown in the image.This is to poison the server.

d. The peer-peer communication works fine until modification.
e. After the modification, suppose a new user logs into the system, the client program crashes and the send/list commands issued on the first client makes it crash too.
2.
a. Initially, we opened the session for the user 'chris' and the server.
b. In the following lines of code in the client program,
privateMessage=PrivateMessage(self.clientPort,*self.username*,userInfo['port'],toUserMessage)
privateMessage=PrivateMessage(self.clientPort,*self.username*,self.currentConnections[jsonMessage['usernam
establishPrivateMessage=EstablishPrivateMessage(self.clientPort,*self.username*,self.clientPublicKey)
we replaced self.username with any random username.
c. The username specified in the above lines are neither verified at the client nor the server. So we opened a new session for the valid user 'a'.
d. When send is issued from user 'a' to user 'chris', we see that the username displayed is what we had given in the code.



4.4. Thangella - Indira Priyadarshini

Apart from the several design flaws (they don't have document, and the logic of the program is very confusing, program crashes also sometimes). In this team code, we were able to do a replay attack on the username. Since they were transmitting hash of username and password in plain text on the network, we were able to eavesdrop those info and extract out the hash. Then we implemented a replay attack with those hashes.

4.5. Sneha Lakshmish - Pranay Surana

In the login module, passwordA is itself being sent in the first message.The server will be storing the password directly and is not safe.If the server is compromised, attacker can get the server private key to decrypt the message.

4.6. Sample4

In the user-user authentication and the list implementation,the team has used private key of the client to sign each message from client and private key of the server to sign each message from the server. Because of this, nobody can decrypt any message. Also, anybody having the public key can verify the message by decrypting it which is absolutely not safe.