CS4203 P2

200007626

17th November 2023

Contents

1 Introduction

A discussion forum is somewhere you go to share ideas and engage in productive conversation - however, these forums are often not designed with security in mind. Information pertaining to the forums and their users may be stored unencrypted on a central database somewhere; often, these forums are open to public view. We were tasked with designing and implementing a secure discussion forum, with confidentiality and privacy of paramount importance.

2 Defining the Problem

A secure discussion forum ought to operate as a zero-trust system to maximise security - no unauthorised body, either user or developer, should have access to information that is not theirs. Every interaction between machines needs to be encrypted and verified. Discussion groups should be access-controlled, invisible and inaccessible to unauthorised users, and invite-based. As per most discussion forums, messages should persist for some length of time - this necessitates some sort of asynchronous encryption standard for users not present for a given session. Message persistence also necessitates some sort of temporary storage system, which should hold messages until a user is able to receive them, at which point the message should be removed from memory. Users should be identifiable to some degree in order to allow meaningful discussion - anonymous conversations with a group of people would allow for users to impersonate each other.

3 Initial Architecture

I decided to use a traditional centralised server-based architecture, as I was most familiar with its intended functionality. Clients would send information to the server for processing, and request information from the server where needed. The server would interface with a database, which in turn stores information and messages prior to sending. I had originally intended to implement the secure discussion forum over HTTPS, with clients interfacing with the system using a browser - however, this raised the issue of secure client-side storage (specifically private-key storage). I am aware of frameworks such as Electron [1] which can be used to facilitate client-side storage, but I didn't feel comfortable enough dealing with potential scripting attacks which could be used to leak private information [2]. With security being of paramount importance, I instead implemented my discussion forum over TCP web-sockets, and provide a command-line interface - this would allow the client to interface with the system, while also allowing

the client application to utilise physical disk storage, eliminating the risk of JavaScript-based scripting attacks.

4 Threat Model

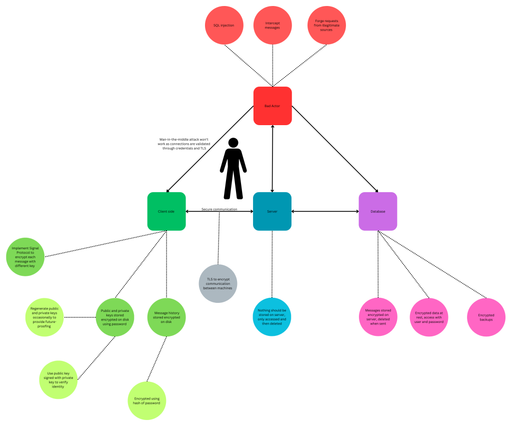

SQL injection
Intercept messages
Forge requests from illegitimate sources
Bad Actor
Man-in-the-middle attack won't work as connections are validated through credentials and TLS
Client side
Secure communication
Server
Database
Implement Signal Protocol to encrypt each message with different key
TLS to encrypt communication between machines
Regenerate public and private keys occasionally to provide future-proofing
Public and private keys stored encrypted on disk using password
Message history stored encrypted on disk
Nothing should be stored on server, only accessed and then deleted
Messages stored encrypted on server, deleted when sent
Encrypted data at rest, access with user and password
Encrypted backups
Use public key signed with private key to verify identity
Encrypted using hash of password

Figure 1: Rough mind-map style modelling of threat model on chosen architecture.

There are several attack surfaces and vectors to consider when designing a system which uses a traditional server-based architecture: the communication between client and server must be authenticated and encrypted; messages sent between clients should be encrypted using asymmetrical keys to maximise security, and any messages stored server-side should be encrypted and deleted at the first possible instance to minimise the available time any attacker has to access and then decrypt the messages; the database must at minimum be password-protected, and would ideally be encrypted at rest; client applications should be password protected against unauthorised usage; and client-side data should similarly be protected (ideally encrypted) at rest.
An attacker may attempt to sniff packets transmitted from a client application to the server, necessitating encryption of the channel. However, simply encrypting the channel won't be enough - in the case that the the encrypted channel is rendered insecure, or that the attacker decides to store sniffed packets and decrypt them at a later point, each message containing confidential or important information, such as the content of messages between clients, should also be individually encrypted. Connections between the server and the client should not only be encrypted, but authenticated to prevent man-in-the-middle attacks; moreover, each client should be required

to authenticate themselves in order to access their application and user data.
Data stored on the server could be subject to illegitimate alteration - in the case of public keys, signatures should be provided for public keys to ensure communication with the intended client. Additionally, in the case that an attacker replaces a users' key bundle entirely, the system should provide the client with access to their own public key for comparison outside of the system through an alternative channel.
In the case that the server source code is compromised, no confidential information should be passed to the server without some level of encryption, to prevent unauthorised access.

5 Chosen Defensive Technologies

To facilitate secure communication between client and server, I implemented TLS with server-certificate authentication - the client's application stores the server's public key locally to authenticate connection to the server, and subsequent communications are then encrypted.

To allow users to login securely, their hashed password and randomly generated salt is stored server-side - when the user attempts to log in, their credentials are sent to the server, which hashes the password with the salt, checks the user's credentials match, and acts appropriately based upon the result.

When users create an account, they generate a key bundle client-side - this key bundle will be used to initialise groups of users through a Triple Diffie-Hellman key exchange. The key bundle consists of an identity key pair, a pre-key pair, and a signature: the user's pre-key signed with their private identity key. This is carried out to facilitate the X3DH Key Agreement protocol outlined by Signal [3], shown in Fig 2.

When two users want to create a group, the first user requests the second user's public key bundle from the server. Upon receipt, the first user checks the signature is valid, and then generates an ethereal key pair used solely for the purposes of establishing communication, and after which it is discarded. The first user then performs key exchanges with a combination of their own private keys and the second user's public keys, and produces a secret key. They then send their public identity key, pre-key, and ethereal key, along with a separate public key (the private counterpart of which is used to initialise their Diffie-Hellman ratchet later on), as an invitation to join the group. When the second user receives this collection of keys, they perform a

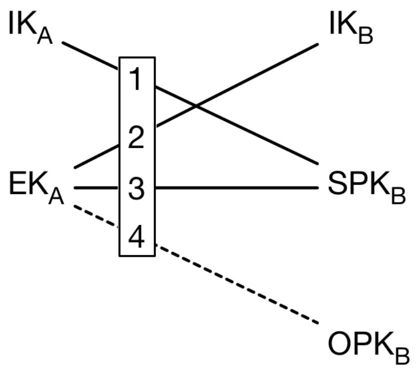


Figure 2: A high-level overview of the X3DH Key Agreement Protocol

similar combination process in order to derive the same secret key as the first user. After computing this secret key, the second user then generates a key pair used to initialise their Diffie-Hellman ratchet, and sends the public key back to the first user - at this point, the two users have successfully established a group with two people. This initialisation process is to facilitate use of the Double Ratchet protocol detailed by Signal [4].

Messages are encrypted client-client, even within groups; for every message that a user sends into a group with x users, they will encrypt it separately x-1 times, and x-1 copies of the message are sent to the server for distribution. In order to add a user to a group, all of the users within the group need to establish a connection via an invite with the new group member. This allows members of the group the opportunity to decline sending the new group member an invitation, meaning that even if all other group members invite the new member, there will be no communication between them, while maintaining the original functionality of the group. This is shown in Fig 14.

For the sake of both brevity and sanity I will avoid attempting to describe communication using Signal's Double Ratchet protocol. Instead, taken from Signal's specification, "when a message is sent or received, a symmetric-key ratchet step is applied to the sending or receiving chain to derive the message key. When a new ratchet public key is received, a DH ratchet step is performed prior to the symmetric-key ratchet to replace the chain keys" [4]. The Double Ratchet protocol ensures both forward secrecy for messages, and implicitly renegotiates keys with each message - in that case that a single message is hacked and the key is uncovered, an attacker will

not be able to use the key to decrypt any messages sent before nor after the decrypted message. The secrecy chain is self-healing, ensuring minimal disruption and maximum security and privacy for group participants. An overview of the processes involved in message sending and receiving is shown in Fig 3

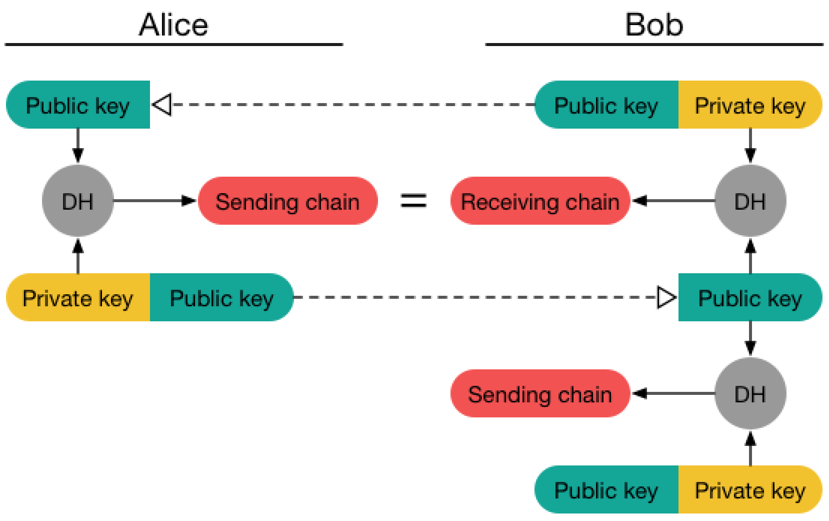


Figure 3: A high-level overview of the Double Ratchet Protocol

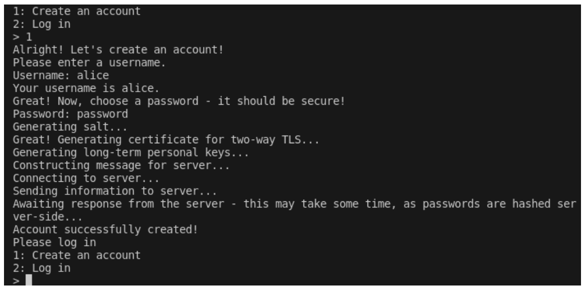
The database is password-protected - I would have liked to have encrypted the database at rest, but the school didn't seem to allow that functionality to my knowledge.

6 Critiquing the system

There are several remaining attack vectors within the system: for example, if the distribution line of client applications is hacked and the public server certificate and server IP is replaced with an alternative, users could be redirected to a different web socket and start to communicate under false pretences. However, even if this were to happen, the attacker posing as a server wouldn't be able to access any confidential information, other than the user's raw password, which isn't much use without the salt. My system is much more vulnerable to DoS attacks, as many of the methods and functionalities are not

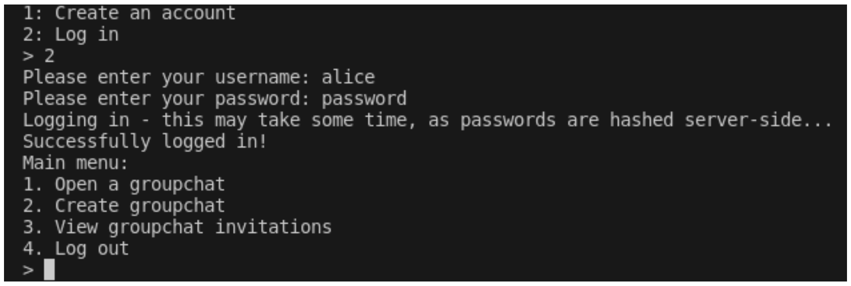
protected against deliberate misuse. It would be simple to disrupt the server by attempting to pass illegal values to the database, for example.

7 Testing and Proof of Functionality



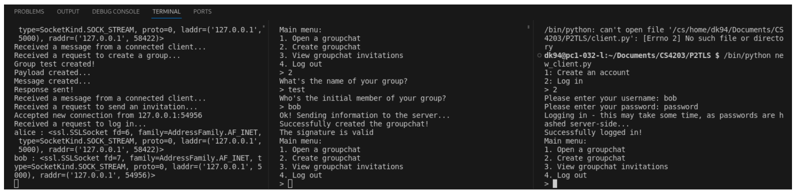
```
1: Create an account
2: Log in
> 1
Alright! Let's create an account!
Please enter a username.
Username: alice
Your username is alice.
Great! Now, choose a password - it should be secure!
Password: password
Generating salt...
Great! Generating certificate for two-way TLS...
Generating long-term personal keys...
Constructing message for server...
Connecting to server...
Sending information to server...
Awaiting response from the server - this may take some time, as passwords are hashed ser
ver-side...
Account successfully created!
Please log in
1: Create an account
2: Log in
>
```

Figure 4: Creating an account



```
1: Create an account
2: Log in
> 2
Please enter your username: alice
Please enter your password: password
Logging in - this may take some time, as passwords are hashed server-side...
Successfully logged in!
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
>
```

Figure 5: Logging in to that account

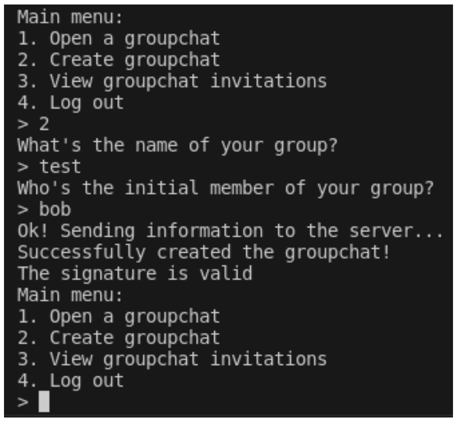


```
 type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1',
 5000), raddr=('127.0.0.1', 58422)>
Received a message from a connected client...
Received a request to create a group...
Group test created!
Payload created...
Message created...
Response sent!
Received a message from a connected client...
Received a request to send an invitation...
Accepted new connection from 127.0.0.1:54956
Received a request to log in...
alice : <ssl.SSLSocket fd=6, family=AddressFamily.AF_INET,
 type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1',
 5000), raddr=('127.0.0.1', 58422)>
bob : <ssl.SSLSocket fd=7, family=AddressFamily.AF_INET, t
ype=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5
000), raddr=('127.0.0.1', 54956)>
```

```
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 2
What's the name of your group?
> test
Who's the initial member of your group?
> bob
Ok! Sending information to the server...
Successfully created the groupchat!
The signature is valid
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
>
```

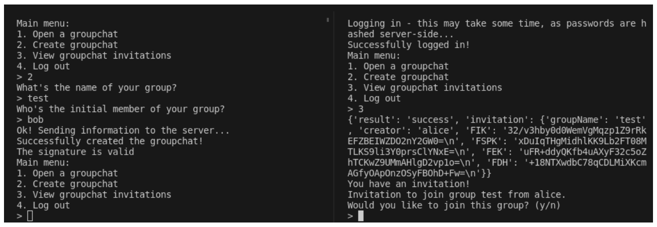
```
/bin/python: can't open file '/cs/home/dk94/Documents/CS
4203/P2TLS/client.py': [Errno 2] No such file or directo
ry
dk94@pc1-032-l:~/Documents/CS4203/P2TLS $ /bin/python ne
w_client.py
1: Create an account
2: Log in
> 2
Please enter your username: bob
Please enter your password: password
Logging in - this may take some time, as passwords are h
ashed server-side...
Successfully logged in!
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
>
```

Figure 6: Second user logs in



```
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 2
What's the name of your group?
> test
Who's the initial member of your group?
> bob
Ok! Sending information to the server...
Successfully created the groupchat!
The signature is valid
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
>
```

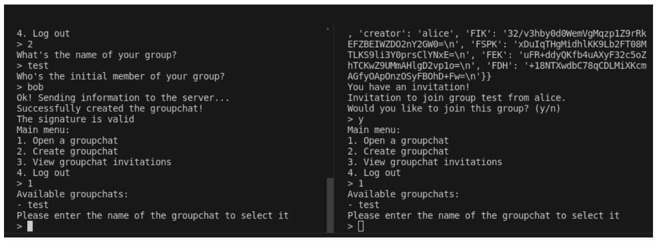
Figure 7: Alice sends invitation to Bob



```
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 2
What's the name of your group?
> test
Who's the initial member of your group?
> bob
Ok! Sending information to the server...
Successfully created the groupchat!
The signature is valid
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
>
```

```
Logging in - this may take some time, as passwords are h
ashed server-side...
Successfully logged in!
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 3
{'result': 'success', 'invitation': {'groupName': 'test'
, 'creator': 'alice', 'FIK': '32/v3hby0d0WemVgMqzp1Z9rRk
EFZBEIWZDO2nY2GW0=\n', 'FSPK': 'xDuIqTHgMidhlKK9Lb2FT08M
TLKS9li3Y0prsClYNxE=\n', 'FEK': 'uFR+ddyQKfb4uAXyF32c5oZ
hTCKwZ9UMmAHlgD2vp1o=\n', 'FDH': '+18NTXwdbC78qCDLMiXKcm
AGfyOApOnzOSyFBOhD+Fw=\n'}}
You have an invitation!
Invitation to join group test from alice.
Would you like to join this group? (y/n)
>
```

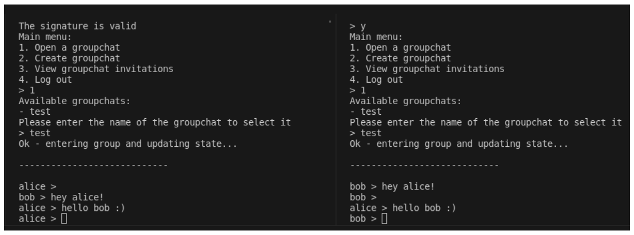
Figure 8: Bob receives invitation



```
4. Log out
> 2
What's the name of your group?
> test
Who's the initial member of your group?
> bob
Ok! Sending information to the server...
Successfully created the groupchat!
The signature is valid
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 1
Available groupchats:
- test
Please enter the name of the groupchat to select it
>
```

```
, 'creator': 'alice', 'FIK': '32/v3hby0d0WemVgMqzp1Z9rRk
EFZBEIWZDO2nY2GW0=\n', 'FSPK': 'xDuIqTHgMidhlKK9Lb2FT08M
TLKS9li3Y0prsClYNxE=\n', 'FEK': 'uFR+ddyQKfb4uAXyF32c5oZ
hTCKwZ9UMmAHlgD2vp1o=\n', 'FDH': '+18NTXwdbC78qCDLMiXKcm
AGfyOApOnzOSyFBOhD+Fw=\n'}}
You have an invitation!
Invitation to join group test from alice.
Would you like to join this group? (y/n)
> y
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 1
Available groupchats:
- test
Please enter the name of the groupchat to select it
>
```

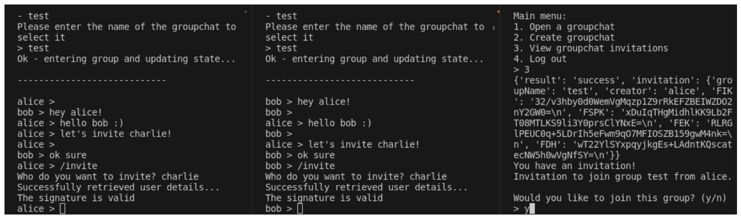
Figure 9: Both users now have access to the same group



```
The signature is valid
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 1
Available groupchats:
- test
Please enter the name of the groupchat to select it
> test
Ok - entering group and updating state...

---------------------------

alice >
bob > hey alice!
alice > hello bob :)
alice >
```

```
> y
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 1
Available groupchats:
- test
Please enter the name of the groupchat to select it
> test
Ok - entering group and updating state...

---------------------------

bob > hey alice!
bob >
alice > hello bob :)
bob >
```

Figure 10: The two users communicating

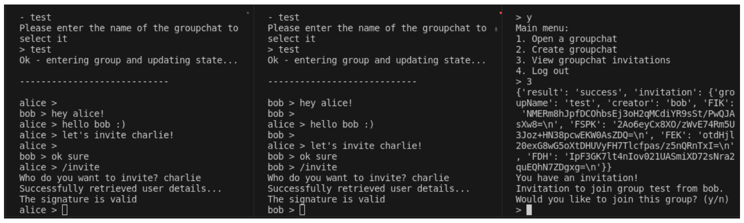


```
- test
Please enter the name of the groupchat to
select it
> test
Ok - entering group and updating state...

----------------------------


alice >
bob > hey alice!
alice > hello bob :)
alice > let's invite charlie!
alice >
bob > ok sure
alice > /invite
Who do you want to invite? charlie
Successfully retrieved user details...
The signature is valid
alice >
```

```
- test
Please enter the name of the groupchat to
select it
> test
Ok - entering group and updating state...

----------------------------


bob > hey alice!
bob >
alice > hello bob :)
bob >
alice > let's invite charlie!
bob > ok sure
bob > /invite
Who do you want to invite? charlie
Successfully retrieved user details...
The signature is valid
bob >
```

```
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 3
{'result': 'success', 'invitation': {'gro
upName': 'test', 'creator': 'alice', 'FIK
': '32/v3hby0d0WemVgMqzp1Z9rRkEFZBEIWZDO2
nY2GW0=\n', 'FSPK': 'xDuIqTHgMidhlKK9Lb2F
T08MTLKS9li3Y0prsClYNxE=\n', 'FEK': 'RLRG
lPEUC0q+5LDrIh5eFwm9qO7MFIOSZB159gwM4nk=\
n', 'FDH': 'wT22YlSYxpqyjkgEs+LAdntKQscat
ecNW5h0wVgNfSY=\n'}}
You have an invitation!
Invitation to join group test from alice.

Would you like to join this group? (y/n)
> y
```

Figure 11: Charlie gets Alice's invite

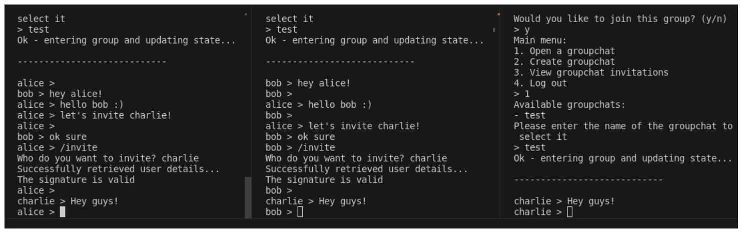


```
- test
Please enter the name of the groupchat to
select it
> test
Ok - entering group and updating state...

----------------------------


alice >
bob > hey alice!
alice > hello bob :)
alice > let's invite charlie!
alice >
bob > ok sure
alice > /invite
Who do you want to invite? charlie
Successfully retrieved user details...
The signature is valid
alice >
```

```
- test
Please enter the name of the groupchat to
select it
> test
Ok - entering group and updating state...

----------------------------


bob > hey alice!
bob >
alice > hello bob :)
bob >
alice > let's invite charlie!
bob > ok sure
bob > /invite
Who do you want to invite? charlie
Successfully retrieved user details...
The signature is valid
bob >
```

```
> y
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 3
{'result': 'success', 'invitation': {'gro
upName': 'test', 'creator': 'bob', 'FIK':
 'NMERm8hJpfDCOhbsEj3oH2qMCdiYR9sSt/PwQJA
sXw8=\n', 'FSPK': '2Ao6eyCx8XO/zWvE74Rm5U
3Joz+HN38pcwEKW0AsZDQ=\n', 'FEK': 'otdHjl
20exG8wG5oXtDHUVyFH7Tlcfpas/z5nQRnTxI=\n'
, 'FDH': 'IpF3GK7lt4nIov021UASmiXD72sNra2
quEQhN7ZDgxg=\n'}}
You have an invitation!
Invitation to join group test from bob.
Would you like to join this group? (y/n)
>
```

Figure 12: Charlie gets Bob's invite

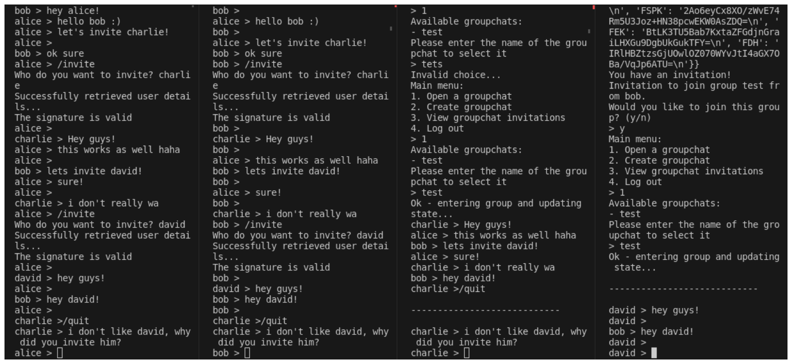


```
select it
> test
Ok - entering group and updating state...

----------------------------


alice >
bob > hey alice!
alice > hello bob :)
alice > let's invite charlie!
alice >
bob > ok sure
alice > /invite
Who do you want to invite? charlie
Successfully retrieved user details...
The signature is valid
alice >
charlie > Hey guys!
alice >
```

```
select it
> test
Ok - entering group and updating state...

----------------------------


bob > hey alice!
bob >
alice > hello bob :)
bob >
alice > let's invite charlie!
bob > ok sure
bob > /invite
Who do you want to invite? charlie
Successfully retrieved user details...
The signature is valid
bob >
charlie > Hey guys!
bob >
```

```
Would you like to join this group? (y/n)
> y
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 1
Available groupchats:
- test
Please enter the name of the groupchat to
 select it
> test
Ok - entering group and updating state...

----------------------------


charlie > Hey guys!
charlie >
```

Figure 13: Three-way messaging



```
bob > hey alice!
alice > hello bob :)
alice > let's invite charlie!
alice >
bob > ok sure
alice > /invite
Who do you want to invite? charli
e
Successfully retrieved user detai
ls...
The signature is valid
alice >
charlie > Hey guys!
alice > this works as well haha
alice >
bob > lets invite david!
alice > sure!
alice >
charlie > i don't really wa
alice > /invite
Who do you want to invite? david
Successfully retrieved user detai
ls...
The signature is valid
alice >
david > hey guys!
alice >
bob > hey david!
alice >
charlie >/quit
charlie > i don't like david, why
 did you invite him?
alice >
```

```
bob >
alice > hello bob :)
bob >
alice > let's invite charlie!
bob > ok sure
bob > /invite
Who do you want to invite? charli
e
Successfully retrieved user detai
ls...
The signature is valid
bob >
charlie > Hey guys!
bob >
alice > this works as well haha
bob > lets invite david!
bob >
alice > sure!
bob >
charlie > i don't really wa
bob > /invite
Who do you want to invite? david
Successfully retrieved user detai
ls...
The signature is valid
bob >
david > hey guys!
bob > hey david!
bob >
charlie >/quit
charlie > i don't like david, why
 did you invite him?
bob >
```

```
> 1
Available groupchats:
- test
Please enter the name of the grou
pchat to select it
> tets
Invalid choice...
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 1
Available groupchats:
- test
Please enter the name of the grou
pchat to select it
> test
Ok - entering group and updating
state...
charlie > Hey guys!
alice > this works as well haha
bob > lets invite david!
alice > sure!
charlie > i don't really wa
bob > hey david!
charlie >/quit

----------------------------

charlie > i don't like david, why
 did you invite him?
charlie >
```

```
\n', 'FSPK': '2Ao6eyCx8XO/zWvE74
Rm5U3Joz+HN38pcwEKW0AsZDQ=\n', '
FEK': 'BtLK3TU5Bab7KxtaZFGdjnGra
iLHXGu9DgbUkGukTFY=\n', 'FDH': '
IRlHBZtzsGjUOwlOZ070WYvJtI4aGX7O
Ba/VqJp6ATU=\n'}}
You have an invitation!
Invitation to join group test fr
om bob.
Would you like to join this grou
p? (y/n)
> y
Main menu:
1. Open a groupchat
2. Create groupchat
3. View groupchat invitations
4. Log out
> 1
Available groupchats:
- test
Please enter the name of the gro
upchat to select it
> test
Ok - entering group and updating
 state...

----------------------------

david > hey guys!
david >
bob > hey david!
david >
david >
```

Figure 14: If a user doesn't invite another user, they won't see their messages

8 References

1. **https://www.electronjs.org/**
2. **https://www.electronjs.org/docs/latest/tutorial/security**
3. **https://signal.org/docs/specifications/x3dh/**
4. **https://signal.org/docs/specifications/doubleratchet/**