# CLI Messaging Application

Nir Boneh

University of Colorado Boulder
nir.boneh@colorado.edu

**Abstract**

*This project discusses the creation and design of a decentralized CLI unix-based messaging application written in C. Examining closely on how to eliminate security risks and introspecting on the usefulness of the application in various situations. The advantages of using this application over other similar application in areas such as security, practicality, and speed is also discussed. Charts showing speed comparisons of sending messages over lan or wan are also shown. The source code for the project can be found at https://github.com/nboneh/CLIMessageApp and the repository can be used for downloading and installing the application.*

## 1. Introduction

Instant messaging has become one of the most practical ways we communicate with one another. In this day and age, instead of having to send mail across the physical plane and wait one to four business weeks to hear back from family relatives or friends, we can simply pull out our mobile device, type a message and send it in nearly an instant. While such power as reduced our face to face communication [1], one can't deny the advantages this tool could bring working within a unix terminal. For us developers and UNIX enthusiasts, spending most of our time on a terminal managing files and building application; A CLI tool that allows messaging quickly from the terminal, would be nice to have added to our vast arsenal of gadgets that are within our efficient terminal fingertips. Not to mention such an application would be extremely practical in a situation where we are limited to just a terminal, such as with special types of machines or on remote ssh. While a great messaging CLI application does exist, called telegram [2], it does not serve the purpose of being a decentralized application. That means that big brother could be watching your messages not to mention that security attacks could happen to the the telegram server which is stated to be vulnerable to MITM attacks according to a tech.eu article [5]. The other risk is that the server could go offline rendering the application useless. There is also a great chat application that is decentralized called chat terminal [3]. While it works great it's missing the key advantage of messaging which is being able to send a message while the other user is offline which could come in handy. Therefore, a messaging application that is decentralized is missing from the terminal tool arsenal which is the main focus of the CLI messaging application.

## 2. Design

The CLI messaging application will be programmed in C. There will be three main components to designing the CLI messaging application.

1. The CLI features, frontend
2. Communication between nodes
3. Avoiding security risks

### 2.1. The CLI features, frontend

A user will interact with an interactive CLI, it will provide a set of commands and features. The help command will let the user see all available commands and a description of how to use them. The user will be notified about this command when the user types in a wrong command. The add command that will take in two parameters one being the IP and the other being the nickname for the user. This will add a new contact that you can quickly setup communication with. The message command which will take either an ip address or a nickname and establish a communication messaging channel with that IP address. Once a messaging channel has been established one can see all previous messages with that IP address. The delete command will let one delete all previous messages with the user that is currently being messaged with. The quit command will also be available which will exit the application either by calling quit or pressing CTRL+D. The CLI will also notify the user when a new message is received.

## 2.2. Communication between nodes

A communication between two nodes will be done via TCP/IP connection using socket programming. Currently the application is limited to being able to establish a messaging channel between you and one other user, but might be expanded to group chat in future releases. Sicne a TCP connection cannot be established if another node is offline, the messaging will work as follows. You can send a message to an offline user, but he/she will only receive it later when both of you are online. This also makes the delete command more powerful, if the other user is offline you can delete the messages that you have sent.

## 2.3. Avoiding security risks

There are two main security risks that will be addressed when building the application. The first one being making sure to encrypt messages between two nodes so no one can decipher them, this will be addressed using RSA through the use of a public key and private key. The second problem is being able to share a public key on a non-secure channel which will be addressed using Diffie Hellman key exchange.

### 2.3.1 RSA for secure communication

The general idea of RSA works as follows, each user has a public key and a private key which they can generate using the RSA algorithm. Upon generating these two keys, a user can share his public key with everyone, but keeps the private key to himself. Now anyone who wants to share secure messages with another user, encrypts the message using the user public key, but the message can only be decrypted using the private key. Therefore, no one can read messages sent to the user other than the intended user himself. This will be the main method of communication when two users are sending messages back and forth using the application. The RSA algorithm involves generating two large prime numbers and multiplying them togther to get a composite, in order for anyone to crack the RSA algorithm it would require being able to figure out the two prime numbers using factoring which is extremely diffcult and would take a ridicilous amount of time doing by brute force, so one can trust RSA to be a rather secure algorithm.

### 2.3.2 Diffie Hellman key exchange

The only issue with the RSA method alone is that the user needs to share his public key with another user on a non-secure channel and this is where security risks are prone, this is where Diffie Hellman key exchange comes into play. One possible security risk that could happen when sharing a public key on a non-secure channel is called man in the middle attack. Let's imagine two users Alice and Bob, and a malicious user called Chris, here is how a man in the middle attack will play out:

1. Bob sends a package to Alice: Hey Alice this is Bob, here is my public key: Bob's key.

2. Chris intercepts the message and swaps out Bob's key with his own key.

3. Alice receives the message and think that Bob's key is actually Chris's key.

4. Alice sends back an encrypted message using Chris's key sharing her public key.

5. Chris intercepts that message as well, and decrypts it with his key and changes Alice's key to his key.

6. Bob receives the message and assume Alice's key is Bob's key.

7. From this point on Chris has got all the power over the conversation, being able to see it and also change it as he likes.

The only way to avoid such a terrible scenario is using the Diffie Hellman key exchange algorithm to share the public keys, the math is somewhat complicated, but the logic is rather simple and can be explained using colors:

1. Bob and Alice share a public color which they both agree on, Chris can know about this public color.

2. Bob and Alice both generate a private color neither one knows of the other and neither does Chris.

3. Bob and Alice mix the public color with their private color and send it to each other, Chris can know about this color.

4. Bob and Alice mix this color with their private color again and they both get the same color, now they both know a color that Chris doesn't know.

5. Bob and Alice can now send the public key using this fourth color for encoding without Chris being able to manipulate the situation.

While it is true that a man in the middle attack could disrupt the handshake, it will not be able to gain power over the conversation, since it can not know the fourth color. If a man in the middle attack those disrupt the process, the handshake can be retried over and over till he gets bored and moves on. So, basically the way the application will avoid security risks is by sharing public keys using Diffie Hellman and from there messages will be exchanged using RSA.

## 3. Evaluation

### 3.1. Methodology

In Order to compare the efficiency of the application, we will compare it to the well known CLI centralized messaging application called Telegram. Specifically we will compare the speed of sending a message using the CLI Messaging application and compare it to the speed of telegram. The CLI messaging application will be tested over WAN and LAN networks and the Telegram will be tested normally, since you have to send a message through a server. We will test the speed on short messages, medium messages, and long messages and average the speed of three instances of each. In the CLI Messaging application we will setup testing code to check on the speed of sending a message. For the telegram we will create what telegram calls a bot and send a message using CURL and eveluate the speed of the curl command using the UNIX time command. We would first obtain a chat id using

```
curl -s \
-X POST
\https://api.telegram.org/bot<token>
/getUpdates
 \| jq .
```

and then test the speed using the following curl command to send a message [4]:

```
curl -s \
-X POST
\https://api.telegram.org/bot<token>
/sendMessage
\-d text="A message from your bot"
\-d chat_id=65535 \
| jq .
```

### 3.2. Results

| Messaging Speed (Seconds) | | | |
|---|---|---|---|
| Message Length | Telegram | CLI Messaging App LAN | CLI Messaging App WAN |
| Short (10 chars) | 0.764 | N/A | N/A |
| Medium (50 chars) | 0.771 | N/A | N/A |
| Long (200 chars) | 0.782 | N/A | N/A |

**Table 1:** *Shows speed comparison of CLI Messaging App on LAN and WAN network as well as telegram*

## References

[1] Hemmer, Heidi. "Impact of Text Messaging on Communication." Minnesota State University, Mankato, 2009.

[2] "Telegram a New Era of Messaging." Telegram. N.p., n.d. https://telegram.org/

[3] "LAN Chat and Text Conferencing in Easiest Way with Vypress Chat." Chat Terminal. N.p., n.d. http://www.vypress.com/free_chat/

[4] Malizia, Nicola. "Getting Started with Telegram Bots." Unnikked. N.p., 25 June 2015. https://unnikked.ga/getting-started-with-telegram-bots

[5] Wauters, Robin. "Supposedly Super Secure Telegram App Is Vulnerable to MITM Attacks, Cybersecurity Expert Claims." Tech.eu. N.p., 29 Apr. 2014. http://tech.eu/brief/supposedly-super-secure-telegram-app-possibly-vulnerable/