Implementing Source-Tracking in a Ratchet based Messaging Scheme

CS6500 Project Report

Jude K Anil (CS21M025), Ravindra Kumar Vaishya (CS21M050)

May 16th 2022

1 Declaration

We declare that the work done in this project has not been and will be not used as-is for any other registered course at IITM; if this project code is extended further for any subsequent MTP project or MS/PhD thesis, this project work will be duly referenced in the reports/theses documents.

2 Abstract

Widely used popular messaging services such as Whatsapp and Signal have implemented secure end-to-end encryption ensuring privacy for users. But this message confidentiality makes content moderation difficult for the platforms. There have been cases of inflammatory and other problematic content being shared/forwarded among a large number of people before action could be taken. Even after being aware of the content being shared, the origin of messages can be hard to identify. One way to mitigate this is to incorporate techniques into the messaging system that allow authors of reported messages to be identified. But such modification goes against the privacy guarantees of the messaging service. This form of information disclosure can be abused if not properly handled. Our work is an implementation of a technique called *source-tracking* on top of a secure messaging framework based on double ratchet that allows user-reporting of messages while safeguarding the privacy of nearly all the participants except for the original author of the message.

3 Introduction

In the current world, instantaneous messaging has become ubiquitous and with it there has been a rise in proliferation of harmful or offensive content. As laws are being enacted by various countries to curb this, there arises a need to have sound and reliable techqniues that allow for the limiting of such content while also safe-gaurding the privacy of its users. One such techqniues focuses on identifying the sources of reported offensive messages. But most such technqiues require either too much metadata or compromises user privacy too much or are limited in their scope. One such techniques is *message-franking* which is used to identify the immediate sender of reported messages.

The notion of identifying the source of malicious messages on a complaint originated in 2019 with Tyagi et al. [3]. They a developed a scheme called *message traceback*, which addresses this issue by allowing a messaging platform to recover the path of a forwarded message after a user reports it for malicious content. This technique guaranteed the privacy of the all users before message reporting and after reporting, only the participants in the forwarding chain of the reported message would only be identified. On the other hand, such disclosure of forwarding information may be seen as too much by some security conscious users. This technique required the platform to store a small amount of metadata per message which can become a problem for very large platforms like Whatsapp that handles billions of messages daily. Moreover, the technique took linear time on the length of the forwarding chain to discover the original author. Source-tracking, introduced by Peale et.al [2], is an alternative technique that guarantees greater privacy by disclosing only the original authors identity while keeping the forwarding chain confidential. It does not require any persistent storage on the side of the platform for each message as all the necessary metadata is carried within the message itself and requires only constant time for looking up the origin of a message after reporting. There are two alternate schemes described in the paper, tree-linkable and tree-unlinkable source-tracking. Here tree-linkable refers to the possibility of a receiver of a forwarded message to distinguish the forwarding tree the forwarded message is a part of. Tree-unlinkable scheme relies on algebraic MACs and El-Gamal Encryption and is more technically challenging that Tree-linkable scheme. Moreover, the additional sophistication increases the computational time required. Due to time constraints, only tree-linkable scheme has been implemented.

4 Problem

Consider a set of users using an end-to-end encrypted messaging service. Conceptually, every new message creates a tree rooted at the author with an edge to the recipient user. If and when the recipient forwards the received message to a new user, the tree (now known as a forwarding tree) is extended with an edge from the former to the latter user. Since everything is end-to-end encrypted, for any message, other than the sender and receiver of that message, no other party (including the messaging platform) knows the contents of the message nor whether it is a forward or nor between whom the message was shared (except for platform). If a message is reported then its contents become known to the messaging platform.

The problem to solve is, given a reported message find the user at the root of the tree while maintaining the confidentiality of the non-participants users. Also for non-reported messages, it should have the same security guarantees as without the source-tracking scheme. Note that the scheme itself cannot identify whether a reported message qualifies as being abusive or not. This must be handled separately.

5 Concepts and Design

5.1 Protocols

There are two sets of protocols or schemes: one for secure messaging called double ratchet and another for reporting of abusive messages called source tracking

5.1.1 Source Tracking

The particular scheme used is Tree-linkable source tracking. The scheme is said to be tree-linkable because when a user receives a forwarded message having plaintext m and then later receives another forward of the same plaintext m, the user is be able to tell whether the two messages are from the same forwarding tree or from different trees whose messages happen to have the same plaintext.

The scheme is a set of protocols and algorithms for computation and communication between the user and platform. The whole thing runs on top of an end-to-end encrypted messaging system by augmenting the usual protocol for sending and receiving messages and by adding a new Report protocol. The protocols make use of the underlying messaging scheme as a black box to provide authenticated encryption and protection against replay attacks. The protocols are briefly described below:

1. *keyGen* Algorithm:
   - Runs on the platform.
   - Generates (pk, sk) where $pk = K_{pub}$, $sk = (K_{pvt}, K_{sym})$ where K_{pvt}, K_{pub} are asymmetric keys used for signing and K_{sym} is used for encryption by the platform.

2. *newUser* Algorithm::

   - Runs on the platform.
   - Takes the userid of the authorized new user and adds it to the set of users $\mathbf{U}'$.

3. *AuthMsg* Protocol:

   - It is used when a new message is created and sent by a user.
   - On User Side: The user generates a commitment $c_m = Commit(m)$ of the plaintext m along with commitment randomness r, the complete message then becomes $(m, \perp, c_m, r)$ where $\perp$ is padding used to hide from the platform that it is a new message. This is then send via the messaging scheme to the recipient user. c_m and message-id e is sent to platform.
   - On Platform Side: The platform receives c_m from user. It will encrypt the sender id to get $src = Enc(K_{sym}, (u_s, md))$ where u_s is the id of sending user, md is metadata associated with the message (empty in our implementation). The encrypted sender id and commit are then signed to get $\sigma = Sig(K_{pvt}, (src, c_m))$. The platform data is then formed, $pd = (\sigma, src)$.

4. *FwdMsg* Protocol: Similar to AuthMsg except the commit $c_m = Commit(\perp)$ and the fd is from the original message, so the message now is (m, fd, c_m, r).

5. *RecMsg* Protocol:

   - On Platform side: Receives request for pd of message with id e and responds with the appropriate pd.
   - On user side: The recipient user requests the pd using message id of the received message. After receiving pd, it checks whether the message is a new forward or not and performs commit and sign verification. If the message is new, then forwarding data is generated as $fd = (\sigma, src, c_m, r)$ from pd and received message. If not new, then the same fd is reused for the next forward.

6. *Report* Protocol:

   - User side: User reports a message. It sends m and corresponding fd to platform.
   - Platform side: calculates $u_s, md = Dec(K_{sym}, src)$ and performs appropriate policy action against the user if found guilty.

Briefly, for every message sent, the platform creates a signature on $(Enc(K_{sym}, U_s), c_m)$. This ties the author to the message content via the commit. When the message is forwarded for first time, the forwarder includes a fd which has the signature, encrypted source, commitment and randomness, alongside plaintext message. This fd is then passed along each such forward without modification. The signature is verified by each receiver in order to ensure authenticity. A reporter sends the message plaintext and fd to the platform who verifies the signature and decrypts the identity of original sender. Since fd is inside the encrypted message while being sent between users, platform confidentiality is ensured, encrypting original sender identity by platform before signing hides it from the users while they send and receive the messages. Accountability and unforgeability is ensured by the encrypted sender's identity and the signature.

The following security goal are met by the scheme:

1. **User Confidentiality**: The confidentiality of non-reported messages is no different than the original messaging mechanism as the user id of the source is encrypted by the platform. Moreover, outside of the platform, no one else is aware of the source id of a reported message.

2. **Platform Confidentiality**: Makes the same guarantee for the platform before a message is reported and additionally requires that, even after a report, the platform learns only the source and associated metadata of a reported message and nothing more like the reported message's forwarding history. This is gauranteed by the messaging scheme encryption.

3. **Accountability**: No malicious user should be able to send a message which cannot later be traced back to them. Every malicious message can traced back to its author. This is ensured by the platform signing the user id that has been authenticated by the underlying messaging platform.

4. **Unforgeability**: No malicious user should be able to frame another user for sending a message it did not send. The identity of the author of a reported message cannot be forged by another user.

5.1.2 Double Ratchet

The Double Ratchet algorithm (previously called Axolotl Ratchet) is key management algorithm designed by Trevor Perrin and Moxie Marlinspike in 2013. It is the current state of Art algorithm used in many big platforms like WhatsApp and Signal.
After the initial key exchange (using X3DH for better security), it manages for the renewal and maintenance of short-lived session keys. It combines hash function based KDF (key derivation function) ratchet (provides future/post-compromise/backward Secrecy) and DH based ratchet (provides Forward Secrecy i.e. if a key is compromised the messages till next DH key exchange will only be compromised) therefore it is called Double Ratchet.
Sources: double ratchet doc, Wikipedia, X3DH doc, blog

- **X3DH (Extended Triple Diffie-Hellman):** Combining X3DH with PKI (public key infrastructure) for the retention of pregenerated one-time keys (prekeys), it will allow for initialization of messaging sessions without the presence of remote peer (asynchronous communication). X3DH with prekeys and double ratchet algorithm provides confidentiality, integrity, authentication, participant consistency, destination validation, forward secrecy, backward secrecy (aka future secrecy), causality preservation, message unlinkability, message repudiation, participation repudiation, and asynchronicity. The output of X3DH is Shared secret keys which are used in KDF functions.

- **KDF (Key Derivation Function):** Given a initial key, it is used to generate the future keys. IT provides future/post-compromise/backward Secrecy i.e. if a key is compromised and known to attacker, he will not be able to recover the past keys. It can be shown as motion of a mechanical ratchet which moves in only one direction. Some part of the KDF function's output is used to update the state of the KDF function and rest part of the output is used as message key.1 and 2
At a time, during the Double Ratchet protocol there are 6 KDF functions running in parallel (Root KDF, Sender Ratchet and Receiver Ratchet) and all above on either side.

5.2 Design

The messaging platform is based on Double-Ratchet. For the source tracking scheme, the user side runs on the messaging client side while the platform is a separate process interacting with the user side through sockets. The platform generates a separate thread for handling each user. The user side maintains a map of message ids and fds. Different users send messages between them through ratchet while for scheme related message exchanges, the separate socket connection with the platform is used. The platform maintains a global store of current pds. Current pds are those that have been created recently and have not been requested by any recipient user. Once the request is made, the pd is removed form the store. This ensures that memory overhead is kept to a minimum.

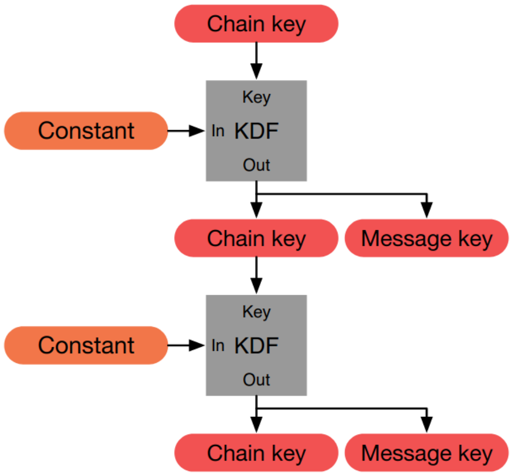

Figure 1: Symmetric Key Derivation Function(KDF)

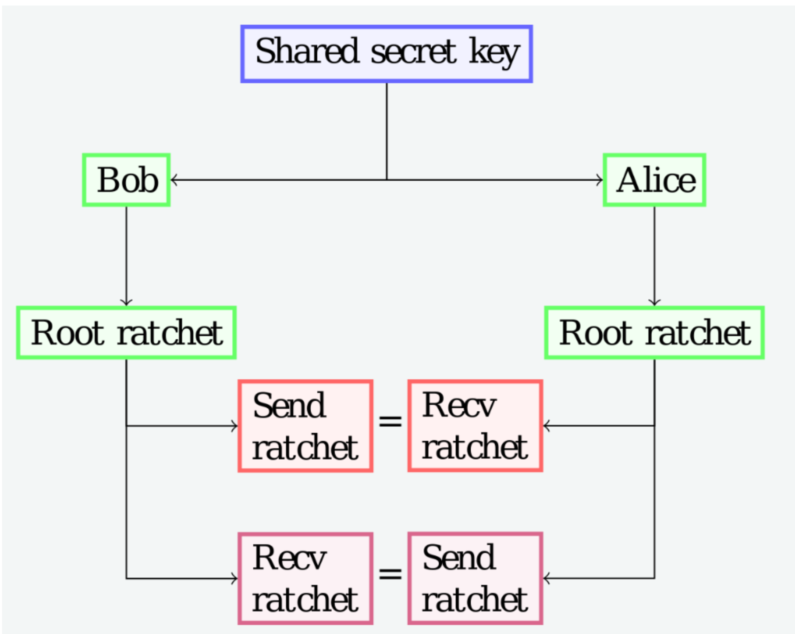

Figure 2: Key Generation and Mapping

6 Implementation

6.1 Development Environment

- Network Stack: TCP/IP
- Programming Language: Python 2.7.
- Versioning: Git hosted on Github.
- Messaging Platform: Double Ratchet with X3DH implementation.

6.2 Code Structure

6.2.1 Classes

1. UserTreeLinkable: An object of this class corresponds to a user in the tree-linkable scheme. Its methods capture the user-side protocol and algorithms of the scheme.
   - author_msg: Takes a plaintext message and generates commit for the message. It then sends the commit and message id to platform for generating pd.
   - receive_msg: Takes a message received from another user, extracts the commit and fd. Then fetches the pd from platform using message id. It performs the necesary verification of commits and signs and then returns the message containing the plaintext, updated fd, commit and randomness.
   - forward_msg: Similar to author_msg but commit is on BOT constant.
   - report_msg: Sends the decrypted plaintext of the message along with the fd to the platform.
2. PlatformTreeLinkable: An object of this class corresponds to a user in the tree-linkable scheme. Its methods capture the platform-side protocol and algorithms of the scheme.
   - register_user : Adds new users to its set of current users.
   - generate_pd: Receives commit from the sender along with message id. Generates the pd and stores it using id as key. (pd is deleted once the recipient user ask for it)
   - report_msg: Recieves the plaintext message and fd from reporter. Verifies the sign in fd, commit and then decrypts src to get the userid.
3. Axolotl: It provides the double ratchet protocol with X3DH initialization of keys. It saves the keys in database (for persistence). It provide the api for encryption and decryption of messages.
   - create_conversation : It sets all the keys, create database and saves the keys and related constants.
   - encrypt: It encrypts the message.
   - decrypt: It decrypts the message.

6.2.2 Functions

- handle_user_scheme1: It handles a connection with a client user. It handles all of the messages passed as part of the protocol between platform and user.
- receiveThread: It performs the actions needed when a message is received.
- chatThread: It performs the actions needed to send a message.
- ReportGUI: It displays the name of reported the person or an error message.

6.2.3 Constants

- BOT: Used as an empty value for generating commits and for empty fd padding inside messages.
- COMMIT_SIZE: 32. It is the number of bytes in a commit.
- R_SIZE: 32. It is the number of bytes in the random number used in commit generation.
- RSA_KEY_SIZE: 2048. Size of the RSA key used by Platform.
- CTR_NONCE: 16 bytes. Counter mode nonce.
- AES_KEY: 32 bytes. It is the AES key used by Platform.

6.2.4 Libraries

- Standard Python Libraries:
  - pickle: For serializing python objects.
  - binascii: For base64 encoding.
  - socket: For socket programming.
  - argparse: For commandline argument parsing.
  - time: For ensuring enough time passes for proper transmission via sockets.
  - os: For file and folder manipulation.
  - sys: For availing system related methods.
  - threading: For multi-threading.
  - Tkinter: for gui
- nacl: for cryptography
- sqlite3: for database
- pyaxo: for key management and encryption and decryption
- Cryptography version 2.7: For cryptographic primitives like SHA256, RSA, AES etc.

6.2.5 Miscellaneous Files

- platform_pub_key.pem: Contains the public key of the platform.
- fd_file_<userid>.pkl: Contains the fds indexed by message ids of previous chats of userid. (The fds in the files are valid only for the current platform session).
- <userid>.db: Contains the conversation history between <userid> and other users.

7 Testing

The following tests were done.

1. Correctness of implementation through unit tests. Tests of User and Platform Objects can be found in test_protocol.py file.
2. Miscellaneous.

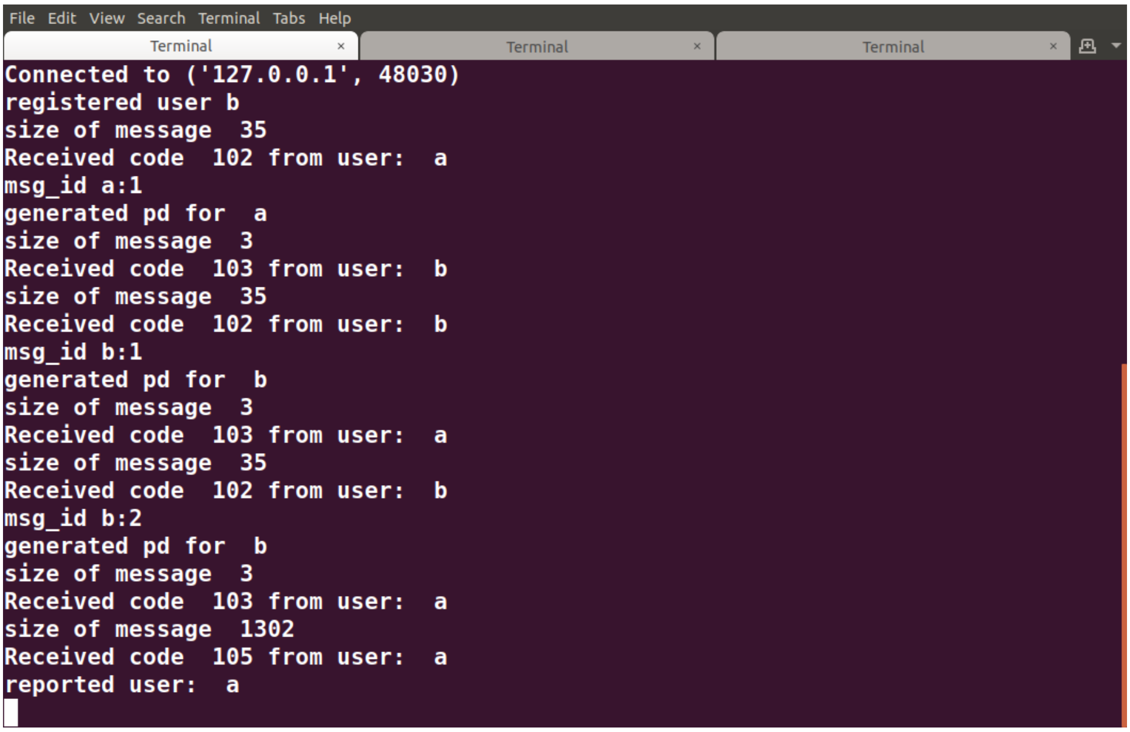

Figure 3: Platform

7.1 Sample Example (with Wireshark tracking):

- 3: Platform stdout.
- 4: Client "a" stdout.
- 5: Client 'b' stdout.
- 6: Reported person.

7.2 Wireshark outputs:

Between two clients:

- 7: Between client "a" and client "b": The encrypted message between two clients.

Following are the interactions between client and Platform for the source tracking.

- 8: Client to platform: To notify that the current message id is being send (can be a forwarded message too). With the messageid client sends the commit too.

- 9: Client to platform: When a client receiver receives a message it asks platform for pd/fd data associated with the given message id.

- 10: Platform to client: Platform response to the receiver client's request for pd/fd.

- 11: client to platform: Clients sends the actual message with message id to platform to report it. (Note Before this step, platform doesnot know the message and in all the above steps it only received the message id only).

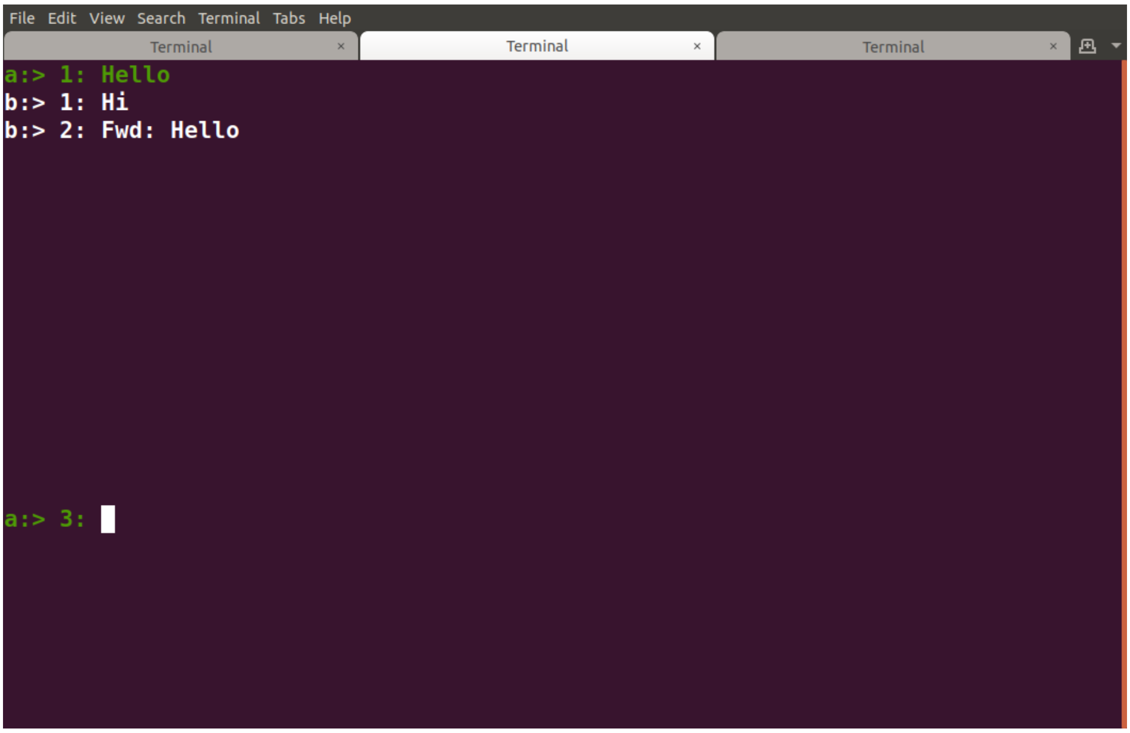

Figure 4: Client ”a”

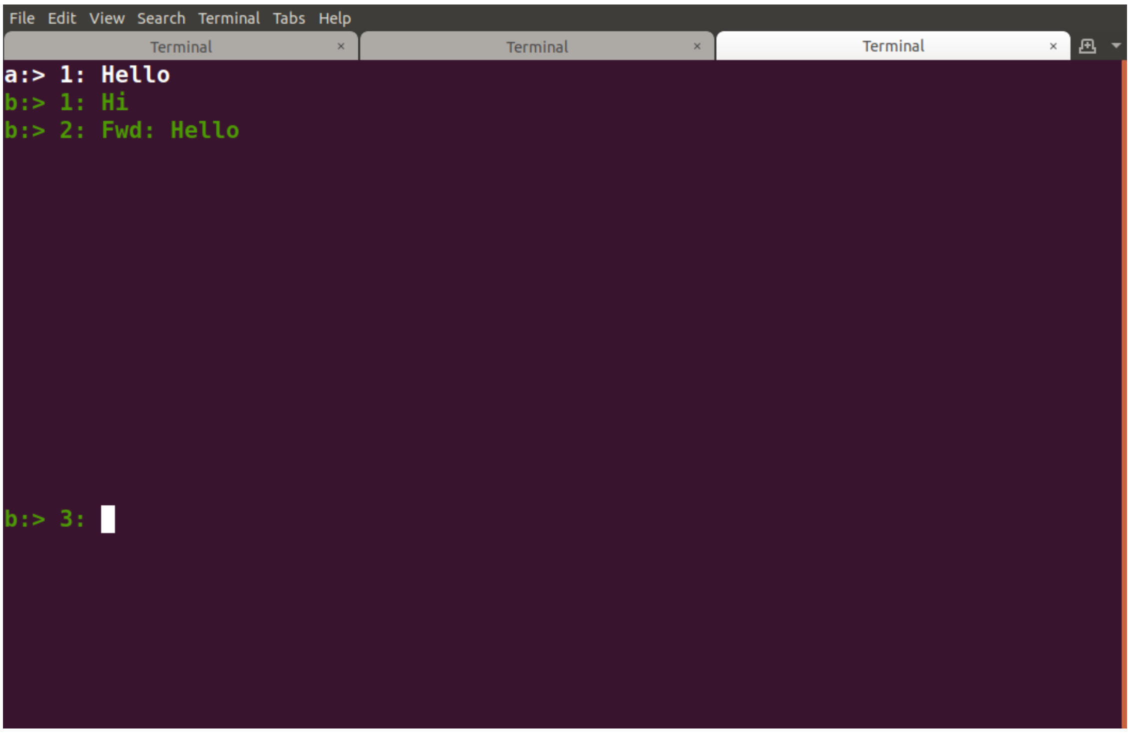

Figure 5: Cleint ”b”

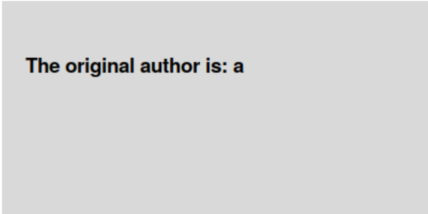

Figure 6: Reported person for b:2 message.

```
0040  fa 80 1c f6 e4 1f 1a 3e  f7 9b c8 5d 15 58 7f df   ······> ···]·X··
0050  7a dc 7f 0e 8b a1 c5 94  b7 73 86 a0 d0 44 b5 31   z······· ·s···D·1
0060  e7 b4 1e 89 17 d4 f5 6e  91 db 9a a3 8a 77 ad c5   ·······n ·····w··
0070  b7 e3 b5 72 99 d3 f7 47  2e 7d 0e 35 f1 c4 c8 e6   ···r···G .}·5····
0080  af 23 5e f1 24 ff b9 a2  b5 c0 ce 4e 02 6e 2b 56   ·#^·$··· ···N·n+V
0090  8f c0 20 41 d8 04 1d 90  ed 4d 51 e3 3b 44 ff 2f   ·· A···· ·MQ·;D·/
00a0  ab 4d 9d fb 5e fc 11 72  7d 91 6c 1c 3c c0 e4 ed   ·M··^··r }·l·<···
00b0  b9 07 99 31 1c e8 78 ac  6f db 1c ff 85 ae 37 cf   ···1··x· o·····7·
00c0  40 0c b3 fb 2c 16 3d 36  d7 f9 55 bc c4 eb f0 90   @···,·=6 ··U·····
00d0  8f 24 eb 26 3b 83 41 b1  04 65 85 a5 de c1 95 e4   ·$·&;·A· ·e······
00e0  74 b3 cb 03 e4 a4 00 95  80 1f f7 6a d1 40 24 f8   t······· ···j·@$·
00f0  ad ab 52 92 11 d4 6f 3b  10 6c 73 79 42 05 aa 7f   ··R···o; ·lsyB···
0100  68 2a ca e9 79 4f 47 8d  c2 a9 ed 18 90 f3 a3 c2   h*··yOG· ········
0110  b0 67 48 bd 48 0f dc 54  33 6b 96 08 10 4f 5f b3   ·gH·H··T 3k···O_·
0120  36 d4 93 8c a8 64 d0 81  23 98 60 e8 4b 4c 98 f9   6····d·· #·`·KL··
0130  4f 71 08 05 2d 86 de 5d  5a ad 98 b6 e5 cb ec ba   Oq··-··] Z·······
```

Figure 7: Encrypted message transfer between both clients (a=>b)

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 5a 92 5b 40 00 40 06  aa 40 7f 00 00 01 7f 00   ·Z·[@·@· ·@······
0020  00 01 bb 96 2b 67 22 2b  69 66 69 5f 75 4e 80 18   ····+g"+ ifi_uN··
0030  02 00 fe 4e 00 00 01 01  08 0a e1 96 8e cd e1 95   ···N···· ········
0040  4a 61 31 30 32 f3 7e e4  96 05 07 46 4b 87 54 48   Ja102·~· ···FK·TH
0050  32 c0 da bf fb 38 94 cb  bc bb 64 2f 31 39 f7 d8   2····8·· ··d/19··
0060  c7 92 d8 06 ac 61 3a 31                            ·····a:1
```

Figure 8: 102 message: Part of author/forward message protocol (Client=>Platform)

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 3a 71 37 40 00 40 06  cb 84 7f 00 00 01 7f 00   ·:q7@·@· ········
0020  00 01 bb 9e 2b 67 fc 2e  8f ed cc 4f a9 8f 80 18   ····+g·. ···O····
0030  02 00 fe 2e 00 00 01 01  08 0a e1 96 90 c7 e1 95   ···.···· ········
0040  d5 74 31 30 33 61 3a 31                            ·t103a:1
```

Figure 9: 103 message: Part of Part of Receive message (request pd) (Client=>Platform)

```
0040  90 c7 31 30 34 68 4a 58  57 44 57 56 75 54 79 43   ··104hJX WDWVuTyC
0050  31 41 57 77 64 53 6c 51  2f 41 36 56 67 71 53 59   1AWwdSlQ /A6VgqSY
0060  76 5a 49 63 79 59 72 63  47 32 64 6c 75 35 66 34   vZIcyYrc G2dlu5f4
0070  73 42 6c 42 50 2f 69 68  2f 35 75 73 4f 59 71 6c   sBlBP/ih /5usOYql
0080  6d 66 69 44 33 50 2f 42  6f 46 58 78 49 31 66 43   mfiD3P/B oFXxI1fC
0090  53 4b 58 4c 77 70 2f 77  4f 73 45 69 6e 67 69 4d   SKXLwp/w OsEingiM
00a0  33 49 6d 79 4d 30 63 64  65 66 73 37 6d 2f 53 4e   3ImyM0cd efs7m/SN
00b0  57 2b 6a 55 77 34 54 73  37 67 73 43 4a 43 6c 63   W+jUw4Ts 7gsCJClc
00c0  51 49 45 63 4f 4f 51 4e  4f 42 2b 6b 47 63 70 71   QIEcOOQN OB+kGcpq
00d0  61 30 44 68 44 41 35 64  63 77 78 6d 49 47 53 49   a0DhDA5d cwxmIGSI
00e0  49 67 63 30 47 45 6c 71  64 4b 6f 73 4a 66 6a 49   Igc0GElq dKosJfjI
00f0  44 55 53 54 53 6f 45 45  4f 72 68 4e 38 63 63 4a   DUSTSoEE OrhN8ccJ
0100  61 63 77 45 68 62 34 36  32 7a 62 4a 52 6d 55 65   acwEhb46 2zbJRmUe
0110  33 52 4e 31 2f 4e 73 51  48 32 38 51 5a 74 30 62   3RN1/NsQ H28QZt0b
0120  50 4c 6c 4b 4b 36 58 52  69 5a 6c 68 46 5a 65 79   PLlKK6XR iZlhFZey
0130  52 6a 47 4e 68 30 6d 46  73 59 51 69 71 2b 37 49   RjGNh0mF sYQiq+7I
```

Figure 10: 104 message: Part of Part of Receive message (returns pd) (Platform=>Client)

- 12: Platform to client: Reported person's userid.

```
0040  28 9b 31 30 35 4b 46 4d  6e 58 48 67 34 4e 46 78   (·105KFM nXHg4NFx
0050  34 4f 54 56 63 65 47 51  32 58 48 4a 6c 62 6b 38   4OTVceGQ 2XHJlbk8
0060  67 58 48 68 69 4e 56 78  34 4d 44 46 73 58 48 67   gXHhiNVx 4MDFsXHg
0070  78 5a 45 70 55 50 31 78  34 4d 44 4e 63 65 47 45   xZEpUP1x 4MDNceGE
0080  31 59 46 78 34 59 54 6b  6d 4c 32 52 63 65 44 67   1YFx4YTk mL2RceDg
0090  33 4d 6d 4a 63 65 47 49  33 58 48 67 77 4e 6c 78   3MmJceGI 3XHgwNlx
00a0  34 5a 44 6c 63 65 47 51  35 62 6c 78 34 5a 54 56   4ZDlceGQ 5blx4ZTV
00b0  63 65 47 5a 6c 4c 46 78  34 4d 44 5a 51 54 31 78   ceGZlLFx 4MDZQT1x
00c0  34 5a 6d 55 6f 58 48 67  33 5a 6c 78 34 5a 54 5a   4ZmUoXHg 3Zlx4ZTZ
00d0  63 65 47 56 69 58 48 67  77 5a 57 4a 63 65 47 45   ceGViXHg wZWJceGE
00e0  35 5a 6e 34 67 58 48 68  6d 4e 7a 39 63 65 47 59   5Zn4gXHh mNz9ceGY
00f0  77 61 46 78 34 4d 54 56  38 53 46 78 34 5a 44 56   waFx4MTV 8SFx4ZDV
0100  63 65 47 59 77 58 48 67  35 4d 69 6c 79 58 48 68   ceGYwXHg 5MilyXHh
0110  6d 4d 46 78 34 59 54 64  63 65 47 5a 6a 58 48 67   mMFx4YTd ceGZjXHg
0120  77 5a 56 78 34 59 6a 42  49 58 48 68 68 4e 31 78   wZVx4YjB IXHhhN1x
0130  34 4f 44 49 6a 4e 79 4a  73 58 48 67 34 59 31 78   4ODIjNyJ sXHg4Y1x
```

Figure 11: 105 message: Part of Report request (Client=>Platform)

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 38 39 88 40 00 40 06  03 36 7f 00 00 01 7f 00   ·89·@·@· ·6······
0020  00 01 2b 67 bb 96 69 5f  78 36 22 2b 6e b1 80 18   ··+g··i_ x6"+n···
0030  02 00 fe 2c 00 00 01 01  08 0a e1 97 e6 8f e1 97   ···,···· ········
0040  e6 8d 31 30 36 61                                  ··106a
```

Figure 12: 106 message: Part of Report response (Platform=>Client)

8 Learning Experience

1. Understand the basics of encrypted messaging from security perspective.
2. Understand source-tracking and its implementations.
3. Understand the widely used Signal's Double Ratchet protocol for E2E message encryption.
4. Understand the Extended Triple Diffie-Helman (X3DH) algorithm to establish symmetric key.
5. Learning of methods that help identify malicious users and thus improve the usability of the platform.
6. Learned to design security protocols.
7. Given a design, learned to find the drawbacks and advantages of using one security mechanism over other.
8. Gain experience of implementing security protocols in practical settings.

9 Conclusion

With ever-increasing need for curbing the spread of harmful content like misinformation through instant messaging platforms like Whatsapp, it is necessary to adopt techniques like source-tracking which not only provides a efficient way for identifying bad-actors but also do it while respecting the privacy of the users (from other users as well as from platform). Such techniques will be vital in safeguarding privacy in an uncertain world as more and more countries censure free speech and expression. Compared to other techniques, source-tracking has minimal overhead and hence can be scaled to the largest of web-services. Thus it is worthwhile exploring various possible implementations of this technique in different settings.

10 Suggestions

Regarding the course, the course syllabus was a little heavy for a single course as there are many things to remember. IT would had been better if for Endsem exam, only portion after midsem would had been in syllabus. Regarding assignments and project, they are time consuming but considering the extra time we are given, it made it a little lenient (though helped a little for 4 courses students as they had Endsem till 14th May and many assignments deadlines after Endsem). IF there were 2-3 courses in current semester than it would had been an well distributed assignments.

References

[1] Melissa Chase, Trevor Perrin, and Greg Zaverucha. *The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption*, page 1445–1459. Association for Computing Machinery, New York, NY, USA, 2020.

[2] Charlotte Peale, Saba Eskandarian, and Dan Boneh. Secure complaint-enabled source-tracking for encrypted messaging. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1484–1506, New York, NY, USA, 2021. Association for Computing Machinery.

[3] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. Traceback for end-to-end encrypted messaging. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 413–430, New York, NY, USA, 2019. Association for Computing Machinery.