# Implementing Secure Complaint-Enabled Source-Tracking for Encrypted Messaging

CS6500 Project Proposal

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

March 16th 2022

## 1 Protocol

We propose as our project implementing the two constructions for source-tracking" as described in [2]. Source-tracking refers to identifying the source of malicious messages (especially message forwards) reported by a user within the context of an encrypted messaging platform while preserving privacy guarantees of the platform. The source-tracking approach is differs over the previous work on this problem called traceback described in [3] in that the identities of the receivers of reported messages are kept private, there is no metadata required to be stored per message and the time taken for finding the source is independent of the size of the forwarding chain.

A source-tracking scheme is build 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. Each protocol has the general form

$$(out_{user}, out_{platform}) \leftarrow \langle U(user-secrets), P(platform-secrets)\rangle(public-parameters)$$

where $out_{user}$, $out_{platform}$ are the values known to the user/platform respectively on successful completion of the protocol, $U$, $P$ are the algorithms implementing the user/platform side of the protocol. The protocols are briefly described below:

1. *KeyGen* Algorithm: $KGen(pp) \rightarrow (pk, sk)$ where $pp$ is the public parameters and $(pk, sk)$ are the platform key-pair.

2. *NewUser* protocol: $(ad, \mathbf{U}') \leftarrow \langle U_{new}, P_{new}(sk, \mathbf{U})\rangle(U_n, pk)$ where $ad$ is authoring data which the user uses when creating a message, $\mathbf{U}'$ is the updated membership set of the platform, $U_n$ is the id of the new user.

3. *AuthMsg* Protocol: $((ad', e), (pd, e)) \leftarrow \langle U_{auth}(msg), P_{send}(sk, md)\rangle(U_s, U_r, pk)$ where $ad'$ is the updated authoring data for future messages, $pd$ is the data used when message is delivered to recipient, $e$ is the identifier for the message sent by underlying messaging mechanism, $msg = (m, ad)$, $m$ is the plaintext message written by user, $md$ is any metadata the platform may wish to keep in case the message gets reported.

4. *FwdMsg* Protocol:$((fd', e), (pd, e)) \leftarrow \langle U_{fwd}(msg), P_{send}(sk, md)\rangle(U_s, U_r, pk)$ where $fd'$ is the updated forwarding data to be used if the user wishes to forward $m$ again. Since the platform shouldn't be able to distinguish between an authored and a forwarded message, the platform protocol $P_{send}$ is identical to the platform's protocol for an authored message.

5. *RecMsg* Protocol:$((m, fd), \perp) \leftarrow \langle U_{rec}, P_{rec}(sk, pd)\rangle(U_s, U_r, e, pk)$ where forwarding data $fd$ to be used by the user when forwarding or reporting this message.

6. *Report* Protocol: $(fd', (U, md)) \leftarrow \langle U_{rep}(fd), P_{rep}(sk, md)\rangle(U_s, U_r, pk)$ where $fd'$ is updated forwarding data (unnecessary), $U$ is the id of the author of the message.

The protocols ensure the following security goals:

1. Confidentiality: In addition to the confidentiality of non-reported messages as guaranteed by the underlying messaging mechanism, the following kinds of confidentiality are preserved as well.
   (a) Tree-unlinkable: A user who receives two different forwarded messages should not be able to tell if they belong to the same forwarding tree or not.
   (b) After a report, the messaging platform should not learn anything new about the reported message's forwarding history other than the identity of the source.
2. Accountability: Every malicious message can traced back to its author.
3. Unforgeability: The identity of the author of a reported message cannot be forged by another user.
4. Deniability: Only the platform no-one else identify a message with its author.

There are two constructions of source tracking described in the paper: tree-linkable and tree-unlinkable.

1. Tree-linkable: This scheme of the source-tracking approach makes it possible for a receiver to know whether two identical message originated from the same source or not. The scheme is as follows:
   (a) *KeyGen* Algorithm: $(pk, sk)$ where $pk = K_{pub}$, $sk = (K_{pvt}, K_{sym})$.
   (b) *NewUser* protocol: Replaced with a single algorithm *newUser* which returns $\mathbf{U}'$. *ad* is not returned as it is not required in this scheme.
   (c) *AuthMsg* Protocol: The user generates a commitment $c_m = Commit(m)$ along with commitment randomness $r$, the message is $(m, \perp, c_m, r)$, as *ad* is not used. This is then send via the messaging scheme. The platform receives $c_m$ from user. $P_{send}$ will append $pd = ((\sigma, src), e)$ where $\sigma = Sig(K_{pvt}, (src, c_m)$, $src = Enc(K_{sym}, (U_s, md))$ to the encrypted message before delivery.
   (d) *FwdMsg* Protocol: Similar to AuthMsg except the commit $c_m$ is derived from the empty message. Also the $fd$ from *RecMsg* is included in the message before sending to the messaging scheme.
   (e) *RecMsg* Protocol: Checks whether the message is a new forward or not. If new, then forwarding data is generated, $fd = (\sigma, src, c_m, r)$ from *pd* and received message. If not new, then the same $fd$ is returned again.
   (f) *Report* Protocol: The source id is got by $Dec(K_{sym}, src)$

   Breifly,for every message sent, the platform adds a signature on $(Enc(K_{sym}, U_s), c_m)$ to it. The signature is verified by each receiver. If a message is forwarded for first time, 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. A Reporter sends the message plaintext and $fd$ to the platform who checks the signature and decrypts the identity of original sender.

   Since $fd$ is inside the encrypted message, it ensures platform confidentiality, encrypting original sender identity provides user confidentiality. Accountability and unforgeability is ensured by the encrypted sender's identity and the signature.

2. Tree-Unlinkable: This scheme of the source-tracking approach prevents receivers from knowing whether two identical message originated from the same source or not. The scheme is as follows:

   The overall workflow is similar to the previous scheme but is a bit more involved. The difference lies in preventing the use of $fd$ to identify different forwarding trees. Users re-randomize $fd$ by re-randomizing the contents of the signature and proving the validity of the re-randomized values in zero-knowledge to the platform, which then issues the user a fresh signature on those values. This is similar to a keyed-verification anonymous credential scheme that is modified to allow for anonymous and unlinkable credential delegation via forwarding. The attributes included in each credential are an encryption of the source identity and metadata as well as a hash of the message content. As the attributes are a mix of group elements and scalars, in order to allow users to efficiently prove properties about these attributes the keyed-verification anonymous credential scheme presented in [1] is used. This construction relies on an algebraic MAC rather than a signature for the credential.

## 2 Implementation

- Programming Language: Python 3.8.
- Versioning: Git hosted on Github.
- Messaging Platform: Double Ratchet implementation of Signal.

## 3 Expected Learning Outcomes

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 variants of Diffie Helman algorithm to establish symmetric key.
5. Learning of methods that help identify malicious users and thus improve the usability of the platform.
6. Gain experience of implementing security protocols in practical settings.
7. Familiarity with El-Gamal Encryption, algebraic MAC and zero knowledge proofs.

## 4 Testing Plan

The testing will fall very broadly into the following:

1. Correctness of implementation through unit tests.
2. Stress-testing of platform.
3. Miscellaneous.

The details will be worked out during development.

## 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.