Session: Secure Messenger with additional Measures for Metadata Protection

Johannes Martin Löbbecke, Holger Kinkelin*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: loebbeckejohannes@hotmail.de, kinkelin@net.in.tum.de

***Abstract*—With the widespread use of end-to-end encryption in popular messengers, the security of direct messages has improved significantly over recent years. However, due to the lacking protection of metadata, like a user's IP address and contact list, the privacy of users is not completely protected. The open-source application Session aims to provide the features expected from a modern chat application while implementing additional measures to protect the user's metadata. This is achieved by using public keys as an identifier and operating on a decentralized anonymous network to send end-to-end encrypted messages. This paper presents the techniques and approaches that Session uses to provide additional privacy protection in comparison to other popular messengers.**

***Index Terms*—privacy protection, end-to-end encryption (E2EE), metadata, decentralised, anonymous network**

1. Introduction

Privacy protection in communication software has been gaining increasing awareness since the revelations of widespread mass surveillance in 2013. Since then, many new messenger applications with additional security measures have been developed and old applications have implemented security protocols to protect their user's privacy. In particular the use of *end-to-end encryption (E2EE)* protects the contents of messages.

However, while E2EE secures the contents of the messages it does not completely protect the user's privacy from attacks that target metadata that can be directly connected to a person like IP addresses and phone numbers as well as quantity and time of messages. Especially protocols like VoIP are susceptible to tracking and timing attacks, even if they are encrypted and transmitted over anonymity networks.

Session is an open-source messenger application, that attempts to provide further protection against these and other issues while providing all the common features of popular chat services like group chats, video calls, and multi-device access. This is achieved by using existing protocols, like the Signal protocol, and operating on Service Nodes in a decentralized permissionless Network.

This paper will present the central protocols and techniques that Session is built upon in Section 2, give an overview of the measures in current messenger applications in Section 3, and then explain details of Sessions implementation in the Sections 4 and 5. If not otherwise cited, the Session implementation details are based on the Session white paper found at [1] as well as an informal conversation with Session developers. The economic viability of the Loki Blockchain, that Session is based upon, has been formally analyzed using game theory by Brendan Markey-Towler [2] and further details regarding the staking requirement can be found in the research by Johnathan Ross et al. [3].

2. Background

To develop a secure Messenger with additional privacy protection measures, basic systems to protect the content of messages as well as hide the IP addresses during routing are needed. The protocols, techniques, and network presented in this section have been formally analyzed and tested and can be used as a strong foundation in any secure messaging service.

2.1. Signal Protocol

The Signal Protocol (originally known as TextSecure Protocol) is a non-federated open-source cryptographic protocol for E2EE. It can be used to protect the contents of messages, voice calls as well as video calls while guaranteeing important like properties of deniable authentication, perfect forward secrecy, integrity, destination validity, and others. In particular, the properties perfect forward secrecy and deniable authentication guarantee, that every time a new session is created, a new key is generated for that particular communication session, which means that a single compromised communication session key, does not compromise the content of past and future conversations. Furthermore, deniable authentication, ensures that a sender can authenticate messages for a receiver, such that the receiver cannot prove to a third party that the authentication ever took place.The Signal Protocol uses the well established elliptic curve known as Curve25519 as a basis for its key generation and the key exchange protocol known as Elliptic-curve Diffie-Hellman. These keys are often called X25519 keys. [4]–[6]

Since the code for the Signal Protocol is open-source, it has been formally analyzed and found to be cryptographically sound [6], which has further increased its widespread use in several chat applications (see Table 1).

2.2. Onion routing

Onion routing is a well-reviewed technique for protecting metadata during network communication. The message that is being transmitted is protected by multiple

doi: 10.2313/NET-2020-11-1_05

layers of encryption, akin to the layers of an routing. Every hop removes a layer of encryption and therefore knows only the next hop, while the actual sender and receiver stay anonymous, since every intermediate node only knows the immediately preceding and following nodes. However this technique does not protect against traffic analysis and the exit node can be a single point of failure. The weaknesses of onion routing have been analyzed multiple times, especially regarding TOR (See Section 2.3). [7], [8]

2.3. TOR

TOR, derived from the name of the original project "The onion Router", is a free and open-source server application licensed under the BSD 3-clause license. TOR is an altruistic network, that allows users to protect their privacy through onion routing Protocols running on volunteer-operated servers. Similarly, TOR can circumvent censorship, by providing access to otherwise blocked destinations and content. TOR has been both extensively peer-reviewed as well as analyzed in formal studies, and can, therefore, be a good inspiration or building block for further software. [7]–[9]

3. Overview of popular secure messenger applications

This section presents an overview of popular messenger applications that use different forms of E2EE encryption to provide certain levels of privacy protection.

3.1. WhatsApp

At 1.600 Million monthly active users, as of October 2019 [10], WhatsApp is the most popular Messenger application. It provides users with the expected features in sending messages, creating group-chats, and sharing different forms of media like pictures and videos.

Messages on WhatsApp are encrypted using the Signal Protocol presented in Section 2.1. WhatsApp is closed-source and has not allowed independent code audits, which makes formal analysis and testing difficult, therefore raising privacy concerns.

3.2. Signal

Signal is an open-source messaging service developed by the Signal Foundation and Signal Messenger LLC. Unlike commercial software like WhatsApp, Signal relies on donations and support for its commercial viability.

Signal uses the Signal Protocol as presented in Section 2.1 and provides further privacy protection by encrypting the sender's information in the message, and only storing the login dates, rather than more detailed information. While providing significant protection, Signal still has several vulnerabilities. In particular through the use of a phone number as identifier for the users as well as centralized servers. (see Section 5.2) [11], [12]

TABLE 1: Overview

Name	E2EE	reviewed	'Identification
WhatsApp	Signal Protocol	No	Phone Number
Signal	Signal Protocol	Yes	Phone Number
Telegram	MTProto	partly	Phone Number

reviewed applications have allowed independent code audits and have been formally tested

3.3. Telegram

Telegram, unlike Signal, is only in part open-source, with its client-side code being open to the public, while the server-side code is closed-source. While all server-client communication is in default encrypted, E2EE is only optional for messaging and video chat between users. [13]

Telegram has received notable criticism, for using its untested E2EE algorithm known as MTProto and making its use optional between users, as well as storing critical information like contacts, messages, and media on their centralized Server. [14], [15]

4. Session Basics

This section presents the basic functions and building blocks of the Session messenger.

4.1. Protections

The goal of Session's design is to provide the following protections to its users.

- **Sender Anonymity:** Personal identity of the Sender is only known to the recipients of the messages, while their IP address is only known to the first hop in the onion routing path.
- **Recipient Anonymity:** The IP address of the recipient is only known to the first hop in the onion routing path.
- **Data Integrity:** Session ensures the integrity of the messages both regarding modification and corruption. Messages where this integrity is violated, are discarded.
- **Storage:** For the duration of their *time to live(TTL)*, messages are cached and delivered to clients.
- **E2EE:** Messages maintain the properties of Deniable Authentication and Perfect Forward Secrecy.

4.2. Incentivized Service Nodes

In a centralized Network the central authority is always a single point of failure, which is why, with the rise of Bitcoin, decentralized Networks have become increasingly popular, with projects exploring applications in different fields. In a permissionless network new users can join at any time and provide additional nodes. [16]

However, many of these projects have struggled with similar problems like overloaded servers, as well as security concerns regarding attacks like Sybill attacks. Here an attacker creates multiple anonymous nodes and can, therefore, use traffic analysis to deanonymize users and access private data. [7], [17]

doi: 10.2313/NET-2020-11-1_05

Session aims to prevent these problems by incentivizing good node behavior and creating a financial precondition for their *Loki Service Network*. This network integrates a blockchain and therefore requires anyone wishing to host a server for Session, to go through a staking transaction, during which an operator has to lock an amount of cryptocurrency assigned to the node.(equivalent value of 7.420 USD as of 10/02/2020) [18]

This ensures, that whenever a buyer decides to run a new Service Node, the supply of Loki is decreased, since it is locked during the staking mechanism and therefore removed from the market. This means that the financial resources to acquire enough Service Nodes for a Sybill attack are significant and increase exponentially with the scale of the attack.

Furthermore, the use of a blockchain integrated network allows for a reward system, where whenever a new block is mined, the service node is paid with a part of the block reward. This incentive system has been formally analyzed using game theory by Brendan Markey-Towler. [2] Combined with a consensus-based testing suite, it ensures a high standard of operation and honest node behavior, while being an alternative approach to altruistic networks like TOR or I2P.

4.3. Onion Requests

Service Nodes provide Session with access to a distributed network with a high standard of operation, but to transmit messages while protecting the user's identity from third parties as well as protecting the contents of the messages it still needs encryption and message routing. Session, therefore, uses an onion routing Protocol(See Section 2.2) referred to as *Onion Requests*. The purpose of Onion Requests is to further protect the IP addresses of Session users, by creating randomized three-hop paths through the Service Node Network. For this purpose, every Session client has access to a Service Node list, containing the IP address of each Service Node, as well as the corresponding storage server ports and X25519 keys. This list is fetched on the first launch and then kept up to date through periodic queries on multiple Service Nodes. On application startup, the Session client should use this list to choose three random nodes to establish an onion routed path. After testing whether this path creation was successful, by sending down a request and waiting for a response, the path should persist through multiple requests. In case a response is not achieved, the client will try to create a new path.

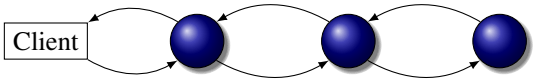

Figure 1: Onion Requests

5. Extending for Messenger app functionality

Through Onion Requests and the Loki Service Network, Session has access to an anonymous network as well as bandwidth and storage space. This section presents central services, that are built on top of the foundation, to allow Session to provide the features expected from a modern chat application.

5.1. Storage

Users of modern chat applications expect message transmission to occur both for synchronous as well as asynchronous communication. To reliably provide this service with an app running on a decentralized network, an additional storage level that provides redundancy is needed, to deal with unexpected operational problems like software bugs. For this purpose, session combines its incentivized Service Nodes with a secondary logical layer, called swarms.

These swarms are created by grouping together Service Nodes and replicating messages across them. To protect from malicious node operation, the initial swarm a node joins is determined by an algorithm that gives minimal influence to the Service Node operator. With nodes joining and leaving the network, the compositions of the swarms have to naturally change during operation:

- **Starving swarm:** In the case, that a swarm needs more nodes, it can "steal" nodes from a different swarm, which has more than its minimal amount of n needed for operation (N_{min})
- **All swarms at N_{min}:** The nodes of the starving swarm will be redistributed among all other swarms
- **Oversaturated swarms:** In the case that multiple nodes will join the network while all swarms are already at maximum capacity (N_{max}), a new swarm will be created from a random selection of N_{target} nodes, where N_{target} is defined as N_{min}>N_{target}>N_{max} to ensure the new swarm is neither under nor oversaturated.

Furthermore, to ensure intended node operation, changes to swarm composition remain synchronized by pushing data records to new members and redistribution of data records by leaving nodes.

5.2. Identification

As can be seen in Table 1, most widespread messenger applications rely on a phone number or email address as an identifier. While this approach has obvious usability advantages like ease of social networking or recovery of login data, it is problematic regarding security and privacy. High-level actors(a person or group, with a lot of resources and access rights) like government institutions or service providers can compromise user accounts tied to a phone number. Furthermore, a high-level actor can in many states directly connect a phone number to more personal information like passport, social security number and more. Even a low-level actor(who has limited resources) can access databases that collect leaked data sets and use them for spoofing attacks like SIM swapping attacks. Therefore, Session instead uses X25519 public-private key pairs for identification and new key pairs can be generated within the application at virtually any time. Upon first launch a user is presented with a generated pair and encouraged to write down their long term private key for potential account recovery in case their device is lost or for other reasons out of service.

doi: 10.2313/NET-2020-11-1_05

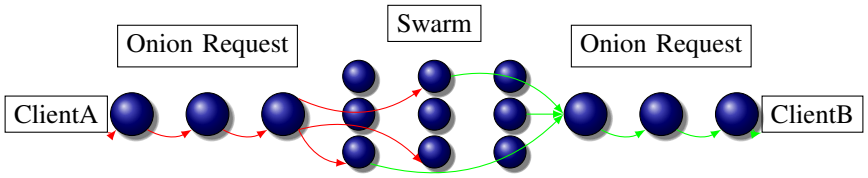

Figure 2: Asynchronous Routing

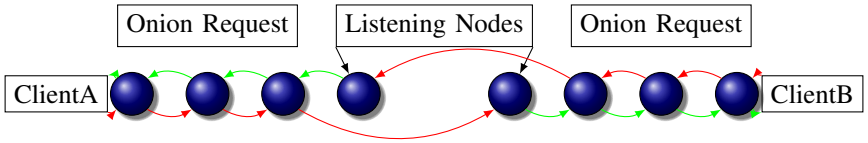

Figure 3: Synchronous Routing

5.3. Message Routing

Through the usage of swarms, as explained in Section 5.1, as well as the public X25519 keys as long-term identification, messages can now be addressed to other Session users, but the actual messages still need to be transmitted. To ensure that the user experience is similar to other modern chat applications, message routing needs to handle both synchronous and asynchronous routing.

5.3.1. Asynchronous Routing. When a user sends a message and either sender or recipient are offline as well as in the default case, Session uses asynchronous routing (Further details on determination in Section 5.3.2) as illustrated in Figure 2. For asynchronous routing the sender‘s client will use the recipients identification key (see section 5.2), to determine the recipients swarm through the deterministic mapping between Service Nodes and long-term-public keys. The sender then serializes the message using the Protocol Buffer method and packs it in a wrapper containing the recipient’s public key, a timestamp the TTL as well as the proof of work. By requiring proof of work, spam attacks become more computationally expensive and, therefore, less practical. Onion Requests create the path towards the recipient’s swarm, where the package is spread to three different Service Nodes inside the swarm. These Service Nodes then spread the message to the other Service Nodes in the swarm. By targeting multiple Service Nodes and spreading the message amongst all members of the swarm, enough redundancy is introduced, to ensure that the message will not be lost for the duration of its TTL and can, therefore, be received whenever the recipient connects to the network.

5.3.2. Synchronous Routing. Asynchronous routing is also used in the default case since the Protocol Buffer of any asynchronous message also contains the online status of the sender as well as a specified Service Node in their swarm, on which they are listening. This means a recipient’s client can then set up a synchronous routing through the specified listening node, by exposing its listening node in the network as illustrated in Figure 3. Once set up the two clients can now simultaneously use Onion Requests to send messages through the network to the conversation partner’s respective listening nodes. Since messages are not propagated through the swarms, there is no redundancy being created and no proof of work is sent. To prevent lost messages, acknowledgments are sent back after every received message. If a message times out, if for example the recipient went offline, the message is resent using the default asynchronous routing.

5.4. Modifications to the Signal Protocol

While onion routing hides the user’s IP addresses via Onion Requests, the actual content of the messages still needs to be encrypted. Session uses the Signal Protocol as described in Section 2.1 as its basis and adds some slight modifications to adapt it to a decentralized network. Additional information is added to each message, to ensure correct routing and verifying correct message creation. Furthermore the sharing of prekey bundles is instead conducted by a "friend request" system. Whenever a user first initiates communication with a new contact, a friend request will be sent. This friend request contains a short written introduction as well as the sender’s prekey bundle and metadata like the sender‘s display name and public key, which as explained in the previous sections can be used to reply to the sender. These friend requests are themselves encrypted for the recipient’s public key using the Elliptic-curve Diffie–Hellman protocol, that the Signal Protocol is based on. In the case that the recipient accepts the friend request, they can then use the prekey bundle to exchange messages, encrypted with the Signal Protocol as desired.

6. Conclusion

Surveillance on the internet is constantly increasing both by individuals as well as states and corporations. In a time where many countries and regimes are working increasingly towards blocking free communication and expression, it has become more important then ever to not just protect the contents of conversations via techniques like E2EE, but also the information if, when, and between whom the conversation took place.

By building on the Signal application with the Signal Protocol and using onion routing like in TOR, Session combines formally tested measures for security and privacy protection, with a decentralized network. The incentive structure designed around the Loki Service Network presents a commercially viable approach to anony-

doi: 10.2313/NET-2020-11-1_05

mous networks and allows for protection against common problems like Sybill attacks and overloaded servers. By extending their systems with an additional logical layer in swarms, Session can use both asynchronous and synchronous routing, to provide the functionality expected from a modern chat application.

Further extensions that allow for the use of multi-device, attachments and group chats, were out of the scope this paper, but are described in the Session white paper as found on their GitHub page and in [1]. Some features like video chats are as of May 2020 still under development.

References

[1] K. Jefferys, M. Shishmarev, and S. Harman, "Session: A Model for End-To-End Encrypted Conversations With Minimal Metadata Leakage," 2020, [Online; accessed 25/03/2020].

[2] B. Markey-Towler, "Cryptoeconomics of the Loki network," 2018, [Online; accessed 25/03/2020].

[3] K. Jefferys, J. Ross, and S. Harman, "Loki Cryptoeconomics: Alterations to the staking requirement and emission curve." 2019, [Online; accessed 25/03/2020].

[4] M. D. Raimondo, R. Gennaro, B. Dowling, L. Garratt, and D. Stebila, "New approaches to deniable authentication." Journal of Cryptology, May 2009.

[5] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, "How secure is textsecure?" in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, March 2016, pp. 457–472.

[6] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," in *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, April 2017, pp. 451–466.

[7] A. Sanatinia and G. Noubir, "Honey Onions: a Framework for Characterizing and Identifying Misbehaving Tor HSDirs," [Online; accessed 22/03/2020].

[8] TorProject.org, "Toor Project: FAQ," [Online; accessed 26/03/2020].

[9] "Tor Project: Overview," https://2019.www.torproject.org/about/overview.html.en, [Online; accessed 18/03/2020].

[10] J. Clement, "Most popular global mobile messenger apps as of October 2019, based on number of monthly active users," https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/, 2019, [Online; accessed 12/03/2020].

[11] Signal.org, "Technology preview: Sealed sender for Signal," https://signal.org/blog/sealed-sender/, 2018, [Online; accessed 15/03/2020].

[12] M. Lee, "Battle of the secure messaging apps: How signal beats whatsapp," *The Intercept*, 2016.

[13] "Telegram F.A.Q," https://telegram.org/faq#q-why-not-open-source-everything, [Online; accessed 22/03/2020].

[14] "why telegrams security flaws may put irans journalists at risk," https://cpj.org/blog/2016/05/why-telegrams-security-flaws-may-put-irans-journal.php, [Online; accessed 22/03/2020].

[15] W. William Turton, "why you should stop using telegram right now," https://gizmodo.com/why-you-should-stop-using-telegram-right-now-1782557415, [Online; accessed 22/03/2020].

[16] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 243–252.

[17] D. McIntyre, "Ethereum Classic Gas System Economics Explained," [Online; accessed 25/03/2020].

[18] Developers, "Loki documentation, loki service node staking requirement," https://docs.loki.network/ServiceNodes/StakingRequirement/, [Online; accessed 17/03/2020].

doi: 10.2313/NET-2020-11-1_05