Unfriendly Chat

Team Wu-Tang LAN

Leslie Zhou (team lead)
Khanh Nguyen
Warren Singh

May 4, 2022

Contents

1 Introduction

In 2013, Edward "Ed" Snowden, a 29 year old government contractor who was a former technical assistant for the CIA and current employee of defense contractor Booz Allen Hamilton came forward with startling revelations: the United States government was indiscriminately collecting and spying on the internet communications of a huge majority of the English speaking world [1]. While security concerns are part of network engineering, major service providers and technology companies did not typically prioritize security at the time. Due to the revelations, securing their services and communications became a top priority, as the scope and extent of the 'bulk collections' programs that were being run by the NSA shocked even industry insiders.

Within six months, prominent companies such as Facebook, Twitter, and Google began implementing upgrades to both internal and external systems [2], and many consider this new approach to be the reason for the quick and widespread adoption of stronger security and end-to-end encryption protocols [3].

But how do technology companies actually secure communications and services for their users? Users will be less likely to use services which do not offer security and privacy, and in general societies are thought to suffer when they cannot protect the privacy of their citizens.

One open source cryptography project [4] is an industry leading standard [5] for end-to-end encryption, developed in the wake of the Snowden revelations: the Signal Protocol. The Signal Protocol [6] is a non-federated cryptographic protocol which is most widely used to ensure end-to-end encryption for communication applications (i.e. text-based messaging and VoIP). Applications which currently implement the Signal Protocol include Google's Messages, Facebook Messenger, Whatsapp, and Skype [5], meaning the number of users whose messages are secured by the Signal Protocol potentially number in the billions (this matches the scope of the problem, as there are billions of users of electronic technologies around the world).

Due to its widespread use, broad influences, intended effect, and open sourced approach, examining the protocol thoroughly is crucial in understanding how industry leaders secure both internal and external network communications, as well as providing a foundation for apprehending and developing further iterations and applications, since developers working on applications continue to use the Signal Protocol as foundation and inspiration for further encryption protocol development.[7][8][9][10] Through the course of this project by which we implement the protocol in a real-time chat application setting, we seek to gain an understanding of this industry standard technology, and transmit that to our colleagues for their benefit as well.

The remainder of this report will details the overview, methods, and results of the implementation of the Signal Protocol. A high level understanding of the protocol is available to the general reader, while others may wish to examine the citations for further exploration.

2 Project Description

2.1 Overview

The actual implementation of the project involves the connecting of a user facing react web chat application with the actual implementation of the cryptographic protocols and algorithms, which are hidden from the user.

A separate server instance is used to store the pre-keys for the initial part of the Signal protocol (which is elaborated on in detail in the Methods section immediately following).

Various so-called cryptographic primitives (which are, in other words, the basic building blocks which make up systems for encryption and security: common examples are one-way hash functions or onion routing/proxy server based 'mix networks') are relied on in the course of the implementation of the actual cryptographic protocols, specifically public/private key pairs for signing from elliptic curve 25519 Diffie-Hellman functions, AES 256 bit encryption for cleartext/ciphertext conversion with respect to the user-generated messages, and HKDF for the key derivation (so-called 'ratcheting') functionality. (more details on these in the methods section which follows)

The authors wish to thank at this time in particular M. Marlinspike and his colleagues at the Signal Foundation[11], as well as R. Schmidt and M. E. Johnson at Privacy Research LLC [12] for their work and generosity. Due to the time constraints of this project, as well as the relatively limited technical expertise and experience of the project team, use of open source libraries and documentation in the project implementation proved necessary under the scope and bounds of the work done. Specific citations follow in the text where appropriate, but in general the materials that these two groups made publically available were very helpful in the research and implementation process. [6][13][14][15][16]

A high-level illustrative flowchart of the project overview is shown on the next page.

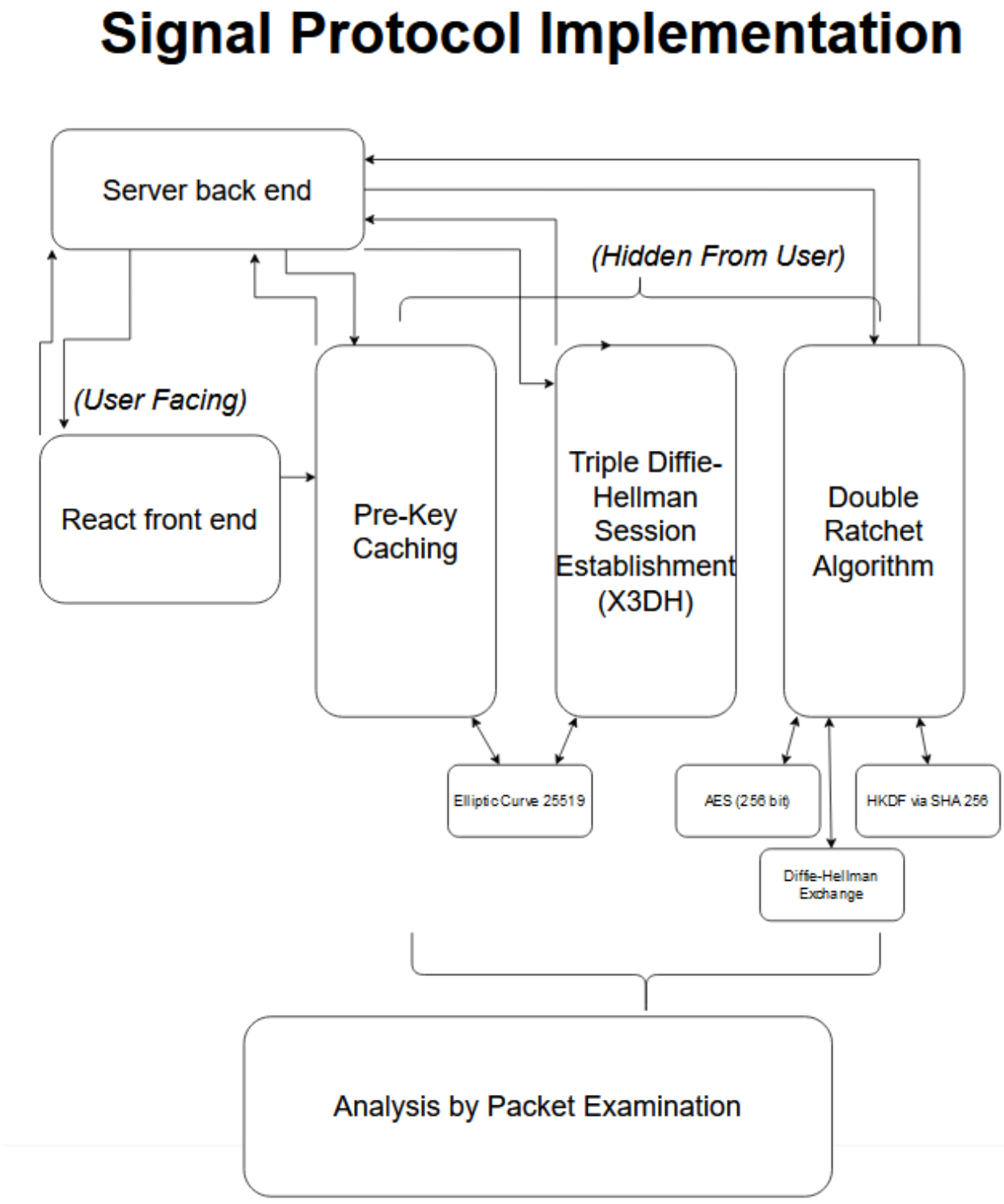

Figure 1: A flowchart illustrating an overview of the overall project design

2.2 Method(s)

2.2.1 React front end

2.2.2 Server back end

2.2.3 Pre-Key Caching

Each user, on registration to the chat service (or, more generally, whatever service is being provided) has a set of keys generated by the application service, some of which are sent to a server for storage and later use. These keys are used in protocol for creation and verification of Edwards-curve Digital Signature Algorithm (EdDSA) compatible digital signatures, as well as for the actual keys sent to the server. [17]

The keys sent to the server form a set of elliptic curve public keys, containing a user identity key, a signed pre-key, a pre-key signature (comprised of a signed identity and signed pre-key), and a set of one-time pre-keys [18] (the actual number of these one-time pre-keys is not defined, but in use is typically more than ten or so, with automatic generation and uploading for 'refilling' to the server when the number runs lower than some developer-defined amount).

The actual implementation of the elliptic curve functions is based on (open-source) C libraries (which is typical for lower-level encryption processes, since lower level languages enable easier access to raw calculations and faster computations), which are then wrapped in higher level languages for access and implementation. [19][20]

Here in this implementation, we note that the specific elliptic curve used is Curve 25519.

2.2.4 Triple Diffie-Hellman exchange

Suppose we have a user Alice who registers with a messaging application which implements the Signal Protocol and wishes to message Bob, another user of the messaging application. In this case, they must establish a shared secret to begin trading messages which each can in turn encrypt and decrypt.

They do this by using a variant of the Diffie-Hellman protocol known as the triple Diffie-Hellman exchange (X3DH). X3DH uses five elliptic curve public keys, which include both Alice and Bob's public identity keys (IK_A, IK_B respectively), as well as one of Bob's pre-keys (OPK_B) and Bob's signed prekey bundle (SPK_B).[18]

Then, the procedure performs (for Alice) a Diffie-Hellman exchange up to four times (but at least three times) in the following way:

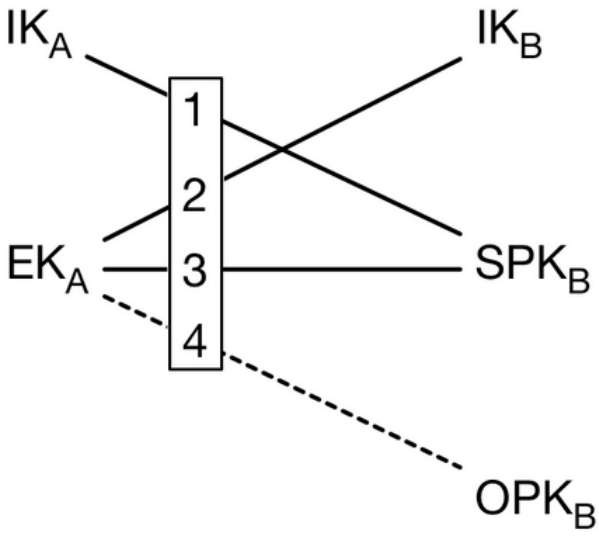

Figure 2: An overview of the Triple Diffie-Hellman exchange[18]

The first two exchanges guarantee authentication of each party (as one might surmise given that they involve each of Alice and Bob's identity public keys), while the latter two provide forward secrecy (including, among other things, a prevention of so-called *replay* attacks).

Assuming that all four exchanges occur, Alice now has four outputs from the Diffie-Hellmans, which are concatenated in order and then used as input to the Key Derivation function for the first secret key (symmetric) for the message encryption and decryption.

Then, Alice sends Bob an initial message containing the necessary information for Bob to replicate the X3DH on his end, such that both parties now have a shared secret key for beginning message exchange, encryption, and decryption.

We pause to note that the Signal Protocol does not guarantee that the public identity key purporting to correspond to the intended recipient is actually under the sole control of the intended recipient: in actual implementation (such as with the Signal Messaging application), verification occurs *out of band* (meaning not involving digital communications mediated by the Signal Protocol). Each user opens a specific window on their piece of application software and can verify the *fingerprint* (the details of which an interested reader may further research as an exercise) of their identity public key, either in a physical meeting, or perhaps in a real-time video teleconference.

2.2.5 Double Ratchet Algorithm

The Double Ratchet Algorithm is most relevant once a communication session is established and preliminary work (the preceding sections) are out of the way. [21] Once a shared secret is established, it is used as the basis for a secret key for message encryption and decryption using AES-256 encryption for the messages, and as input to a Key Derivation Function (KDF). The KDF in the Signal Protocol is most often implemented with SHA-256 or SHA-512 (the hashing algorithms); here it is SHA-256.

Because of the use of hashing, each subsequent key is arrived at via a so-called *one way* function: even if an adversary accessed this key, they would not be able to derive previous keys. This is

the source of *forward* secrecy, and one of the ratchets in the nomenclature of the Double Ratchet Algorithm. (since a mechanical ratchet only turns one way, this is used in general language to indicate the one way function of some operation)

Each time the secret key is used in the KDF, the two important outputs are keys: a message key used for encryption and decryption, and a chain key, used as input to derive the next keys. (there is a third output, which serves as an *initial vector* (iv) for the AES-256 message encryption, but the details of CBC mode AES encryption are outside the scope of the project and this report)

Each participant keeps two chains of keys, where the sending chain of Alice matches the receiving chain of Bob, and vice versa such that they can send and receive encrypted messages.

The astute reader might here find an issue: if an adversary got a hold of a key, they might not be able to go backward and derive *previous* keys in order to read previous messages in the conversation history, but they would be able to use the KDF (since the specifications are open source, there are a limited number of possible KDFs in use for any service which implements the Signal Protocol) to get a hold of any and all *future* keys, thus compromising the integrity of any following communications after the inital key disclosure.

This is where the second ratchet comes into play. Each message contains within its header a new input for a new Diffie-Hellman exchange, and each party then feeds the resulting shared secret into the input for the KDF. Thus the KDF has two inputs: the previous key, and this new shared Diffie-Hellman secret, which provides additional entropy.

Thus, even if a key is obtained by an outside party, only the corresponding message is vulnerable; once the new input to the KDF is processed, the following key is secure since it cannot be obtained from just the previous key.

Therefore, the Double Ratchet Algorithm provides secrecy for the communications it encrypts both forward and backward in sequence, and logically derives its name from this property.

2.2.6 Analysis by Packet Examination

3 Demo/Evaluation

3.1 Experimental Setup

3.2 Results

4 Conclusion and Future Work

The Signal Protocol, apart from being an established industry standard approach for the securing of network communications, is a well studied cryptographic approach for security. [22][23]

This may be credited in largest part to its open source nature: the protocol documentation is open for anyone to read [6], as well as source code being openly available for one of its key implementations [24].

However, the protocol as described here and as implemented in our project has limitations. The protocol may be extended to more than two members, but in application this rapidly becomes unfeasible: sessions must establish X3DH exchanges from every member to every other member, and similarly for the header-based key updating between messages. Such group exchanges are in practice mediated by the Sesame algorithm [25], but is outside the course of our project implementation (and thus this report).

Furthermore, there are limitations to the practical security that the Signal Protocol provides. The protocol does not prohibit an attacker from simply spying on the screen of the device which the user is communicating through (a so-called *shoulder surfing* attack), nor does it prevent a keylogger from being able to read what a user is sending. Any security which the protocol assures (which, as mentioned before, has been audited by reputable, peer-reviewed research [22][23]) is solely on the application software and network levels. It does not extend to device and physical security, either by design or by practice.

Further work in placing secure systems at the hands of users is ongoing, and is a promising area of research and development, especially as the increasing collection and usage of individual data becomes more and more relevant in governmental and industry settings.

5 References

References

[1] G. Greenwald, E. MacAskill, and L. Poitras. *Edward Snowden: the whistleblower behind the NSA surveillance revelations*. June 2013 [Online]. Available. URL: https://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance.

[2] C. C. Miller. *Angry Over U.S. Surveillance, Tech Giants Bolster Defenses*. Nov. 2013 [Online]. Available. URL: https://web.archive.org/web/20131106015230/https://www.nytimes.com/2013/11/01/technology/angry-over-us-surveillance-tech-giants-bolster-defenses.htm.

[3] A. Cuthbertson. *Snowden "Sped Up Encryption" by Seven Years*. June 2016 [Online]. Available. URL: https://www.newsweek.com/snowden-sped-encryption-seven-years-452688.

[4] C. Garling. *Twitter Open Sources Its Android Moxie*. Dec. 2011 [Online]. Available. URL: https://web.archive.org/web/20111222010355/http://www.wired.com/wiredenterprise/2011/12/twitter-open-sources-its-android-moxie/.

[5] J. Lund. *Signal partners with Microsoft to bring end-to-end encryption to Skype*. Jan. 2018 [Online]. Available. URL: https://web.archive.org/web/20200202152037/https://signal.org/blog/skype-partnership/.

[6] M. Marlinspike *et. al. Signal Technical Information, Specifications, and Documentation*. [Online]. Available. URL: https://signal.org/docs/.

[7] O. Eyal. *Canada, Germany and Australia are getting e2e encryption*. May 2016 [Online]. Available. URL: https://web.archive.org/web/20161005083000/http://www.viber.com/en/blog/2016-05-03/canada-germany-and-australia-are-getting-e2e-encryption.

[8] (unavailable). *Viber Encryption Overview*. [Online]. Available. URL: https://web.archive.org/web/20160711035838/http://www.viber.com/en/security-overview.

[9] J. Mayfield. *Forsta developer AMA (interview)*. Apr. 2018 [Online]. Available. URL: https://web.archive.org/web/20180502045526/https://www.reddit.com/r/crypto/comments/8b1m6n/forsta_signal_based_messaging_platform_for/.

[10] J. Mayfield. *Forsta codebase, (Github repository)*. July 2019 [Online]. Available. URL: https://web.archive.org/web/20180613054634/https://github.com/ForstaLabs/libsignal-node.

[11] M.Marlinspike *et. al. Signal Foundation*. [Online]. Available. URL: https://signalfoundation.org/.

[12] R. Schmidt *et. al. Privacy Research LLC*. [Online]. Available. URL: https://privacyresearch.io/.

[13] R. Schmidt *et. al. Privacy Research Group Github repository*. [Online]. Available. URL: https://github.com/privacyresearchgroup.

[14] S. Nonnenberg *et. al. Signal Protocol JavaScript Library*. Nov. 2017 [Online]. Available. URL: https://github.com/signalapp/libsignal-protocol-javascript.

[15] R. Schmidt *et. al. Signal Protocol Typescript Library*. [Online]. Available. URL: https://github.com/privacyresearchgroup/libsignal-protocol-typescript.

[16] R. Schmidt and M. E. Johnson. *Demo React Application using the libsignal-protocol-typescript*. [Online]. Available. URL: https://github.com/privacyresearchgroup/libsignal-typescript-demo.

[17] T. Perrin *et. al. The XEdDSA and VXEdDSA Signature Schemes*. Oct. 2016 [Online]. Available. URL: https://www.signal.org/docs/specifications/xeddsa/.

[18] M.Marlinspike and T. Perrin. *The X3DH Key Agreement Protocol*. Nov. 2016 [Online]. Available. URL: https://www.signal.org/docs/specifications/x3dh/.

[19] R. Schmidt. *Typescript Library for Curve 25519*. [Online]. Available. URL: https://github.com/privacyresearchgroup/curve25519-typescript.

[20] R. Schmidt. *Clean Room Reimplementation of Curve25519*. [Online]. Available. URL: https://github.com/privacyresearchgroup/curve25519-typescript/blob/master/native/curve25519-donna.c.

[21] M.Marlinspike *et. al. The Double Ratchet Algorithm*. [Online]. Available. URL: https://signal.org/docs/specifications/doubleratchet/.

[22] K. Cohn-Gordon et. al. "A Formal Security Analysis of the Signal Messaging Protocol". In: *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*. 2017, pp. 451–466.

[23] N. Kobeissi et. al. "Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach". In: *2nd IEEE European Symposium on Security and Privacy*. Apr. 2017, pp. 435–450.

[24] M.Marlinspike *et. al. Signal: Everywhere and nowhere*. [Online]. Available. URL: https://github.com/signalapp.

[25] M.Marlinspike *et. al. The Sesame Algorithm: Session Management for Asynchronous Message Encryption*. Apr. 2017 [Online]. Available. URL: https://signal.org/docs/specifications/sesame/.