Table of Contents

1. Introduction....1
The application....2
2. Basic concepts....2
Symmetric-key encryption....3
Public-key encryption....3
Authentication....3
Digital signatures....3
The algorithm....4
End-to-end encryption....4
Limitations of end-to-end encryption....4
User metadata....4
Man-in-the-middle attacks....5
Endpoint security....5
Backdoors....5
3. Existing technologies....5
Signal Protocol....5
Extended Triple Diffie-Hellamn....7
Double Ratchet....10
EdDSA signature....11
MTProto....11
Signcryption....12
Letter Sealing....14
Threema....17
Group messaging....18
4. Technologies used....19
5. The application....19
6. Conclusions....19
7. References....19

1. Introduction

As online communication gained more popularity, using instant messaging applications has become a standard in our quotidian lives. Therefore, the need for assurance that there is no third party spying on our conversations, either the government, the service provider or a person with malicious intent, grew even more, especially in states where free speech is threatened.

End-to-end encryption is used to protect the privacy of the messages sent between two or more participants, as they are in transit or at rest, with the intended recipients being the only ones that can decrypt and read the messages. Thus, the third parties interested in intercepting the information sent are unable to see the actual plaintext.

Messaging apps using end-to-end encryption have been around since 2012, with iPhone's native messaging app iMessage[1] and then Signal[4], previously known as TextSecure and RedPhone, developed in 2013. But this practice was popularized by WhatsApp in 2016[2], after they announced that the users' messages will have end-to-end encryption enabled by default for all of their chats from now on, and Facebook's testing of secret chats in the application, in the same year[3].

Both are using the Signal protocol in the background, which came with a few novelties in the field, such as the Double Ratchet Algorithm[5].

Various other apps became more popular in the past years after they received endorsement from different public figures or are used in a more restricted area.

Some of these are implementing their own encryption schemes and protocols, but they are mostly relying on Elliptic Curve cryptography, especially the Curve25519 variant of the Diffie-Hellman key exchange and SHA 256 for hashing ++

A more in depth analysis of the protocols used by some popular applications will be discussed in [Section 3](#3-existing-technologies), as well as their security issues or proposed improvements.

The main cryptographic concepts on which these are based will be briefly presented in [Section 2](#2-basic-concepts), details added accordingly for each protocol extending them.

The application

The app created is a web messaging application which provides end-to-end encryption for both private and group chats. In order to illustrate the risks of using chat applications without end-to-end encryption, the user can switch between the encrypted and non-encrypted versions.

The implementation and the used frameworks and libraries are discussed at large in [Section 4](#4-technologies-used) and [Section 5](#5-the-application).

2. Basic concepts

- definitions and small descriptions of various base concepts that will be used throughout the thesis, more will be added later

Symmetric-key encryption

- HOAC 33

Symmetric-key encryption is an encryption scheme which uses the same key for both encryption and decryption. In this case, the key must be a shared secret between the communicating parties, which might result in security issues if the key is intercepted, if it is sent through an insecure channel.

Public-key encryption

- HOAC 43

Public-key encryption, also known as asymmetric encryption, is an encryption scheme which uses a public and a private key. The public key can be publicly available, while the private key must be kept secret by the user.

To encrypt a message, the sender uses the public key of the receiver, but the messages can be decrypted only by the recipient, using their private key.

This algorithm is usually used for digital signatures.

Authentication

- HOAC 42 intro, 401 ident + ent auth

Authentication is the process of proving the identity of an entity, called claimant, to a verifier, and preventing impersonations. It might be done using certain credentials (a password) or with a digital certificate (in case of websites).

Digital signatures

Digital signatures are equivalent to handwritten signatures and are a means of verifying the authenticity of messages or documents, by providing the entity a way to bind its identity to a piece of information, making it dependent on a secret known by the sender and the content of the message. A valid digital signature is supposed to assure the recipient that the sender is authenticated and that the data was not altered in transit, becoming a way to detect forgery or tampering.

They must be verifiable and they can also provide non-repudiation, which means that the signer cannot successfully claim that they did not sign the message.

The algorithm

Consists of the following:

Key generation: A public and a private key are generated. The private one is kept secret, while the other one is publicly available.

Signing procedure: The signature is produced using the private key of the signer and the message.

Signature verification procedure: From the public key of the sender, the message and the signature, the authenticity of the message can be either accepted or rejected.

Also, the following properties must be satisfied by the signing and verification transformations:

1. A signature of a party on a message is valid if and only if the verification function returns true.

2. It is computationally infeasible for any entity other than A to find a signature for any message from A such that the verification function returns true.

End-to-end encryption

End-to-end encryption is a communication system in which the messages can be read only by those participating in the conversation. The data is encrypted by the sender, at the endpoint, using the public key of the receiver and the only way to decrypt it is by using the recipient's private key. This ensures that the data cannot be read or modified by the service provider or any third party involved, hackers, Gov etc. because they don't have access to the private keys.

The need for this method arises from the fact that many email or messaging applications use third parties to store the data. As it is "in transit", it is encrypted (ex: tls), but when it reaches the server, it can be decrypted and so it is put at risk because the server or a third party is able to read or alter the data before it is send forward to the intended recipient.

Limitations of end-to-end encryption

User metadata

- an important issue regarding the limitations of end-to-end encryption is the fact that matadata is available to the server and it can be read and collected to be used for advertising etc.

- so the server knows to whom did you talk to, at what hour, for how long, from where etc.

- an example would be the update of terms and services of WhatsApp[] at the end of 2020, which resulted in a massive shift of the users to Signal or Telegram

- proposed ways of handling this and collecting as little metadata as possible about the users are going to be later addressed in the next section

Man-in-the-middle attacks

This type of attacks require that the attacker injects themselves between the two endpoints and impersonates one or more of the participating parties. The sender will unknowingly use the public key of the attacker and they are able to decrypt and read or tamper with the messages before sending them forward.

This can be avoided if the identities of the participants are verified.

Endpoint security

The messages are only protected from possible eavesdroppers on the communication channel or while the data is at rest, but the endpoints are still vulnerable. After decryption, the messages in plaintext are available to anyone who has access to the endpoint device, so they can be accessed using other methods (ex. device theft, social engineering, hacking the device).

Backdoors

The application providers might include, intentionally or not, ways to access the data by bypassing the encryption, called backdoors. - surveillance etc.

3. Existing technologies

- about the technologies used in some of the popular end-to-end encrypted apps and a brief description of how they work

Signal Protocol

- [Signal protocol - wiki](https://en.wikipedia.org/wiki/Signal_Protocol)

- [Signal docs](https://signal.org/docs/)

- [The X3DH Key Agreement Protocol](https://signal.org/docs/specifications/x3dh/)

- [A Formal Security Analysis of the Signal Messaging Protocol](https://eprint.iacr.org/2016/1013.pdf)
- notes on how the protocol works

- apps: Signal, Whatsapp, Facebook Messenger, Wire

- combines

  - double ratchet algo

  - prekey bundle

  - x3dh handshake

- uses

  - Curve25519

  - AES-256

- HMAC-SHA256

- [20. Cohn-Gordon2020_Article_AFormalSecurityAnalysisOfTheSi](./PDF/Papers/Signal/20.%20Cohn-Gordon2020_Article_AFormalSecurityAnalysisOfTheSi.pdf) - for security analysis too

- [20. Signal Protocol - Makalah-Kripto-2020-06.pdf](./Pdf/papers/signal/20.%20Signal%20Protocol%20-%20Makalah-Kripto-2020-06.pdf)*

- ratcheting, forward secrecy

- X3DH

- **EXTRA** - OTR was the first security protocol for instant messaging and after each message round trip?, the users established a new ephemeral Diffie-Hellman shared secret => ratcheting because you couldn't decrypt past messages

- **EXTRA** - widespread adoption of secure instant messaging protocols started with iMessage

- 3 stages:

    - initial key exchange with X3DH - long term, mid term and ephemeral DH keys to produce a shared secret root value

    - async ratchet - users alternate in sending the new ephemeral keys with prev generated root keys to generate forward secret chaining keys

    - sym ratchet - use key derivation functions to ratchet forward chaining keys to create sym enc keys

- => each message is encrypted with a new message key

- the ping pong pattern of new epehemaral keys inject auto entropy

- TextSecure - used Double ratche, called Axolotl ratchet at that time => RedPhone => Signal

- over 10 diff types of keys and a chain of updated keys

- async transmission protocol which requires pre-send batches of ephemeral public keys

- when a sender wants to send the messages, they get the keys and performs an AKE like protocol using the long term and ephemeral keys tp compute the message encryption key

- the message keys depend on previous computations of the keys

- registration - users register their identity w/ a key distribution server and upload the long, mid term and eph keys

- session setup - get the public keys of the recv and establish initial enc keys (x3dh)

- sync messaging (asym ratchet updates) - sender exchanges their public keys with the recv and generate a shared secret => start chains of message keys, fresh ephemeral keys

- async messaging (sym ratchet) - a new sym message key is derived from the previous state, if no new message was sent by the recv, keys derived from the previous ephemeral dh public key of the sender

- the following are present there *

- uses X25519 or X448 ECDH

- key derivation functions: HMAC SHA256, HKDF SHA256

- AEAD - encrypt then MAC scheme: AES256 in CBC, PKCS#5 padding, MAC is HMAC sha 256

- xeddsa signature scheme - x25519 or x448

- the rest is highlighted in the other version of the paper

Extended Triple Diffie-Hellamn

- [x3dh](./PDF/Signal/x3dh.pdf)

- key agreement protocol, it establishes a shared secret key between two parties who mutually auth each other based on public keys

- forward secrecy and crypto deniability

- async - offline communication

- 3 phases
    - Bob publshes his id key and prekeys to the server
    - Alice gets the prekey bundle from the server, used to send the first message to Bob
    - Bob recv the message from Alice

Publishing keys

- public keys - elliptic curve public keys (receiver - Bob)
    - id key - uploaded once
    - signed prekey - replaced after a time interval

- prekey signature - replaced after a time interval

- set of one-time prekeys - replaced at undefined times ?

- private key corresp to previous signed prekey can be kept, but should be deleted at some point in order to keep forward secrecy

Initial message

- the prekey bundle (of the recv) should contain:

- id key

- signed prekey

- prekey signature

- one-time prekey (opt)

- after the one-time prekey is sent, it should be deleted

- the signature verification occurs

- if the bundle doesn't contain the one-time prekey, the sender will compute an ephemeral key pair with their public key

- after the SK is computed, the sender deletes the ephemeral private key and the DH outputs

- computes associated data AD (maybe the metadata? since there might be certificates, usernames etc)

- the initial message (from the sender) contains:

- id

- ephemeral key

- ids of the fetched prekeys used

- An initial ciphertext encrypted with some AEAD encryption scheme [4] using AD as associated data and using an encryption key which is either secret key or the output from some cryptographic PRF keyed by secret key

- initial ciphertext has 2 roles:

- first message in post x3dh protocol

- sender's x3dh initial message

- later can be used the same secret key or keys derived from it within the post-x3dh protocol

Recv the initial message

- Bob retrieves Alice's id key and ephemeral key from the message

- using them and the private id key and the private keys corresp to the signed prekey and the one-time prekey, Bob, the recv, obtains the secret key and deletes the DH values

- bob constructs the associated data byte seq with the id key of the sender and his id key and decrypts using the secret key and deletes the one-time prekey private key (forward secrecy)

- if the decryption fails, the secret key is deleted

- like before, the secret key can be used later or derivations of it

Security considerations

- authentication

    - the parties may compare their id public keys (ex QR code scanning, compare pk fingerprints)

    - if the auth is not performed, there is no guarantee that they are who they claim to be (mitm attacks possible then)

- protocol replay

    - if there is no one-time prekey from the sender, the message can be replayed to the recv and it might seem that the sender sent it more times and the same secret key might be derived

    - to avoid

        - in a post-x3dh protocol - new enc key for the sender from a new random input (ex Diffie Hellman based ratcheting protocol)

        - blacklist observed messages

        - replace old signed prekeys earlier

- deniability

    - no proof that that of the contents or that the communication took place

    - [OBS - deniable encryption describes encryption techniques where the existence of an encrypted file or message is deniable in the sense that an adversary cannot prove that the plaintext data exists](https://en.wikipedia.org/wiki/Deniable_encryption)

    - if one of the participants is collaborating with a third party, then proof of their communication can be provided

- signatures

- if there is no signature verification - the server could provide forged prekeys => key compromise

- key compromise

- compromise of private keys => could lead to impersonation or affect the security of secret key values

- mitigation (a kind of) - use of ephemeral keys and prekeys

- some compromise scenarios (pg 9)

- server trust

- malicious server could cause communication failure, refuse to deliver messages

- if the parties are auth, the server might refuse to send the one time prekeys and then the forward secrecy of the secret key depends on the signed prekey's lifetime

- if one party attempts to drain the one-time prekeys of the other party, the server should prevent this (ex: rate limits on fetching the prekey bundles) - affects the forward secrecy

- identity binding

- auth doesn't prevent identity misbinding/ unknown key share attacks

- an attacker could impersonate a party by presenting their prekey bundle as someone else's

- making it harder for the attacker to lie about their identity: add more identifying info in the associated data, hash more identifying info etc.

Double Ratchet

- [double ratchet](./pdf/signal/doubleratchet.pdf)

- after the key exchange, the parties are using the Double ratchet algorithm to send and receive messages

- keys are derived for eveyr double ratchet message

- uses KDF chains

- kdf

- a crypto function that takes a secret random KDF key and input data and the output should be indistiguishable from random

  - the function should still be able to provide a random output, even if the key is known

- kdf chain - when a part of the function output is used as output key, and another part is used to replace the kdf key and used for another input

- properties:

  - resilience

  - forward security

  - break-in recovery

- a double ratche session between two parties has a key for 3 chains:

  - root chain

  - sending chain

  - receiving chain

- on message exchange, the parteis exchange new DH pks and the output secrets become the inputs to the root chain

- output keys from the root chains become new kdf keys for sending and recv chains

- the sending and recv chains advance as each message is sent and recv (symmetric key ratchet)

EdDSA signature

- [xeddsa](./PDF/Signal/xeddsa.pdf)

- enables use of a single key pair for ECDH signatures

- signatures are defined on [twisted Edwards curves](https://en.wikipedia.org/wiki/Twisted_Edwards_curve)

MTProto

- apps: Telegram

- section from the [docs](https://core.telegram.org/api/end-to-end)

- upgraded from MTProto 1.0, vulnerabilities analyzed later

- the MTProto 2.0 uses

    - SHA 256

    - padding

    - the message key depends on the message and a portion of the secret chat key

    - 12 - 1024 padding bytes used

- key generation using Diffie Hellman (the rest of the section is basically the theory of the exchange)

- initiator (sender) obtains the Diffie Hellman parameters (before each key generation, obviously): p prime, high order element g (a generator)

- p should be a 2048 bit prime and if the sender doesn't generate a good random number, the server gets this task etc.

- sender executes request encryption and the receiver gets an update for all associated auth keys (devices), with data about the requestee

- after the receiver accepts the request, the chat is created and the receiver also gets the fresh config params for Diffie Hellman => random 2048 bit number b generated

- after the receiver gets g_a (the key sent from sender), if its length is smaller than 256 bytes, add 0 bytes as padding and the fingerprint has length 64 (for SHA1)

- the fingerprint is used to check for bugs

- key visualization uses the first 128 bits of the SHA1 original key (obtained at chat creation) + 160 bits from the SHA 256 key

- about the same procedure for the sender (?) and if the fingerprint are the same, you can start chatting, otherwise, discard the encryption process (?)

- forward secrecy - clients reinitiate re-keying after 100 msg enc / decr or if it has been in usde for 1 week

- the old keys are discarded

- group messages are not encrypted, encryption is not by default

Signcryption

- apps: iMessage

Apple iMessage

- [official docs (this section)](https://support.apple.com/en-us/HT209110) say that iMessage and FaceTime use end to end encryption, attachements are also encrypted (are uploaded and deleted after 30 days if the message was not sent)

- you can choose automatic deletion of the messages

- info stored:

    - use of services

- messages that can't be delivered - held for 30 days

- metadata about FaceTime calls, 30 days

- contacts, 30 days

Apple security and encryption

- [Encryption and data protection, official (this section)](https://support.apple.com/guide/security/encryption-and-data-protection-overview-sece3bee0835/web)

- [Security overview](https://support.apple.com/fr-fr/guide/security/secd9764312f/web)

- APN - Apple Push Notification service

- IDS - Apple Identity service

- encryption - RSA 1280 bit key, EC 256 bit key on NIST P 256 curve

- signatures - ECSDA

- private keys are saved in the device's keychain (API for passwords, keys and other sensitive credentials)

- public keys are sent to IDS

- methodology - Data Protection

- sender retrieves the public keys and APNs addresses for all associated devices of the receiver

- the message is individually encrypted for each device

- sender generates a random 88 bit value as a HMAC SHA 256 key to construct a value derived from the sender and recv public keys and the plaintext

- 88 + 40 = 128 bit key, which encrypts the message using AES in CTR mode

- 40 bit part used by the recv to verify the integrity of the decrypted message

- AES key is encrypted using RSA-OAEP public keys of the recv device, but iOS 13 or later might use ECIES encryption

- combination of the encr message and encr message key - hashed with SHA 1 and signed with ECSDA, using the private signing key

- recv gets: encryted message text, encrypted message key, sender's digital signature

- metadata (timestamp, APN routing info) is not encrypted

- if message too long, attachement is encrypted using AES in CTR mode with a randomly generated 256 bit key

- process repeated for each participant in a group

Signcryption

- introduced in 1997, along with an elliptic curve signcryption

- public key

- functions of both digital signature and encryption, so you don't digitally sign the message and then encrypt it and in this way you decrease the cost and optimize the procedure

- key generation, signcryption, unsigncryption

- properties:

  - correctness

  - efficiency

  - security, and some signcryption schemes provide public verifiability and foward secrecy, so:

    - confidentiality

    - unforgeability

    - non-repudiation

    - integrity

    - public verifiability

    - forward secrecy and message confidentiality

- iMessage uses signcryption to encrypt the messages

- iMessage uses a scheme that involves symmetric encryption with the key derived from the message, as it is stated in the article

- EMDK (encryption under message derived keys)

- protocol was revised after [CVE 2016 1788](https://nvd.nist.gov/vuln/detail/CVE-2016-1788) was reported (addressed later)

- signcryption aims to provide privacy of the message and authenticity

Letter Sealing

- https://linecorp.com/en/security/encryption/2020h1

- https://help.line.me/line/?contentId=50001520

- https://d.line-scdn.net/stf/linecorp/en/csr/line-encryption-whitepaper-ver2.0.pdf

- apps: Line

- transport protocol: based on SPDY

- handshake protocol based on 0-RTT

- enc - elliptic curve crypto with secp256k1 curve for key exchange

- symmetric encryption - AES + HKDF for key derivation

- static keys - the private key is stored on the server??? and the public keys are embedded in the Line client apps

- key pair for key exchange - ECDH

- key pair for server identity verif - ECSDA

- clients are preinitialized with static ECDH keys => clients can include encrypted app data in the beginning

- handshake protocol, you need to exchange some data first:

- client:

    - generate ephemeral ECDH key + 16 byte client [nonce](https://en.wikipedia.org/wiki/Cryptographic_nonce)

    - derive temp transport key and initialization vector (16bbytes long) using the server's static key and the ephemeral key generated before

    - epehmeral ECDH client handshake generated

    - etc.

- server:

    - temp transport key and initialization vector using server's static ECDH key and client's initial eph key

    - decrypt recv app data and get public key

    - generate eph key pair and nonce

    - derive forward sec transport key and init vect

    - gen sign and handshake state using server's static key

    - encrypt app data

    - send to client: server public key, server nonce, server's static signing key, enc data

- client finish:

    - handshake signature verif

    - if verified, continue with getting the forward secrecy keys

    - enc using the keys

- data is encrypted with 128 bit key using AES GCM [AEAD](https://en.wikipedia.org/wiki/Authenticated_encryption) cipher

- [LINE enc](./pdf/papers/line/20.%20LINE%20Encryption%20Report%20-%20linecorp-com-en-security-encryption-2020h1.pdf) / pg 11

- tls + letter sealing

- text and voice/ video

- e2ee for: if supported?

  - private chats

  - groups

  - location messages? for both above

  - audio calls - private

  - video calls - private

- have their own https api? Line Event delivery GatewaY

- optional in 2015 and by default in 2016, but all the participants in the supported message types must have letter sealing enabled to work

- metadata and other things that are only tls enc:

  - website prev function

  - spam report - the message that was reported

  - media file

  - stickers

  - open chat

  - group calls

  - meeting

  - social plugin

- forward secrecy supported by only a few comm channels

  - supported in case of line server key compromise (client-server)?

  - in case of per device private key compromise

Threema

- Swiss, 2012

- app remote protocol for app and web client data exchange

- uses 2 diff encryption layers

    - end-to-end

    - transport layer

- Elliptic curve based (255 bits)

- ECDH with curve25519

- hash function + random nonce = 256 symmetric key for each message

- to encrypt: XSalsa20

- 128 bit MAC added to detect forgeries

- forward secrecy on the network connection

- doesn't require phone number/ email and it provides a [Threema ID](https://threema.ch/en/faq/threema_id) and contacts access is not mandatory

- personal info is hashed

- contacts or group chats are stored in a decentralized way on the users' devices, not on a server

- messages and media are e2ee

- data stored on servers:

    - messages and group chats until they are sent

    - email addresses and contacts are hashed until the comparizon is done (I guess it's the fact that the other one accepts your request/ you are in their contacts list?)

    - key pairs are locally generated

    - no logs on who talks to whom

- for android - AES 256 based encryption for stored messages, media and the ID's private key

- for ios - ios data protection for local encryption

- ID verification to avoid MITM attacks

- 3 verif lvls
    - typed id and contact not found in the address book by phone nr or email
    - id matched with one of the contacts
    - persoanlly verified id (scanned the QR code)
- [NaCl encryption](https://nacl.cr.yp.to/) box model used to encrypt and auth the messages
- SHA256 of the raw public key => first 16 bits are the fingerprint
- to encrypt and decrypt a message
    - ECDH over Curve25519 hased with the result of HSalsa20 => shared secret
    - random nonce generated
    - XSalsa20 stream cipher with the shared secret and the random nonce to encrypt the plaintext
    - sender uses Poly1305 to compute a MAC and adds it to the ciphertext (prepend) a portion from XSalsa20 is used ot form the MAC key
    - sends MAC, cipertext and nonce to recv
    - decrypt and verify authenticity by reversing the steps

Group messaging

- [2020 - Anonymous Asynchronous Ratchet Tree Protocol for Group Messaging](./PDF/Papers/20.%20Anonymous%20Asynchronous%20Ratchet%20Tree%20Protocol%20for%20-%20sensors-21-01058.pdf)

- [2020 - Challenges in E2E Encrypted Group Messaging](./PDF/Papers/20.%20Challenges%20in%20E2E%20Encrypted%20Group%20Messaging%20-%20GroupMessagingReport.pdf)

4. Technologies used

5. The application

6. Conclusions

7. References

[1]: source for the first time iMessage got the encryption

[2]: [Whatsapp whitepaper](https://scontent.whatsapp.net/v/t39.8562-34/122249142_469857720642275_2152527586907531259_n.pdf/WA_Security_WhitePaper.pdf?

ccb=1-3&_nc_sid=2fbf2a&_nc_ohc=pLKbcESAck8AX95AjA-&_nc_ht=scontent.whatsapp.net&oh=73fbf3d0da3f6cae0b216e22b95cbd8b&oe=6079F899)

[3]: [Messenger Starts Testing End-to-End Encryption with Secret Conversations](https://about.fb.com/news/2016/07/messenger-starts-testing-end-to-end-encryption-with-secret-conversations/)

[4]: source for the history of Signal

[5]: Trevor Perrin - *[The XEdDSA and VXEdDSA Signature Schemes](https://www.signal.org/docs/specifications/xeddsa/xeddsa.pdf)*, 20.10.2016

[6]: Trevor Perrin, Moxie Marlinspike - *[The X3DH Key Agreement Protocol](https://www.signal.org/docs/specifications/x3dh/x3dh.pdf)*, 04.11.2016

[7]: Trevor Perrin, Moxie Marlinspike - *[Double Ratchet Algorithm](https://www.signal.org/docs/specifications/doubleratchet/doubleratchet.pdf)*, 20.11.2016

[8]: Moxie Marlinspike, Trevor Perrin - *[The Sesame Algorithm: Session Management forAsynchronous Message Encryption](https://www.signal.org/docs/specifications/sesame/sesame.pdf)*, 14.04.2017