Table of Contents

Different from last time:

- "basic concepts" section, updates on:
    - symmetric key
    - public key
    - auth
    - AES
    - Diffie Hellman
    - Elliptic curves
- "existing tech" section
    - group messaging
- "application" section
    - features and layout

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. An advantage of this type of ciphers is that they are more efficient in terms of software and hardware.

- One such cryptographic algorithm used by some end-to-end encryption protocols is AES (Advanced Encryption Standard).

Public-key encryption

- HOAC 43

- Public-key encryption, or asymmetric encryption, is an encryption scheme which uses a public and a private key pair for each user. The public key is known and can be publicly distributed, so sending it through an insecure channel is not an issue anymore, but the private key must be kept secret by the user.

- To encrypt a message, the sender uses the public key of the receiver, which can be decrypted only using the recipient's private key.

- The security of this encryption scheme resides on the property of the key pair that while knowing the encryption key, it must be computationally infeasible to obtain the plaintext message from a random ciphertext (obtaining the decryption key).

- Public key encryption is less efficient than symmetric key encryption, so it can be used as a secure channel for key exchange or for encrypting smaller data sets.

- Examples of asymmmetric key algorithms, commonly used in popular end-to-end protocols, are the Diffie-Hellman key echange protocol and Elliptic curve based cryptography.

Attacks

Impersonation

- There still remains room for an impersonation attack. This means that an adversary can place themselves in the communication between two parties, A and B, and send their public key such that A thinks it was B's public key.

- In this way, the adversary can decrypt the message, read and/ or alter it before encrypting it with B's key and sending it forward.

- This kind of attack can be mitigated using authentication, so guaranteeing that the recipient is the intended one.

Chosen plaintext

- [wiki](https://en.wikipedia.org/wiki/Chosen-plaintext_attack)

- [wiki, semantic security](https://en.wikipedia.org/wiki/Semantic_security)

- The adversary chooses arbitrary plaintext and then is given the corresponding ciphertext and the intention is to reduce the security of the encryption scheme.

- There are two forms, of this attack:

  - batch attack - when the adversary knows the plaintext before seeing the ciphertext

  - adatptive - when the attacker can request other ciphertexts after seeing some ciphertexts of corresponding plaintext

- This vulnerability can be fixed by providing semantic security, meaning that the adversary should not be able to derive anything but negligible information about a plaintext message, given the ciphertext and the public key. This property is also called indistiguishability under chosen plaintext attack.

Chosen ciphertext

- [wiki](https://en.wikipedia.org/wiki/Chosen-ciphertext_attack)

- This type of attack consists of an adversay that has access to decryptions of chosen ciphertexts and the intention is to obtain the private key.

- This type of attacks can be split in two categories:

  - indifferent or lunchtime attacks - the attacker receives the decryptions of any chosen ciphertext. These must be chosen before receiving the target ciphertext ???

  - adaptive attacks - the attacker has access to the victim's **decryption machine**, and may request decryptions of related ciphertext, but not the target ciphertext, based on the previously received plaintext

- To avoid such attacks, the cryptosystem should not provide any decryption oracles, for example.

- This algorithm is usually applied in digital signatures schemes

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).

- Data origin authentication or message authentication techniques assures one party of the identity of the sender.

- Usually, the message has additional information attached so the receiver can determine it.

- A difference between entity authentication and message authentication is that the latter does not provide timeliness guarantees regarding when the message was created, while entity authentication does the verification in real-time, during the execution of the verification protocol.

Message authentication codes

- some other info from Serious crypto pg 179

- contemporary crypto/ 318

- [MLS](PDF/Papers/20. Evaluation of the MessagingLayer Security Protocol - FULLTEXT01.pdf)

- hash function - function that transforms data of an arbitrary size to a fixed size hash value, and it can be used for data verification, signatures etc.

- it should not produce collisions, that is having the same hash for two different sets of data

- keyed hashing functions (hashing func with secret keys) => message auth codes (MAC) and pseudorandom func (PRF)

- macs protect the identity and auth by creating a value (authentication tag) from the message and the key

- authentication tags, computed and verified with a secret key. They depend on the authenticated message and the secret key, which is known to the communicating parties

- if you know the macs key, you can confirm that a message was not modified in transit => integrity and auth

- often combined with a cipher => preserving message's confidentiality, integrity, auth

- ex: ssh, tls, ip security generate a mac for each packet sent

- forgery - create a tag when you don't know the key

- attack vectors:

  - known-message attack - tags and data collected by an eavesdropper

  - chosen message attack - the attacker chooses the messages to be auth and if the attacker is able to adaptively choose other messages and their corresponding MACs, it is an adaptive chosen-message attack

- replay attacks - capture a message and resent it to the receiver, pretending to be the sender - mitigation by numbering the messages?

Pseudorandom functions

- pseudorandom functions sc/ 181
- contemporary crypto/ 327
- "should be indistiguishable from a random function"
- takes in a message and e key and the output should seem random => unpredictable values
- called xor macs
- not meant to be used on their own
- key derivation schemes use this to generate crypto keys from a master key or password
- ident keys use this to generate a response from a random challenge? - ex a server sends a random challenge message and the recv should prove with this that it knows the key
- tls - prf to generate key material from a master secret and session specific random values
- stronger than macs and any secure prf is also a secure mac

Hash based message authentication codes

- sc pg 184
- hash based mac
- build a mac from a hash function
- produces a secure prf if the underlying hash is collision resistant or if the hash's compression function is a prf

- contemporary crypto/ 323
- authenticity and integrity
- more efficient when you want to auth the messages

Authenticated encryption

- sc 200
- [wiki](https://en.wikipedia.org/wiki/Authenticated_encryption)
- contemporary crypto/ 451 - entity encryption
- AE or AEAD (auth enc with assoc data) - assure confidentiality and auth
- produce the tag and encrypt the data => combination of cipher and mac
- about AES GCM also
- combinations:

  - encrypt and mac - ciphertext and tag are generated from the plaintext
  - mac then encrypt
  - encrypt then mac
- auth ciphers - alternative to the cipher and mac combinations => like normal ciphers but return an auth tag with the ciphertext

Authenticated encryption with associated data

- data processed by an auth cipher, but not encrypted => data is auth but in plaintext
- ex: if you want to send a header and a payload, you enc the payload, but keep the header (assoc data) unencrypted so it can be processed, but you still want to auth it
- takes in the key, plaintext and the assoc data and returns the ciphertext, unenc assoc data, the auth tag
- if assoc data is empty => normal auth cipher
- if plaintext is empty => mac
- you should avoid predictability of enc schemes using a nonce
- more on security and AES GCM from 206

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 and allowing them to securely communicate through an unsecured channel.

- 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 because they don't have access to the private keys.

- The need for this method arises from the fact that many email and messaging applications use third parties to store the data and it is protected only "in transit" (ex: TLS), but when it reaches the server, it can be decrypted and read and/ or tampered with before redirecting it to the recipient

- Thus, the privacy of data and the user is put at risk, since the contents can be used and interpreted by anyone with access to the server

Limitations of end-to-end encryption

User metadata

- an important drawback 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](https://www.whatsapp.com/legal/updates/privacy-policy/?lang=en), now owned by Facebook, at the end of 2020, which resulted in a massive shift of the users to Signal or Telegram (some blog post here)

- information that is collected includes, besides hardware and model information, browser information, mobile network, connection info, location information (for location related features)

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

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 participants' identities are verified and some applications provide authentication via QR code scanning or using safety numbers.

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.

AES

- contemporary crypto/ 282

- NIST paper

- It is a symmetric block cipher, based on a substitution-permutation network, which uses keys of length 128, 192 or 256 bits to process data in blocks of 128 bits, introduced by NIST in 2001.

- the operations are performed on a state array, which is a two-dimensional array of bytes, having the block length div by 32, Nb (Nb = 128 / 32 = 4) rows and columns and the cells contain 1 byte of the block

- the key length is computed in the same way, so it would have length Nk 4, 6, 8, depending on number of columns in the cipher key (from 128, 192, 256 bits)

- the state array can be interpreted as a state array of 32 bit words

- operations over a finite field

  - addition - xor

  - multiplication - multiplication of the polynomials mod irreducible polynomial x^8 + x^4 + x^3 + x + 1

- round keys are values derived from the cipher key and are applied to the state

- these values are fixed: Nr = 10, Nk = 4; Nr = 12, Nk = 6; Nr = 14, Nk = 8

- the round function is composed of:

  - byte subst using a subst table (S-box)

  - shift the rows of the state by different offsets

  - mix data in each column

  - add round key to state

- after an initial round key addition, the round function is implemented Nr - 1 times

- the inverse cipher (decryption) is following the previous steps but in reverse order

[aes]: NIST, [Announcing the ADVANCED ENCRYPTION STANDARD (AES)](https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf)

Classical Diffie Hellamn

- serious crypto/ 268

- Diffie Hellamn (1976) is a protocol that allows the participants to share a secret between them, with the exchanged information being public

- the secret could be turned into a secure channel for transmitting symmetric keys, for ex

- the mathematical function involves a big prime number p and a base number/ generator g (public part) and a number a from the Zp* set, chosen by each participant (private part)

- for example, for two participants, we have the numbers a and b

- then each of the participants computes A = g^a mod p, B = g^b mod p and makes these computations publicly available

- the other participant takes this result and raises it to their private number and this will be the shared secret, so: (g^a mod p)^b = (g^b mod p)^a = g^ab mod p

- even if this computation seems easy, its security resides on the discrete logarithmic problem, which means that you need to recover a from g^a mod p

- this algorithm is designed for secure key agreement protocols => a shared secret between two or more parties, which is turned into session key(s)

- the session keys are symmetric keys used to encrypt and authenticate data during the duration of the session

- attack models, as named and defined in the book at / 275

- the eavesdropper - attacker observers the message exchange and can record, modify, drop, inject messages; to protect against, the protocol should not leak info

  - the data leak - the attacker has access to some of the session keys and temp secrets (from one or more exec of the protocol), but not the long term secrets

  - the breach - the attacker knows the long term key of one or more parties

- some security goals

  - authentication - auth key agreement - both parties auth

  - key control

  - *forward secrecy* - if the session is compromised, the keys used in the past cannot be compromised

  - resistance to key-compromise impersonation - happens when the long-term key is compromised

Elliptic curve cryptography

- serious crypto/ pg 288 and the presentation saved somewhere

- they are curves which are also groups, so they keep the group axioms

- they can also be defined over finite fields and the law is constructed geometrically, the points on the curve with equation y^2 = x^3 + ax + b (Weierstrass curve) - but can have differrent forms

- the operations are addition and multiplication ++

- security relies on elliptic curve discrete logarithm problem ++

- combined with DH => DH key agreement over elliptic curves

  - digital signatures (ECDSA)

  - encryption

- NIST curves: standardized curves and one that is throughoughly used is Curve25519 and has the equation y^2 = x^3 + 486662x^2 + x, which works with numbers mod 2^{255} - 19 (256 bit prime number)

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
- [Security audit](./PDF/Papers/Threema/20.%20security_audit_report_threema_2020.pdf)
- security audit in oct 2020, conducted by cure53
- shows no high security vulnerabilities, only minor or general flaws

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)

- info is available in [this blog post](./pdf/Papers/Groups/signal-org-blog-private-groups-.pdf) too

- some of the previously mentioned apps are implementing the same ee2e protocol for one-to-one ahts, but, when talking about the group conversations, the approapches are different, or they don't support this feature at all

- Telegram, for example, doesn't have ee2e for group chats, for security reasons (here we will have some stories about this. I saw this in the whitepaper or another paper)

- so it relies only on the transport layer security, as well as Facebook messenger, the content of the messages being available to the server

- if you want to have e2ee in group conversations, you may want to keep forward secrecy, post-compromise security, deniability (if possible)

- bigger groups => gets harder to manage the keys (it is hard in the first place, since you need to do certain things for each participant)

- there are 3 ways in which you can handle this:

  - pair-wise - the sender takes the secret key of each of the receiver, encrypts the message and sends it forward to the intended recipient (behaves like normal one to one chats); all the properties of the simple chats are preserved and the groups could be practically invisible for the server

  - encrypted message keys - the sender should choose a new random key for each message with which they would encrypt the message and send only a copy to the recipients. The encrypted message keys are then send out to each of the recipients, in order to decrypt the message; the properties are still kept, but the server is aware of the fact that there is a group

  - shared group-keys - a group key-exchange should be made (both static and ephemeral public keys from each member); while this would significantly lower the complexity of sending messages, the key exchange complexity could grow, especially when you want to keep forward secrecy and post-compromise security, as the keys must be changed regularly; the server, again, is aware of the group structure

- metadata could be collected, again

- one way to obstruct metadada collection by third parties is to use TLS to secure the comm between the user and client

- the server still needs to know where to send a message

- inside the app:

  - if you can use only email or an username, yo could gain anonomity, but the app would still be aware of your social graph (you will need to find and communicate with the other users anyway)

  - if you have contacts list - use a hash, but this is still not enough, because someone can obtain the tuples of username and hash

  - **SGX**

    - [wiki](https://en.wikipedia.org/wiki/Software_Guard_Extensions)

    - [some more on this](https://www.blackhat.com/docs/us-16/materials/us-16-Aumasson-SGX-Secure-Enclaves-In-Practice-Security-And-Crypto-Review.pdf)

    - intel

    - set of security related instruction codes that are built into some intel cpus

    - have private regions (enclaves) whose contents are protected and unable to be read or saved by any process (including those with higher priv) outside the enclave

    - encryption of some parts of the memory, actually

App comparison

- signal

- double ratchet procedure for authenticated key-exchange: long term + short term keys from each participant => shared key

- key used in a key deriv ratchet protocol => ephemeral session keys

- forward secrecy and post-compromise security achieved

- ake takes long term enc keys => deniability

- initiator sends, in a pair-wise fashion, a fresh keys to each participant

- group session key is derived from the shared group key, using double ratchet and the group keys are updated on each newly added participant

- online-offline key exchange - static keys are given by the server, but the ephemeral key exchanges are dealt with by having all users frequently upload one-time prekeys to the server, so the inline party can perform the key exchange

- safety numbers or qr code for auth

- SGX - metadata is stored encrypted on the server

- wapp

- similar, using pair-wise channels

- but the server has access to the medatada

- imessage
- public/ private key pair for encryption and signature, and the public keys are saved on the server
- only static public keys are used => no forward sececry, post-compromise security
- pair-wise and ineffiecient, but groups are not used that much, since it was initially created to replace sms
- no way to auth the users

- wire
- double ratchet for session keys for pair-wise comm and channels for group conversations
- auth with safety numbers

- threema
- public keys published to the server and these are used for encr and sign => no forward sececry, post-compromise security
- shared secret keys ofer deniability
- group communication with pair-wise channes, but multimedia is encrypted using the message key method
- may offer anonimity - username, phone number or email to create an account
- auth with safety numbers

- [2020 - Challenges in E2E Encrypted Group Messaging](./PDF/Papers/20.%20Challenges%20in%20E2E%20Encrypted%20Group%20Messaging%20-%20GroupMessagingReport.pdf)
- gives definitions about group anonimity features, such as internal group anonimity and external group anonimity and propose the Anonymous Async Ratchet Tree protocol
- the properties of forward secrecy and post-compromise security should be kept

About MLS

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

- messaging layer security - initiative from The Internet Engineering Task Force - protocol for group messaging with the best security properties and and more efficient key exchanges (log communication cost, using bin trees)

- [MLS](PDF/Papers/20. Evaluation of the MessagingLayer Security Protocol - FULLTEXT01.pdf)

- [draft?](https://github.com/mlswg/mls-protocol/blob/master/draft-ietf-mls-protocol.md)

- [all drafts](https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/)

Async ratcheting trees

- allow async communication where none of the parties need to be online at the same time

- keeps backward secrecy

- the idea is to generate a group key, encrypt the message once and send it to the group participants (maybe in a server-side fan-out fashion)

- a binary tree is created, where the member device is a leaf node => O(log(N)), N nr of participants

- the parent nodes are generated using Diffie Hellman operations between the teo children and the intermediate nodes are subgroups => each device might be a member of log(N) subgroups

- the ones in the subgroups can use the key pair to encrypt and decrypt the messages

- when a user changes their key or a new user is added, then a new leaf is added to the tree and the updates go up the tree => the root key is changed on each update

- this means that all the nodes know the private key of the parent, but not of the other nodes

- a node is removed => the path from the root to the removed leaf is blanked and a new shared group secret is computed

TreeKEM

- developed based on ART ideas

- users arranged in left-balanced trees

- the parent key is computed by hashing the key of the last modified node

- the nodes know the private key of the parent node, then the updater needs to encrypt O(log(N)) messages and each node receives a mesage with a secret

++ add and remove operations

MLS

- WIP but would support 2 - 50.000 users

- initially based on ART, but the current versions (as of 2020) are based on TreeKEM

- a few initial implementations available (pg 11)

- hash function for parent nodes replaced with a key deriv function

- the messages are encrypted once and broadcasted as the group operations

- it is assumed that participant has the public keys of all the nodes in the tree, and the private keys for those in the subgroup/ copath (siblings to each node in the path)

- tree verification by recursive hashing

  - hash value of each node is based on the info about the current node (leaves) or the hashes of the children (non-leaves)

  - running transcript hash value - created using the group ops leading to the current state - each step is a combination of the prev transcript hash function and the current op

- keys and nonces are updated every time the state changes, using a number of key schedules

- the server should provide

  - authentication service - connect identity to one or more keys (long term identifier, key that can be used to auth protocol messages)

  - delivery service - delivers messages in an async fashion and they are stored until the recipient becomes available; the clients publish a set of initial keys/ keying material, which is used only once; the users can be auth using the auth service

Group operations

- 4 types of operations can be performed: welcome, add, remove, update (these are functions)

- group addition

- users publish the initialization keys to the delivery server and other clients can request these keys to create a group

- the requestee gets the initialization keys of the chosen participants and create a new group state

- then they will send the a welcome and then an add message to all of these participants consecutively, updating the group state after each addition

- if the a new participant is added, the process is repeated and the add message is broadcast to the rest of the group and the new member will perform an update after they were invited (recommended)

- updating

- changes the participant's leaf secret and the direct path from the leaf => bacward secrecy on the participant's leaf secret

- updates depend on the application

- deleting

- a member sends the removal request with the index in the tree and the direct path from the leaf to the root is blanked and the tree is truncated at the rightmost blank leaf

Initial implementations

- MLS++ Cisco - https://github.com/cisco/mlspp

- Molasses, Trail of bits - https://github.com/trailofbits/molasses

- Melissa, Wire - https://github.com/wireapp/melissa

4. Technologies used

5. The application

The functionalities provided by the application are as follows:

- Login/ Signup - the user can login to an existing account or create a new account with the email and a password, upon opening the application, and is redirected to the main application window.

- Logout - the user is send back to the login page.

- Display chats - the conversations of the logged user are displayed automatically, if any.

- Select chat and send messages and attachments - once a chat is selected, the user can send messages to the recipient(s) and also see the previous messages sent and received

- Add chat - a private or a group chat can be added. Here, the user can select whether they want it encrypted or not

- Delete chat - each chat will have a delete button, which erases it from the database and it won't appear in the user's list anymore, but will still be visible for the recipient. In case of a group, a message will be displayed that the user left the chat

- Delete message - each message will have a delete button and will be removed from the user's message list. It will be still visible for the recipient.

Layout

The user is greeted with a login page. From this, they can create a new account or login, using an email and a password.

On the left side, the list of conversations for that account (it is empty if a new account was created) will appear, as well as buttons to add a new private or group chat. Clicking one of the buttons will open a form over the chat list and, for the private chat, will ask for the email of the recipient and to choose if you want the chat unencrypted (encryption is selected by default) and for the group, there will be an additional input for the chat name and an button saying "add another participant".

The right side contains settings, the email of the user, information about the selected conversation etc. and the logout button. The logout button will redirect the user to the login page.

After choosing one of the listed chats (or after adding one), the user is prompted with a chat box and can send messages to the selected recipient. If the user wants to add an attachment, a the file explorer will open and the user will be able to choose the desired file.

Both the conversations and the messages can be deleted by clicking on the "x" icon next to each and then confirming the removal from the dialog box.

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