Understanding a real-world Protocol

Homework 5

Computer and Network Security course A.A. 2020/2021

30/11/2020

1 About

This document is about the report of the Homework 5 for the Computer and Network Security course at Sapienza University of Rome (A.A. 2020/2021).

The goal of the Homework consists in the understanding of real-world protocols for secure communication used by two famous messaging application: WhatsApp and Telegram.
Several points of reflection were given and I chose the assignment number 2, *"A comparison of the two protocols: high level discussion of the two protocols and then a detailed discussion of interesting differences and similarities"*.

The discussion about differences and similarities is done step by step, trying to present to the reader a gradual comparison between the two applications and the respective protocols where it is possible.

2 Application Overview

The real-world protocols that I am going to describe later in details are **Signal Protocol**, adopted by WhatsApp, and **MTProto**, developed by Telegram.

Before to enter inside the technical aspects it is good to present which are the domains of interest of the two applications with the purpose to construct a more general reasoning around some architectural and technical choices: WhatsApp and Telegram are two of the most used messaging application in the world right now, and with the growth of the users basin some methods and techniques to guarantee a secure communications evolve, some other gets modified or entirely changed.

End-to-End Encryption (E2EE) A little eye is due on the End-to-End Encryption: this is a communication system where only the involved users can read the messages, preventing potential eavesdroppers from having access to the cryptographic keys needed to decrypt the conversations, including the internet providers and even the provider of the service itself.

2.1 WhatsApp Overview

WhatsApp is a cross-platform American free application for exchanging messages, audio and video calls and other media founded in 2009. The service is owned by Facebook, Inc. WhatsApp client runs on mobile devices, but it is also accessible from desktop computer by opening a session on a web browser or on a desktop application.

To use the service is required a cellular mobile number: the software automatically compares all the phone numbers from the device's address book with its **central database of WhatsApp users** to automatically add contacts to the user's contact list.

Each WhatsApp account has a random generated password on the server side to be recognized that makes the account **unique for each phone number**. [1]

WhatsApp uses a "**Store and Forward**" technique for exchanging messages between two users: after a message has been sent by an users, it travels to the WhatsApp server where it is stored. Then the server repeatedly requests the receiver to acknowledge receipt of the message. As soon as the message is acknowledged, the server drops the message; it is no longer available in the database of the server. The WhatsApp server keeps the message only for 30 days in its database when it is not delivered (when the receiver is not active on WhatsApp for 30 days). [2] [3]

Since the period between 2014 and 2016 WhatsApp uses **End-to-End Encryption** to exchange messages, provided and performed by the Signal Protocol. Before this time, WhatsApp used to send messages in plaintext.

2.2 Telegram Overview

Telegram is a cross-platform c**loud-based** instant messaging, video calling and VoIP service released in 2013. Telegram client apps are available for mobile devices and through a browser web access. [4]

Telegram **accounts** are associated with mobile phone numbers and they are verified by SMS. Users can add **multiple devices** to their account and manage messages and settings on all of them.

The default method of authentication used by Telegram is a SMS based **single-factor authentication**: by receiving a passcode with an SMS, the user gains access to his cloud-based messages. This one-factor authentication has been intercepted, proving that a one-factor authentication can not be defined as secure. [5] [6]

Telegram's standard messages are **cloud-based**, so that they can be accessed on any of the user's devices. All messages are sent to Telegram Messenger LLP's servers, including photos, videos, audios, files et cetera.

Normal messages and media use client-server encryption during transit, with Telegram's Protocol **MTProto**. According to Privacy Policy, *"All data is stored heavily encrypted and the encryption keys in each case are stored in several other data centers in different jurisdictions. This way local engineers or physical intruders cannot get access to user data."*. [7]
It is however to say that this data is also encrypted at rest, but can be accessed by Telegram **developers, who hold the encryption keys**.

End-to-End Encryption is provided by Telegram only for encrypted calls and for **secret chats**, that are special chats that use a different implementation of MTProto, studied to avoid persistency of the content of this kind of chat.

2.3 Comparison

As seen, the two applications are quite different but have some contact points: on one side WhatsApp adopts a store and forward technique, getting rid of user data as soon as possible, and on the other side Telegram collects everything and, even if encrypted, those data are accessible from some people who owns the keys.
Telegram's cloud-based system can be imagined like an email system, with data sparse in all parts of the world, while WhatsApp servers are located in

Facebook Datacenters.
Another aspect to consider when comparing only the surface of the security trust to give a service is that in WhatsApp privacy policy is well specified that the application automatically records a large number of various information [8], so even if no sensitive information is collected, a profilation of the user is done.
Telegram says that the no commercial use of the collected data is done, and the only information collected is needed by the application "*to function as a secure and feature-rich cloud service*".
In both the Privacy Policy details is written that the service will assist "government enforcement agencies" or "Law Enforcement Authorities" in case of needs.

3 Communication Protocol

As specified above, the two communication protocols I am talking about are **Signal Protocol** and **MTProto**, respectively adopted by WhatsApp and Telegram: it is possible to deduce that some choiches described in the previous sections about many fields, from the application architecture to the service design, are somehow related to the communication protocols and their functioning.

3.1 Signal Protocol (WhatsApp)

Signal Protocol is a protocol that can be used to provide end-to-end encryption for voice calls, video calls and instant messaging conversations.

The protocol combines the Double Ratchet Algorithm, prekeys, and a triple Elliptic-Curve Diffie-Hellman (3-DH) handshake, and uses Curve25519, AES-256 and HMAC-SHA256 as low level cryptographic algorithms (primitives). [9] [10] [1]

The protocol provides confidentiality, integrity, authentication, participant consistency, destination validation, forward secrecy, future secrecy, causality preservation, message unlinkability, message repudiation, participation repudiation, and asynchronicity. It does not provide anonymity preservation.

[1]It is possible to refer at TextSecure application since this has been the first application using Signal Protocol

Signal Protocol lets the exchanging of messages from one party to another, between an initiator Alice, a receiver and eventually responder Bob. To do so, it requires servers (also known as **Key Distribution Servers**) for the relaying of messages and storing of public key material: the servers only store and relay information between the communicating sides and does not compute anything.

Signal assumes that each party has a public-private key pair denoted as **identity key**. However, since the parties might be offline at any moment of the messages exchange, standard authentiated key-exchange(AKE) solutions cannot be directly applied: if for instance one of the two sides of communication goes offline during a key exchange it is impossible for the protocol to maintain all its security properties.

To make a secure messaging exchange possible, Signal implements an **asynchronous transmission protocol** by requiring potential recipients to pre-send batches of ephemeral public keys during registration or later.
When the sender (Alice) wants to send a message she asks the server for the receiver's (Bob) keys and encrypts the message with the long-term and the ephemeral keys. This basic setup is then extended by making the message keys dependent on all previously performed exchanges between the parties, using a combination of "ratcheting" mechanisms to form "chains", with the adding of new random and secret values computed by the communicators. [11]

The protocol consists of several steps for the authenticated key exchange and the messaging.

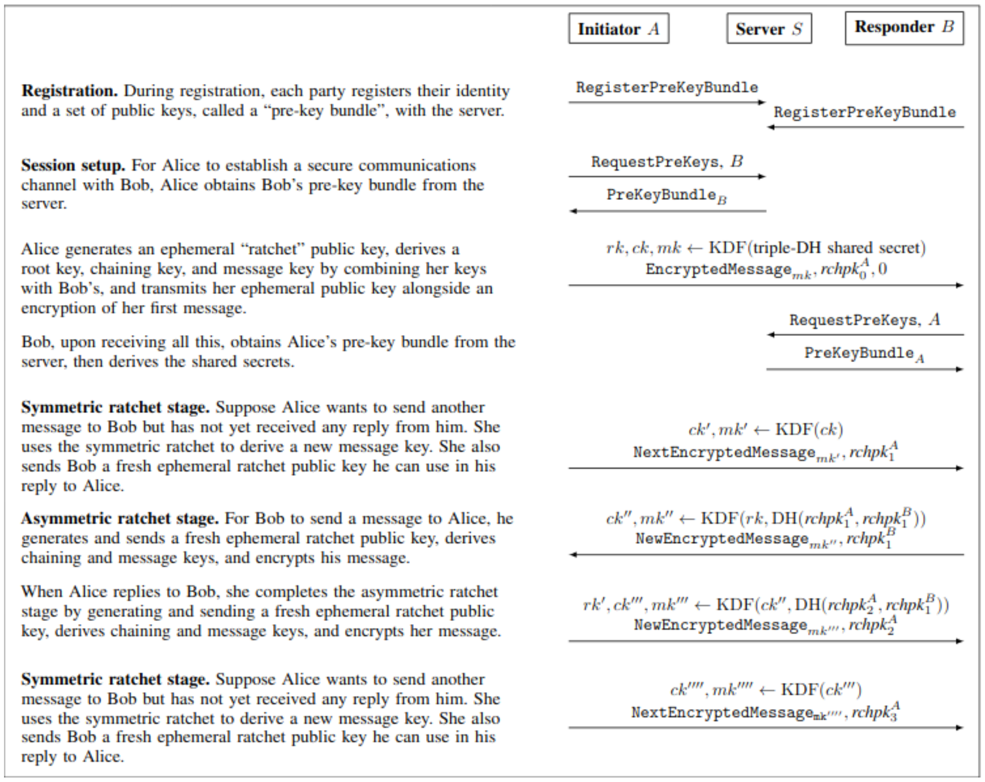
At **registration** time each user registers indipendently its own identity with the server and also sends various time range term public keys.

To **setup a session** Alice queries the key distribution server for Bob's public pre-keys and starts a long-term messaging session by establishing the initial encryption keys. This phase is called **TripleDH Handshake** or **X3DH**.

For **synchronous messaging** step, i.e. when Alice has just received a message from Bob, she exchanges Diffie-Hellman values with Bob, generating new shared secrets and using them to begin new chains of message keys. Each DH operation is a stage of the "**asymmetric ratched**".
Basically what happens is that for each message exchange, in form of a sort of ping-pong from Alice and Bob, a new ephemeral public key is generated and sent to the other user, and so does the recipient: this continuous exchange

creates a true chain of ephemeral keys used to encrypt and decrypt the messages.

In the **asynchronous messaging** moment, so it is called when Alice wants to contact Bob but has not received a message from Bob since her last sent message, Alice derives a new symmetric encryption key using a pseudo-random function (PRF). Each PRF application is a stage of the "**symmetric ratchet**".



Registration. During registration, each party registers their identity and a set of public keys, called a "pre-key bundle", with the server.

Session setup. For Alice to establish a secure communications channel with Bob, Alice obtains Bob's pre-key bundle from the server.

Alice generates an ephemeral "ratchet" public key, derives a root key, chaining key, and message key by combining her keys with Bob's, and transmits her ephemeral public key alongside an encryption of her first message.

Bob, upon receiving all this, obtains Alice's pre-key bundle from the server, then derives the shared secrets.

Symmetric ratchet stage. Suppose Alice wants to send another message to Bob but has not yet received any reply from him. She uses the symmetric ratchet to derive a new message key. She also sends Bob a fresh ephemeral ratchet public key he can use in his reply to Alice.

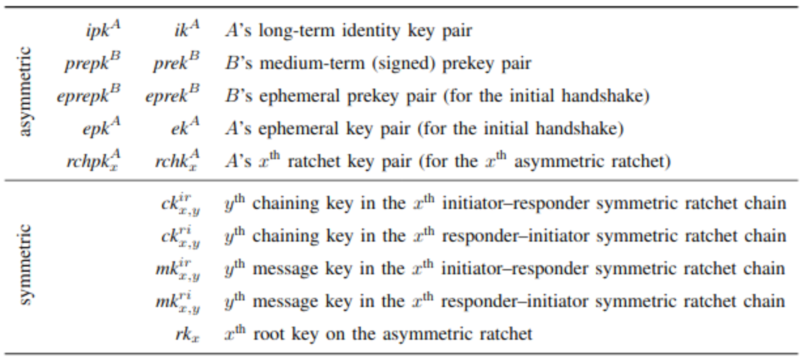
Asymmetric ratchet stage. For Bob to send a message to Alice, he generates and sends a fresh ephemeral ratchet public key, derives chaining and message keys, and encrypts his message.

When Alice replies to Bob, she completes the asymmetric ratchet stage by generating and sending a fresh ephemeral ratchet public key, derives chaining and message keys, and encrypts her message.

Symmetric ratchet stage. Suppose Alice wants to send another message to Bob but has not yet received any reply from him. She uses the symmetric ratchet to derive a new message key. She also sends Bob a fresh ephemeral ratchet public key he can use in his reply to Alice.

Initiator A
Server S
Responder B
RegisterPreKeyBundle
RegisterPreKeyBundle
RequestPreKeys, B
PreKeyBundle$_B$
$rk, ck, mk \leftarrow \mathrm{KDF}(\text{triple-DH shared secret})$
EncryptedMessage$_{mk}$, $rchpk_0^A, 0$
RequestPreKeys, A
PreKeyBundle$_A$
$ck', mk' \leftarrow \mathrm{KDF}(ck)$
NextEncryptedMessage$_{mk'}$, $rchpk_1^A$
$ck'', mk'' \leftarrow \mathrm{KDF}(rk, \mathrm{DH}(rchpk_1^A, rchpk_1^B))$
NewEncryptedMessage$_{mk''}$, $rchpk_1^B$
$rk', ck''', mk''' \leftarrow \mathrm{KDF}(ck'', \mathrm{DH}(rchpk_2^A, rchpk_1^B))$
NewEncryptedMessage$_{mk'''}$, $rchpk_2^A$
$ck'''', mk'''' \leftarrow \mathrm{KDF}(ck''')$
NextEncryptedMessage$_{mk''''}$, $rchpk_3^A$

Figure 1: Message flow of an example Signal execution between two clients A and B via a server S. Notation and someoperations have been simplified for clarity.



asymmetric	ipk^A	ik^A	A's long-term identity key pair
	$prepk^B$	$prek^B$	B's medium-term (signed) prekey pair
	$eprepk^B$	$eprek^B$	B's ephemeral prekey pair (for the initial handshake)
	epk^A	ek^A	A's ephemeral key pair (for the initial handshake)
	$rchpk_x^A$	$rchk_x^A$	A's xth ratchet key pair (for the xth asymmetric ratchet)
symmetric		$ck_{x,y}^{ir}$	yth chaining key in the xth initiator–responder symmetric ratchet chain
		$ck_{x,y}^{ri}$	yth chaining key in the xth responder–initiator symmetric ratchet chain
		$mk_{x,y}^{ir}$	yth message key in the xth initiator–responder symmetric ratchet chain
		$mk_{x,y}^{ri}$	yth message key in the xth responder–initiator symmetric ratchet chain
		rk_x	xth root key on the asymmetric ratchet

Figure 2: Notation table for a better comprehension of Figure 1

3.2 MTProto - Mobile Protocol (Telegram)

MTProto is a symmetric encryption scheme for data exchanging protocol developed by Telegram for its own messaging service. It is based on 256-bit symmetric AES encryption, SHA-256, 2048-bit RSA encryption and Diffie–Hellman key exchange. [12]. The currently used version of MTProto is the MTProto 2.0.

Before a message is transmitted over the network it is encrypted in a certain way and an extra header is added consisting of a 64-bit key identifier **auth_key_id** (that uniquely identifies an authorization key for the server as well as the user) and a 128-bit message key **msg_key**.

The authorization key **auth_key** combined with the message key **msg_key** define an actual 256-bit key **aes_key** and a 256-bit initialization vector **aes_iv**, which are used to encrypt the message using AES-256 encryption in infinite garble extension (IGE) mode. Note that the initial part of the message to be encrypted contains variable data (session, message ID, sequence number, server salt) that obviously influences the message key (and thus the AES key and iv). In **MTProto 2.0**, the message key is defined as the 128 middle bits of the SHA-256 of the message body (including session, message ID, padding, etc.) prepended by 32 bytes taken from the authorization key. In the older MTProto 1.0, the message key was computed as the lower 128 bits of SHA-1 of the message body, excluding the padding bytes. [13]

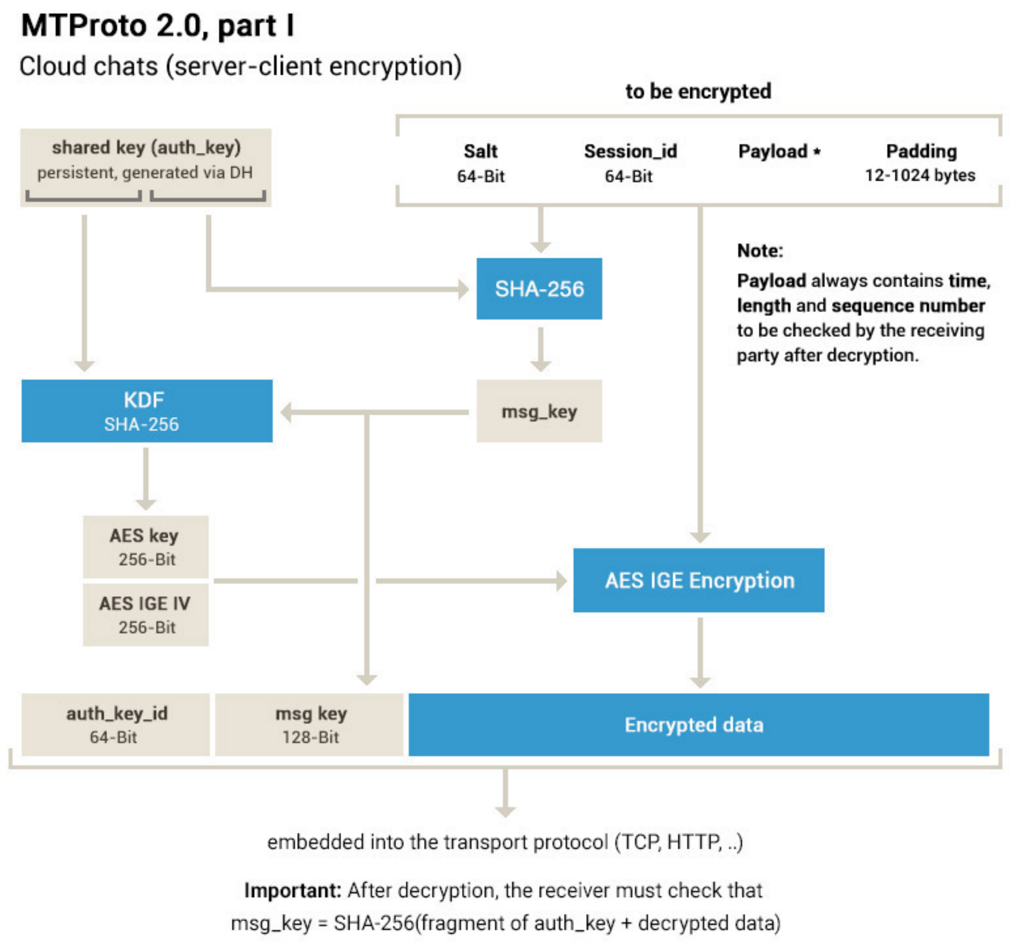

MTProto 2.0, part I
Cloud chats (server-client encryption)
to be encrypted
shared key (auth_key)
persistent, generated via DH
Salt
64-Bit
Session_id
64-Bit
Payload *
Padding
12-1024 bytes
SHA-256
Note:
Payload always contains time, length and sequence number to be checked by the receiving party after decryption.
KDF
SHA-256
msg_key
AES key
256-Bit
AES IGE IV
256-Bit
AES IGE Encryption
auth_key_id
64-Bit
msg key
128-Bit
Encrypted data
embedded into the transport protocol (TCP, HTTP, ..)
Important: After decryption, the receiver must check that
msg_key = SHA-256(fragment of auth_key + decrypted data)

Figure 3: MTProto Encryption Scheme

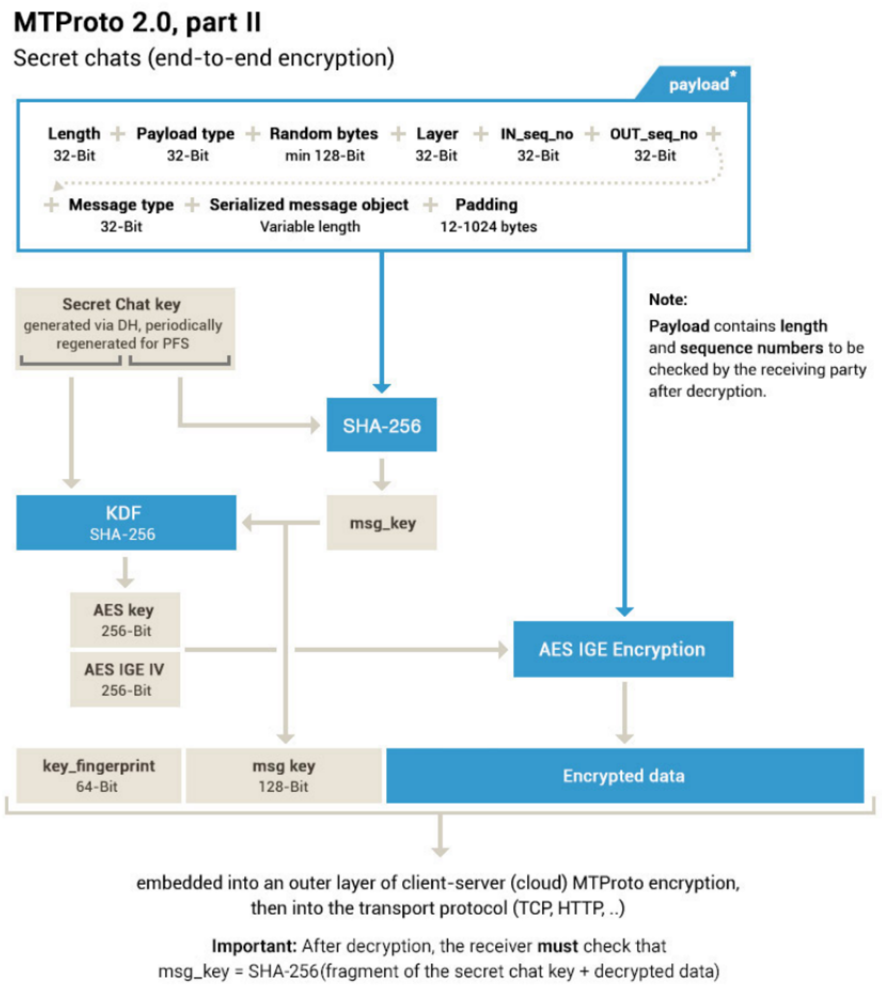
In the **cloud-chat** system, the default Telegram's chat system, after a message has been encrypted with MTProto, it is then transmitted to Telegram Messenger LLP's servers.

Then on the servers messages are decrypted and stored encrypted [2], to

[2]This step is deduced, but it is not known: since the code for the client is open source, the code on the cloud servers is not public and so nobody except Telegram's developers know what really happens over there. This hypothesis has been done because some components of the encrypted message, such as msg_id, are time-dependent.

be encrypted again and sent to the receiver with the Figure 3 scheme.

Secret Chats Differently from cloud-based chats, Telegram's secret chats provide an End-to-End Encryption and messages can be exchanged in a client-to-client mode [14]. These messages are encrypted with MTProto protocol, too. Unlike the default ones, this kind of chats can only be accessed on the device upon which the secret chat has been started and on the device upon which the secret chat has been accepted.
In this case, the server has the role to receive from client A, encrypt and resend the messages to client B, without storing any data.


MTProto 2.0, part II
Secret chats (end-to-end encryption)
payload*
Length 32-Bit
Payload type 32-Bit
Random bytes min 128-Bit
Layer 32-Bit
IN_seq_no 32-Bit
OUT_seq_no 32-Bit
Message type 32-Bit
Serialized message object Variable length
Padding 12-1024 bytes
Secret Chat key generated via DH, periodically regenerated for PFS
Note: Payload contains length and sequence numbers to be checked by the receiving party after decryption.
SHA-256
msg_key
KDF SHA-256
AES key 256-Bit
AES IGE IV 256-Bit
AES IGE Encryption
key_fingerprint 64-Bit
msg key 128-Bit
Encrypted data
embedded into an outer layer of client-server (cloud) MTProto encryption, then into the transport protocol (TCP, HTTP, ..)
Important: After decryption, the receiver must check that msg_key = SHA-256(fragment of the secret chat key + decrypted data)

Keys are generated using the **Diffie-Hellman** protocol: if Alice wants to start a secret chat with Bob, she asks the server to obtain the DH parameters and then opens an EncryptedChat session passing the appropriate DH generated numbers. After Bob confirms the creation of a secret chat with Alice, he receives the configuration parameters for the DH method. Then Alice and Bob exchange their keys and if no problems occurred they can start exchanging encrypted messages.

3.3 Comparison

As seen, the two protocols are very different even if they share some similarities.

Cryptographic Primitives On the primitive cryptographic functions side both Signal and MTProto make large use of 256-bit AES encryption, SHA-256 and respectively, for asymmetric encryption phases, Elliptic Curves or RSA.

Asynchronous Communication To cope with the situations when users are not online at the same time two different approaches have been developed: Signal Protocol uses a **Key Distribution Server** solution where servers are only intermediary and they are only deputed to store and relay information to let the communication being feasible and secure between the parties providing End-to-End Encryption, while MTProto bases all its functioning around the **cloud** server side, where servers compute encryptions and decryptions, store and forward data to the interested users.

The two different methods bring to one of the main differences between the applications, that is the fact that in Telegram it is possible to access to all conversations from different devices with the same account, cause the data are stored in the cloud, while in WhatsApp this is not possible, because all data are only available on the unique client.

Keys Dinamicity Even if in both the protocols some public and private keys are generated at the registration and then they never change, it has to be said that in each of them is present a sort of key dinamicity: in MTProto it is granted by some values that are time-related, such as the Server Salt or the Message Identifier; different is the situation for Signal Protocol, in which the dinamicity is provided by the generation of ephemeral public keys, especially when talking about synchronous messaging, where chains of this kind of keys grant the fact that even if the conversation is intercepted by a Man In The Middle, not all messages can be read in clear.

4 Conclusions

As seen, the two applications are finely embroidered around the respective protocols, and the protocols themselves maintained and updated by the same companies who own the applications.

Over time new and stronger security techniques have been developed and applied to the protocols, that are now proven to be secure and well accepted (albeit not without controversies) by the cyber-security community.

However one last less technical consideration is a must: even if no doubts are left on the strength of the protocols, as seen the user is forced between a rock and a hard place, and somehow whom who wants to use one service or the other has to choice between being profiled, as we saw with WhatsApp, or having all his conversations somewhere accessible by some people with the keys, as we saw in the case of Telegram.

Real security should not have compromises.

References

[1] cfr. `https://en.wikipedia.org/wiki/WhatsApp#Technical`

[2] cfr. *Gaurav Rathee*, "Explore WhatsApp Clock Sign, Single Tick, Double Tick"
`https://digitalperiod.com/explore-whatsapp-clock-sign-and-tick/`

[3] WhatsApp Privacy Policy, "The Information WhatsApp Does Not Collect"
`https://www.whatsapp.com/legal?doc=privacy-policy&version=20120707`

[4] crf. `https://en.wikipedia.org/wiki/Telegram_(software)`

[5] cfr. *Advox GlobalVoice*s, "Is Telegram Really Safe for Activists Under Threat? These Two Russians Aren't So Sure."
`https://bit.ly/39BKbv5`

[6] cfr. *Reuters*, "Exclusive: Hackers accessed Telegram messaging accounts in Iran - researchers"
`https://reut.rs/39nnvhV`

[7] Telegram Privacy Policy, 3.3.1 "Cloud Chats"
`https://telegram.org/privacy`

[8] WhatsApp Privacy Policy, "The Information WhatsApp Collects"
`https://www.whatsapp.com/legal?doc=privacy-policy&version=20120707`

[9] cfr. *Frosch, Tilman; Mainka, Christian; Bader, Christoph; Bergsma, Florian; Schwenk, Jörg; Holz, Thorsten*
"How Secure is TextSecure?"(March 2016), pp. 457-472

[10] cfr. *Frosch, Tilman; Mainka, Christian; Bader, Christoph; Bergsma, Florian; Schwenk, Jörg; Holz, Thorsten*
"How Secure is TextSecure?", sec. 3, pp. 3
`https://eprint.iacr.org/2014/904.pdf`

[11] cfr. *Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, Douglas Stebila*
"A Formal Security Analysis of the Signal Messaging Protocol", Extended Version, July 2019, cpt. 2
`https://eprint.iacr.org/2016/1013.pdf`

[12] Telegram, FAQ for the Technically Inclined,
https://core.telegram.org/techfaq#q-where-can-i-read-more-about-the-protocol

[13] Telegram, Protocol Description
https://core.telegram.org/mtproto/description

[14] cfr. https://en.wikipedia.org/wiki/Telegram_(software)#Secret_chats