

INTRODUCTION

In this study, three different popular secure instant messaging applications namely TELEGRAM, THREEEMA and SIGNAL are compared. Applications are investigated in terms of their authenticated and unauthenticated messaging. Also when man-in-the-middle attacks are possible and how can peers sure whether there is a man-in-the-middle or not. During the study, applications behaviors are tested.

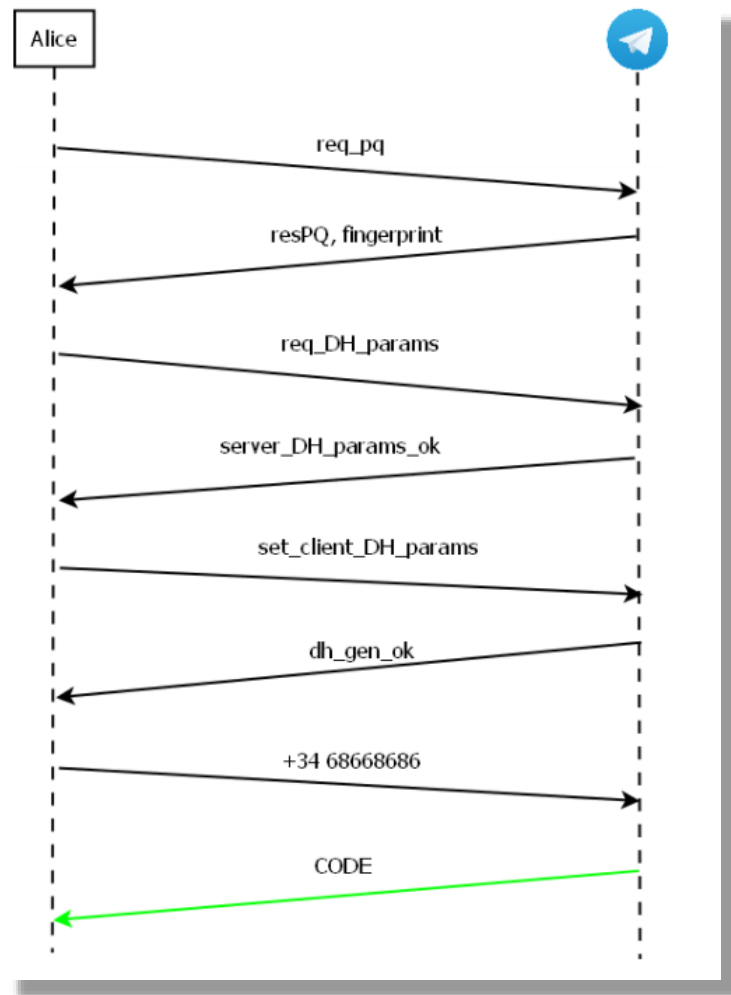
TELEGRAM [\[1\]](#)

In Telegram, when installing a new client in a device, client and server negotiate a shared key that will be used from that moment onwards to cipher all communications between them.

When a user (new or not) installs a Telegram's client in a compatible device, this client communicates with Telegram's servers in order to create a shared key. This key is called authorization key and will be subsequently used to cipher all communications between client and server, having a high persistence

For negotiating the key, Telegram uses the Diffie-Hellman key exchange protocol. Hence, the key itself is never transmitted in clear text nor encrypted. For the user to know that she is communicating with the legitimate Telegram's server, the latter specifies its public key (more precisely, the fingerprint of its public key) in the first message the server sends to the client.

Below diagram summarizes the telegram authentication steps.



1. The user, Alice, sends a request `req_pq` to Telegram's server. This message contains a nonce, generated by Alice herself, denoted `nonce`.
2. Telegram responds with the `resPQ` message, containing a new nonce generated by the server, named `server_nonce`. Additionally, it includes a small composite integer $n=pq$, product of two primes p and q , and the fingerprint of its public key, `fingerprint`. From this moment on, all messages will contain the pair `(nonce, server_nonce)`, used to identify the session.
3. Alice verifies that it has the public key associated to `fingerprint` and, if affirmative, she factorizes n , getting p and q . She then creates a new random number, `new_nonce`, and sends the three values encrypted with Telegram's public key, composing the message `req_DH_params`.
4. Telegram deciphers the received values and verifies the factorization of n . If it is correct, it uses `new_nonce` and `server_nonce` to generate a symmetric key that will be temporarily used with AES working in IGE mode. It will also generate its contribution to the Diffie-Hellman exchange, g_a . Telegram sends g_a to Alice, encrypted with the AES temporary key. This is the `server_DH_params_ok` message.

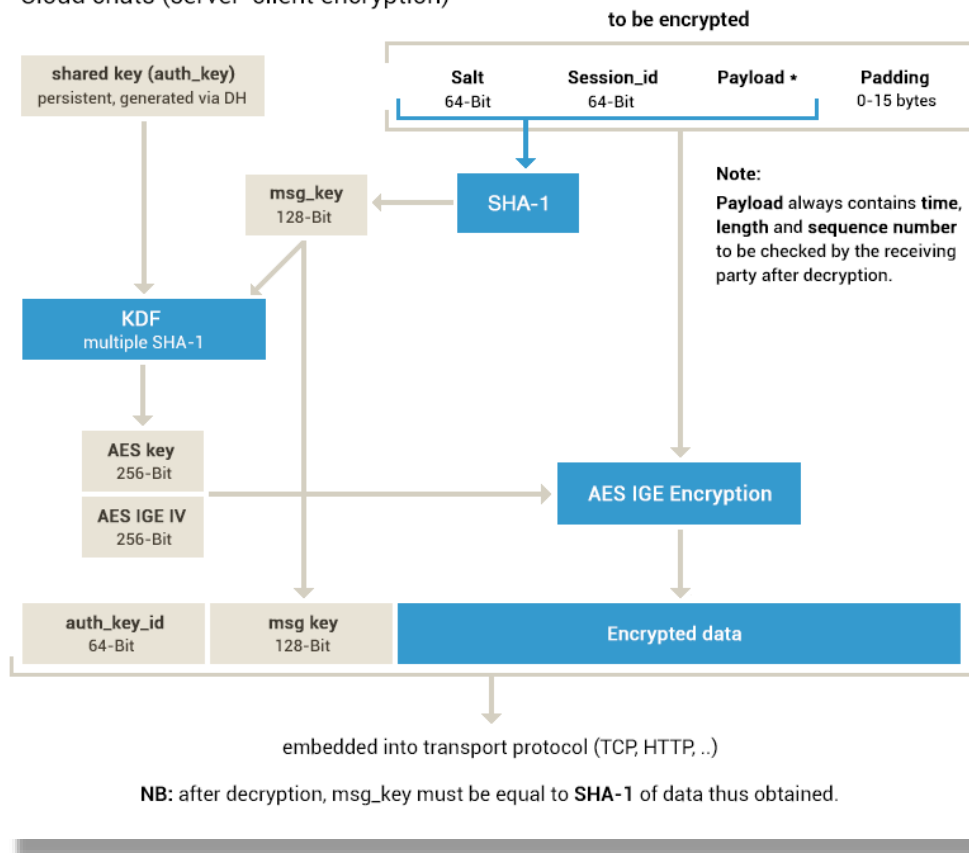
5. Alice, knowing `new_nonce` and `server_nonce`, derives the same AES key and decrypts `ga`. She checks that it complies with the necessary cryptographic properties and then generates her own `gb`. At this point, the client knows that in case of a successful completion of the protocol, the final authorization key will be `gab`. In her last message, `set_client_DH_params`, Alice sends `gb` to Telegram, also encrypted with the temporary AES key.

6. Telegram decrypts the received value and sets the authorization key to `gab = gba`. As an acknowledgement, Telegram sends a last message to Alice, `dh_gen_ok`, which includes a SHA1 hash with a specific syntax which depends on the values of `new_nonce` and authorization key.

Once Alice has established the key with Telegram's server, the latter redirects her to the server (or servers) physically near to her, with whom Alice repeats the same process. Finally, Telegram sends Alice an SMS including a code to the phone number specified by her. After sending back the code, the authentication is completed.

MTPROTO, part I

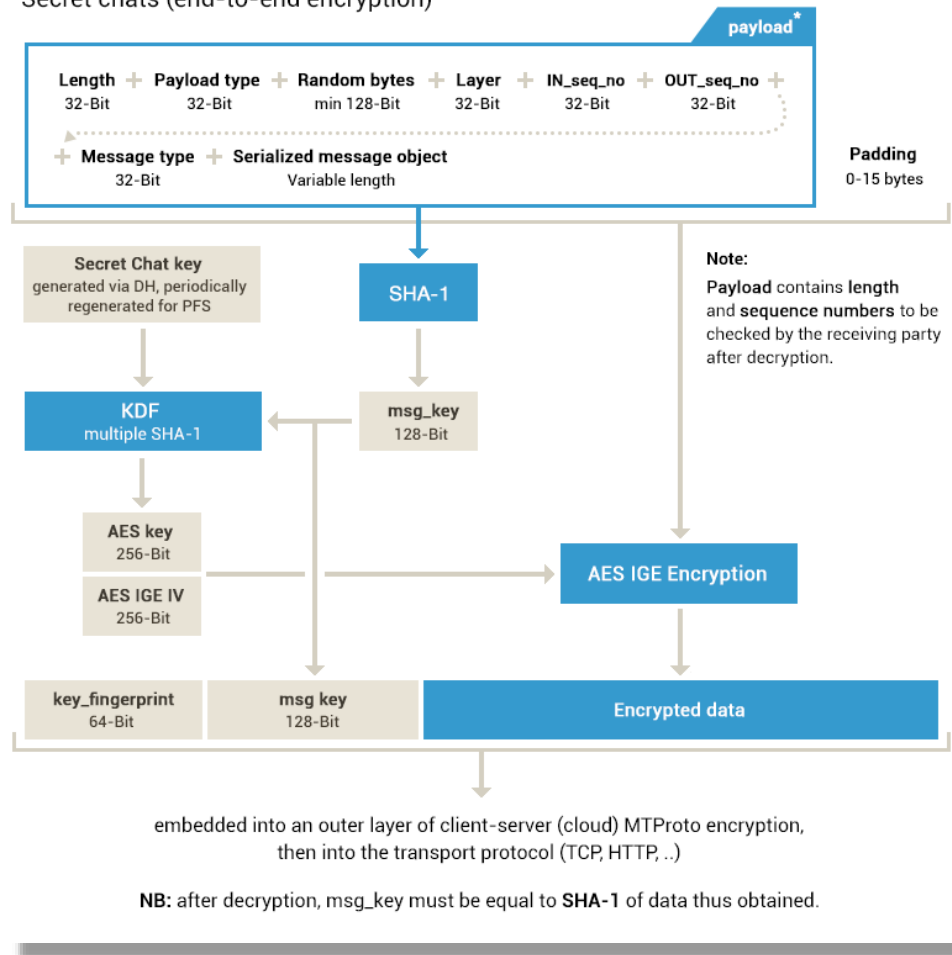
Cloud chats (server-client encryption)



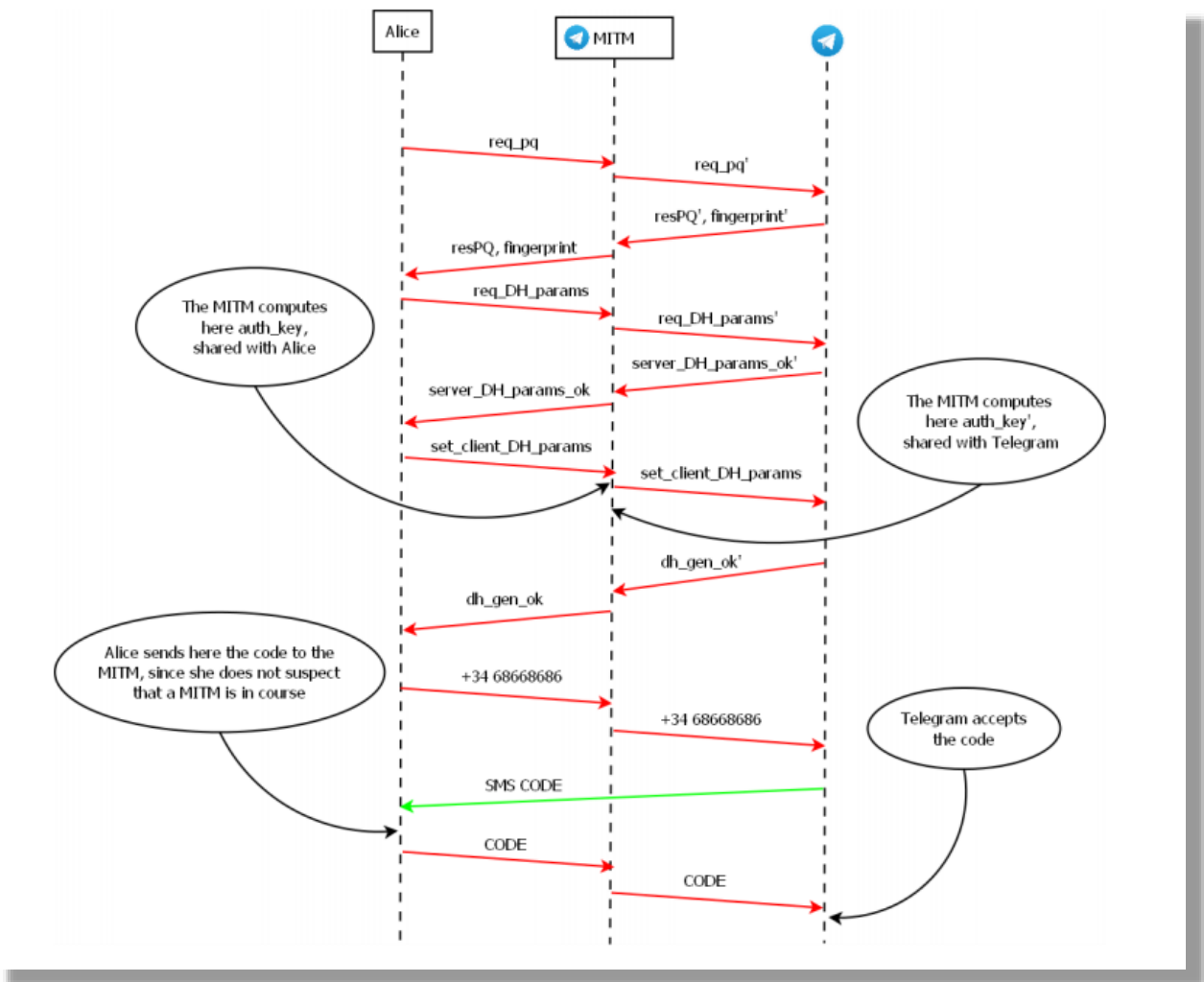
The above diagram shows the telegram protocol flow for client-server namely cloud chat.

MTPROTO, part II

Secret chats (end-to-end encryption)



The above diagram shows the end-to-end encryption namely secret chat protocol flow.



The above diagram depicts the MITM attack while in cloud chats. In end-to-end communication it is also possible to make MITM but Telegram offers to authenticate the peers. This method is described below.

What should an attacker do to launch the attack?

Briefly, he would have to change Telegram's public key (and the IP address, or perform a "live" spoofing attack), included in the client, and distribute the malicious client to the victim(s). With this, the client would communicate with the attacker, who would then be able to decrypt the third message of the protocol, since it has been encrypted with the attacker's public key.

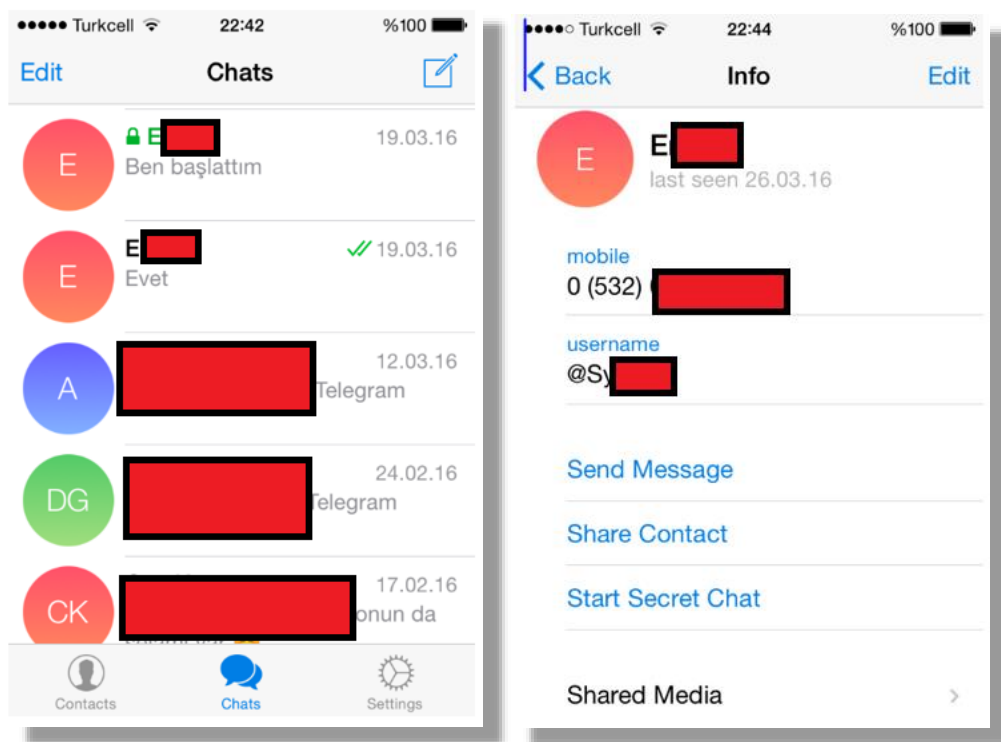
Thus, the attacker will obtain the new nonce value, and will be able to derive the temporary key and decrypt and re-encrypt all the messages, emulating the legitimate protocol (i.e., recalculating the integrity protection hashes in case of making some modification, etc.) until he establishes an authorization key with each communication end.

Does the SMS code prevent the attack?

No. Since the user will still receive the message (the attacker does not modify the cellphone number). Given that it is the legitimate user who initiates the authentication process, she will not suspect that an attack is in course, and will send the code to Telegram, via the attacker. In case of having previously installed Telegram in another device, the victim will receive a message informing that a new authentication has been performed from the IP address x.y.z.v. However, probably only the advanced users will take care of verifying that the specified IP address actually matches the one assigned to their device.

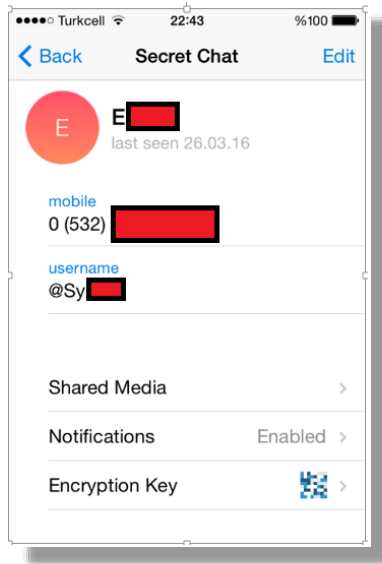
Using Mobile Telegram Application:

The cloud messages are appeared as without a lock sign. In this messaging form, messages relayed from telegrams servers. This means that telegram owners can manipulate the traffic and listen the conversation. Also MITM attack can be done as described above. The conversation can be end-to-end secured by “start secret chat”.

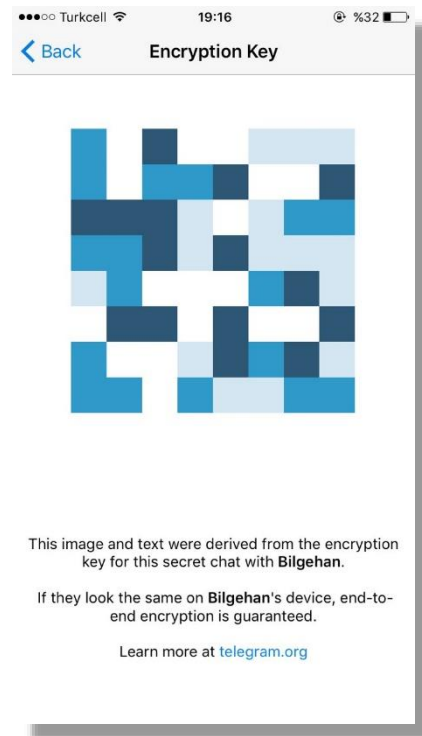


Instead of using cloud messaging, anybody shall prefer for end-to-end communication. Here also MITM attack is possible but telegram offers to the peers that they can verify if the peer is real by comparing encryption key representation diagram. Anyone can compare diagram by just seeing the other peer's diagram or diagrams can be sent from other trusted channels. Although this

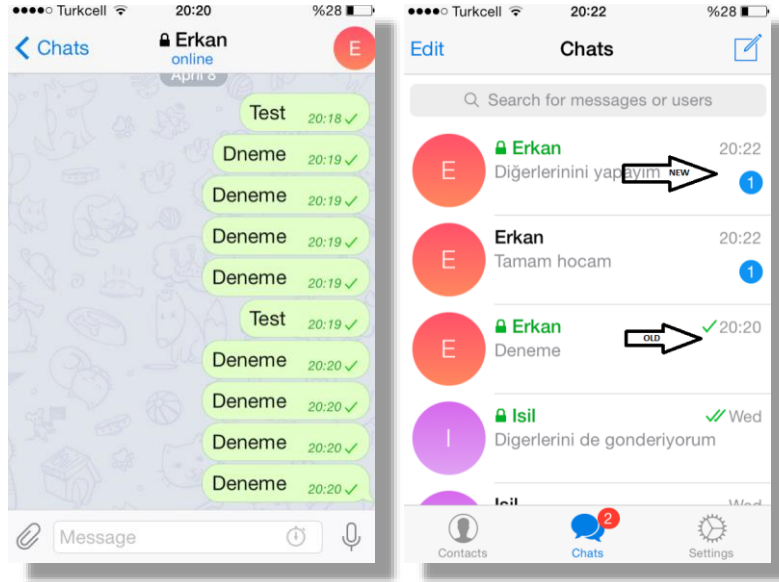
method is not hundred percent secure, it is feasible compared with other method when peers are not close. Telegram does not support end-to-end secure messaging for group chats.



Comparison can be done only with visual inspection. Telegram does not offer scanning the representation of the encryption key a QR code like others. QR codes are specific to the conversations. This mean every conversation shall be authenticated to detect MITM attack presents or not. The diagrams are the same but the applications used during test runs on different IOS. That is why the messages at the end are different.



During the tests, application was uninstalled and reinstalled or id deleted from the one peer. By doing this, any secured communication resides on the peers are useless. Any messages written to old identity does not do anything from telegram application since the identity was renewed. As shown below only one tick exists on messages. User only sees the messages that are not sent. Application is secure but does not warn the user.



THREEMA [2]










In Threema, all messages (whether they are simple text messages, or contain media like images, videos or audio recordings) are end-to-end encrypted. For this purpose, each Threema user has a unique asymmetric key pair consisting of a public and a private key based on Elliptic Curve Cryptography.

Key Generation and Registration

When a Threema user sets up the app for the first time, the following process is performed:

1. The app generates a new key pair by choosing a private key at random¹, storing it securely on the device, and calculating the corresponding public key over the Elliptic Curve (Curve25519).
2. The app sends the public key to the server.
3. The server stores the public key and assigns a new random Threema ID, consisting of 8 characters out of A-Z/0-9.
4. The app stores the received Threema ID along with the public and private key in secure storage on the device.

Verification Levels of Peer:

-    Red (level 1): The public key has been obtained from the server because a message has been received from this contact for the first time, or the ID has been entered manually. No matching contact was found in the user's address book (by phone number or email), and therefore the user cannot be sure that the person is who they claim to be in their messages.
-    Orange (level 2): The ID has been matched with a contact in the user's address book (by phone number or email). Since the server verifies phone numbers and email addresses, the user can be reasonably sure that the person is who they claim to be.
-    Green (level 3): The user has personally verified the ID and public key of the contact by scanning their 2D code. Assuming the contact's device has not been hijacked, the user can be very sure that messages from this contact were really written by the person that they indicate.

Message Encryption

Threema uses the “Box” model of the NaCl Networking and Cryptography Library to encrypt and authenticate messages. For the purpose of this description, assume that Alice wants to send a message to Bob.

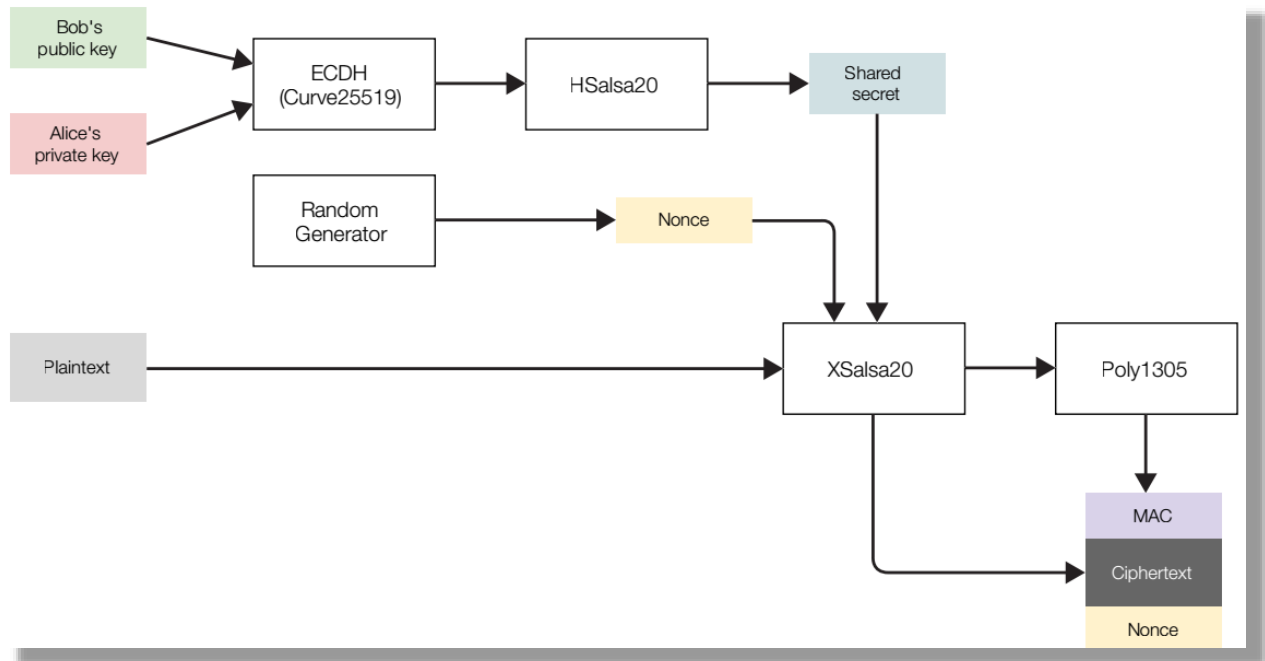
Encryption of the message using NaCl works as follows:

Preconditions

1. Alice and Bob have each generated a key pair consisting of a public and a private key.
2. Alice has obtained the public key from Bob over an authenticated channel.

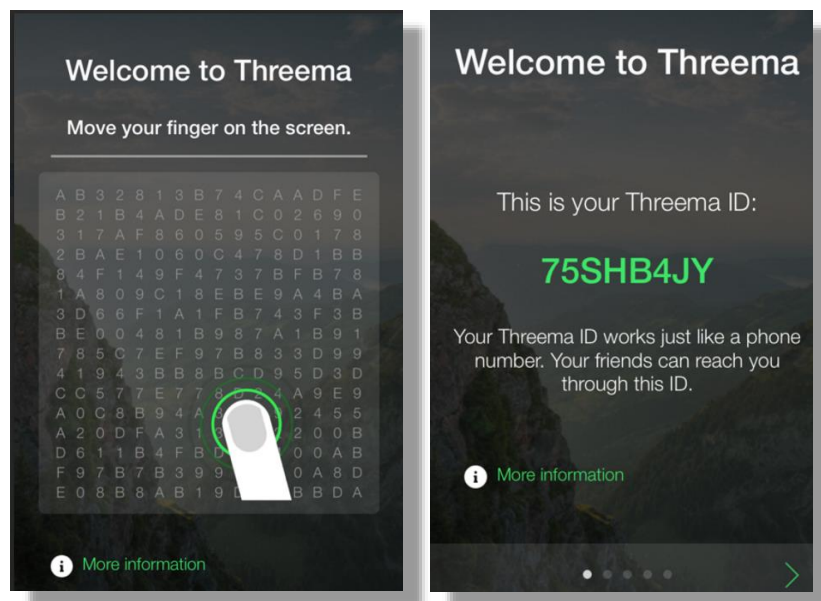
Procedure to encrypt a message:

1. Alice uses Elliptic Curve Diffie-Hellman (ECDH) over the curve Curve25519 and hashes the result with HSalsa20 to derive a shared secret from her own private key and Bob's public key. Note that due to the properties of the elliptic curve, this shared secret is the same if the keys are swapped (i.e. if Bob's private key and Alice's public key are used instead).
2. Alice generates a random nonce.
3. Alice uses the XSalsa20 stream cipher with the shared secret as the key and the random nonce to encrypt the plaintext.
4. Alice uses Poly1305 to compute a Message Authentication Code (MAC), and prepends it to the ciphertext. A portion of the key stream from XSalsa20 is used to form the MAC key.
5. Alice sends the MAC, ciphertext and nonce to Bob. By reversing the above steps and using his own private key and the Alice's public key, Bob can decrypt the message and verify its authenticity.

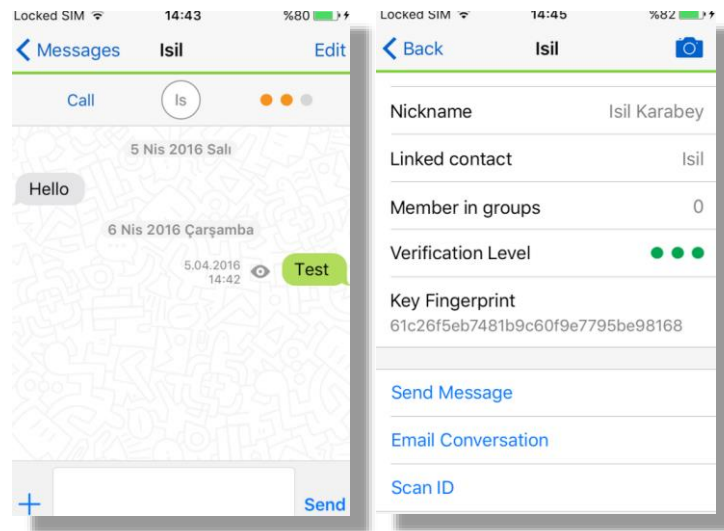


Using Mobile Threema Application:

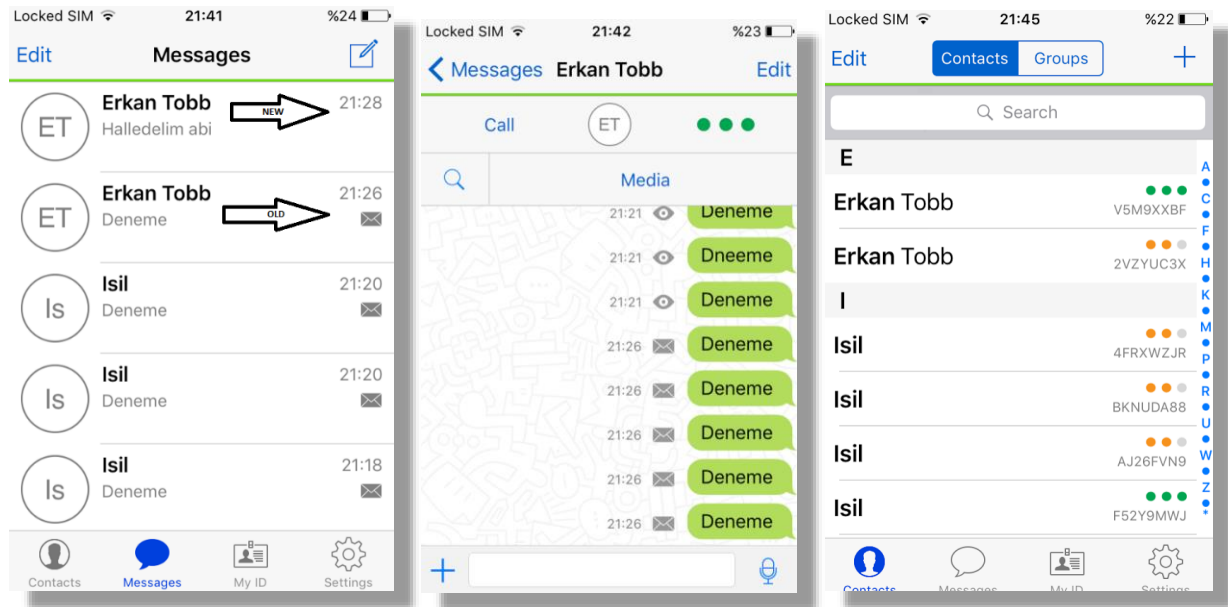
After installing Threema, application generates the key pair and for randomization it requires some movements. After key pairs are generated public key send to threema and ID generated and send to the client.



Conversation can be started anytime with any peer. Threema gives the levels for the peers as described above sections. To make the conversation hundred percent secure, (ie. Green) you shall scan the QR code of the peer, or you can send QR-code to the peer in other secure communication channels although this is not hundred percent secure but feasible. When the peer is verified, verification level is green.

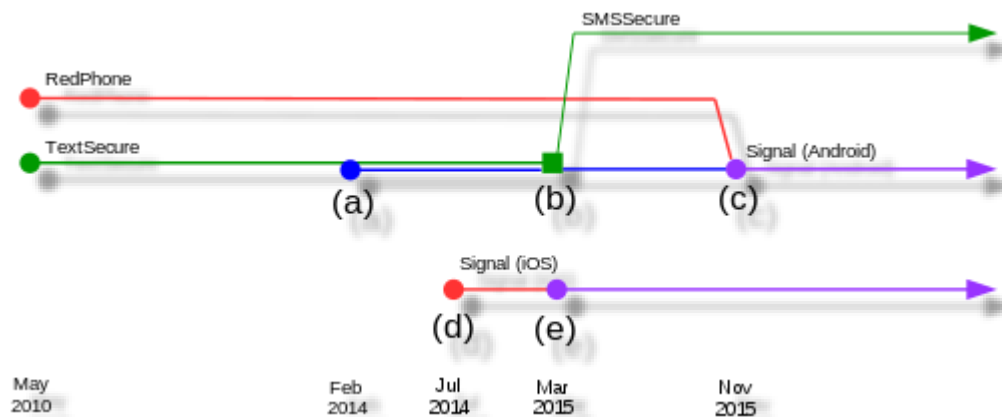


During the tests, one of the peer has uninstalled the application and reinstalled or id deleted. We faced with some inconsistencies that when ios application has uninstalled application id is not removed and id is kept, no sms sent for registration but android application directly remove id when uninstall the application. Old conversations kept and no messages are sent. No errors or warnings are displayed about the user has new id. Below screenshots shows that new conversation added for the peer. The old one is kept and messages can be sent but in waiting status. The application does not warn the user about changed identity. User can manually verify the identity of the peer.



SIGNAL^[3]

Signal is formerly based on Textsecure for secure messaging, Redphone for secure voice calls.



The above diagram shows the timeline of signals development.

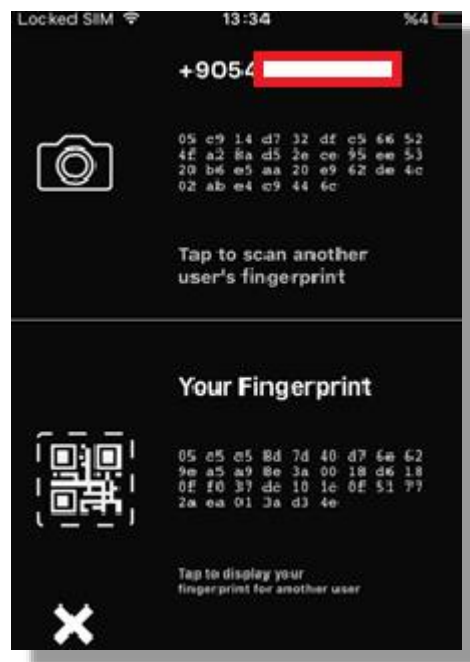
Signal's Properties:

1. All communications to other Signal users are automatically end-to-end encrypted.
2. The keys that are used to encrypt the user's communications are generated and stored at the endpoints (i.e. by users, not by servers).

- Signal also has built-in mechanisms for verifying that no man-in-the-middle attack has occurred. For calls, Signal displays two words on the screen. If the words match on both ends of the call, the call is secure. For messages, Signal users can compare key fingerprints (or scan QR codes) out-of-band. The user is notified if the correspondent's key changes.

Using Mobile Signal Application:

Signal application is by default using end-to-end encryption. The secure messaging can be verified by scanning the QR-code of the peer.



During the tests, one of the peer has uninstalled and reinstalled the application. When trying to rescan the code, signal gives the conflict error. Also signal gives the error when you open the conversation. Signal allows to open one conversation at a time for a peer unlike the others.

SUMMARY

As a summary, the following comparison chart is prepared. Comparisons show that Signal is the most secure instant messaging platform compared with the others.

Our comparison chart: [4]

	Telegram	Threema	Signal
Encrypted in Transit	Yes	Yes	Yes
Opensource	Yes	No	Yes
Support E2E Secure Communication	Yes (when enabled)	Yes (by default)	Yes (by default)
PFS	Yes (in secret chat)	Yes	Yes
Secure Voice Call Support	No	No	Yes
Support Group Chat Secure Communication	No	Yes	Yes
QR code/Diagram verification for E2E Secure Communication	Yes	Yes	Yes
QR-code scanning	No	Yes	Yes
MITM is possible on unauthenticated messaging	Yes (for every messaging)	Yes (in first conversation. Give Auth level of peers)	Yes (in first conversation)
MITM is possible on authenticated messaging	No (Does not send messages .No warning)	No (Does not send messages. No warning.)	No (Actively warn users and does not send messages and receives.)
What does QR code represents?	Session key	Long Term Key	Long Term Key
Warn users when a peer changes identity	No	No	Yes
Free	Yes	No	Yes
Edward Snowden Used it [5]	No	No	Yes

References:

[1]https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/INT_Telegram_EN.pdf

[2] https://threema.ch/press-files/cryptography_whitepaper.pdf

[3] [https://en.wikipedia.org/wiki/Signal_\(software\)](https://en.wikipedia.org/wiki/Signal_(software))

[4] <https://www.eff.org/secure-messaging-scorecard>

[5] https://twitter.com/Snowden/status/661313394906161152?ref_src=twsrc%5Etfw

PS: This study is the research homework of TOBB Computer Engineering Bil548 – Internet security protocols course. Instructor is Prof.Ali Aydin Selçuk.

This paper is prepared by Bilgehan TURAN and Erkan Erdoğan