Understanding a real-world protocol
HW5 - CNS Sapienza

Gianfranco Romani 1814407

30 November 2020

1 Introduction

We all use every day messaging apps to communicate with friends, family or for work. But how much secure are these platforms? In this report I will talk about Telegram and in particular I will present MTProto, the encryption protocol used by the company to ensure privacy for their customers, to verify if it really is as secure as they claim.

2 Telegram

Telegram is one of the most famous and most used messaging apps right now, with hundreds of millions monthly active users in 2020. It was created to provide surveillance-proof communication for the users, so privacy and security are their main concern. In fact, along with a lot of cool features and support across multiple devices, the developers of this app provide different way to protect our data, such as end-to-end encryption or self-destructive messages.

3 History

Before going into the details of the protocols used, it could be useful to delve into the past of the app to contextualize its development.

Telegram was founded in 2013 in Russia by two brothers, Nikolai and Pavel Durov, who previously founded the social network VK. Pavel Durov was removed as CEO of VK in 2014 and immediately after the Mail.ru group bought the company. Durov stated that he was dismissed because of political reasons, particularly because he refuses to hand over user's personal

details to the Russian government and to shut down groups dedicated to anti-corruption. After that, the brothers start to travel the world in self-imposed exile and to focus on Telegram, which MTProto protocol was in the meantime develop by Nikolai (which is a mathematician but not knows as an expert in security). The app is financed by Pavel because it is free, open source (at least the client part is) and it does not have adds.

4 MTProto

Telegram uses two different types of encryption for client's messages: server-client encryption, that is the one used by default, in which messages are encrypted between the cloud server and each user, and end-to-end encryption for secret chats. Both ways are based on 256-bit symmetric AES encryption, 2048-bit RSA encryption and Diffie–Hellman key exchange.

4.1 Server-client encryption (version2)

The protocol defines three independent areas:

- High-level component (API query language): defines how APIs queries are converted to binary messages;
- Cryptographic layer: the one analyzed in this report;
- Transport component: defines how messages must be exchanged.

The procedure to encrypt communications between the client and the server is shown in the image 1. The first step of the protocol consists in adding a header to the message that is composed by data that has to be checked in decryption. The data to be encrypted consists in a server salt (64 bit), changed periodically, a session id (64 bit), the payload (composed by time, length and sequence number that have to be checked by the receiving party) and some padding (12-1024 bytes).

The server key is a 2048-bit RSA key used by the server to sign its own messages while the auth_key is not yet generated. The application has a public server key to verify messages from the server. The private server key is changed rarely.

The session key is long 64 bit and identify an individual session. It can't be used for a different session from the one it was instantiated for. Internally a single session, messages have a message identifier (64 bit) that is incremented monotonically.

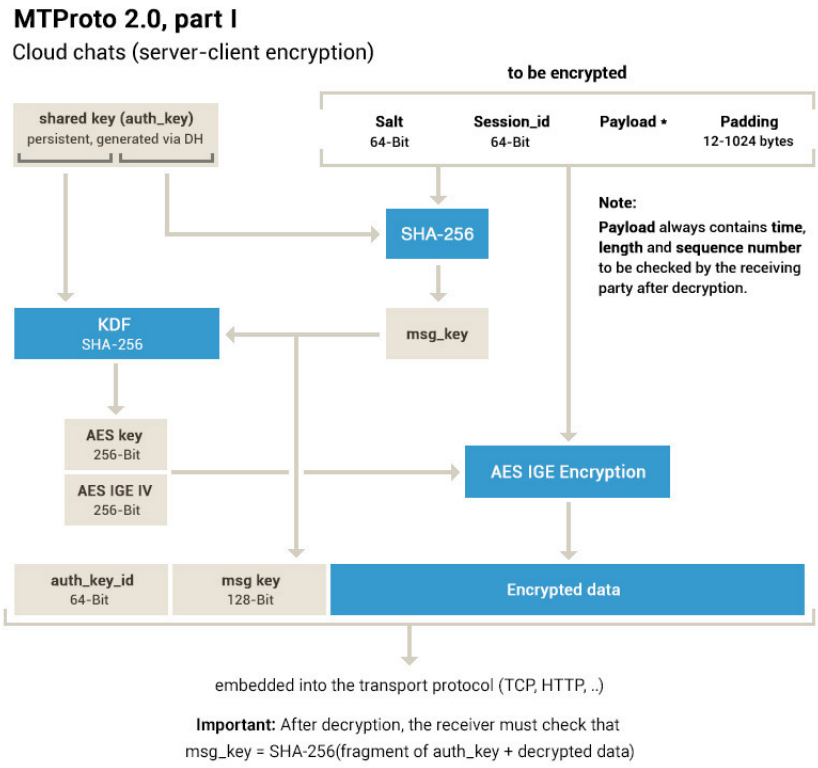

Figure 1: Telegram's server-client encryption model.

The message sequence number is a 32-bit long number that is set to twice the number of messages that require acknowledgement.

After the generation of these data, to obtain the msg_key, the auth_key is needed. auth_key is the authorization key, a 2048-bit key shared between client and server, created using the keys exchanged in Diffie-Hellmann algorithm and never transmitted over the network. It is unique for each user. The msg_key is generated by taking the middle 128 bits from the hash obtained using SHA-256 on 32 bit from the auth_key prepended to the message that has to be encrypted.

At this point, before encrypting the message, a 256-bit AES key (aes_key) and the 256-bit initialization vector (aes_iv) are needed. They are obtained using the whole auth_key and the msg_key, using the following algorithm (I take it directly from Telegram's site [1]):

- msg_key_large = SHA256 (substr (auth_key, 88+x, 32) + plaintext +

random_padding);

- msg_key = substr (msg_key_large, 8, 16);
- sha256_a = SHA256 (msg_key + substr (auth_key, x, 36));
- sha256_b = SHA256 (substr (auth_key, 40+x, 36) + msg_key);
- aes_key = substr (sha256_a, 0, 8) + substr (sha256_b, 8, 16) + substr (sha256_a, 24, 8);
- aes_iv = substr (sha256_b, 0, 8) + substr (sha256_a, 8, 16) + substr (sha256_b, 24, 8);

where x = 0 for messages from client to server and x=8 for those from server to client. Once aes_key and aes_iv are computed, the encryption of the message is done using AES encryption with IGE mode. IGE (Infinite Garble Extension) is a block cipher mode that follows the chaining sequence $y_i = E_k(x_i \oplus y_{i-1}) \oplus x_{i-1}$, where E_k is the block cipher encryption function with key k. For the first block, x_0 and y_0 correspond to the IV and are computed with a second key k'. The output of AES IGE encryption is the

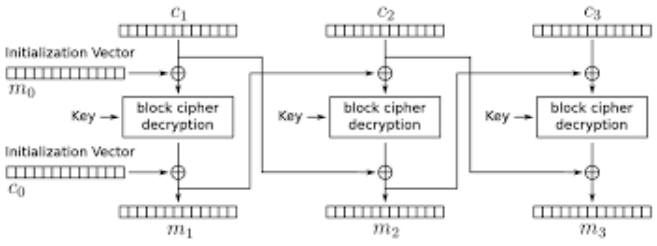

Figure 2: Infinite Garble Extension

encrypted data, at which are added as prefixes the auth_key_id, that are the lower 64 bits of the SHA-1 hash of the auth_key, and the msg_key.

To decrypt a received message it is needed to repeat in reversed order these steps. The receiver must also check if the SHA-256 hash of the decrypted message, plus the 32 bytes taken from the shared key, are equal to msg_key.

4.2 Server-client encryption (old version)

In the section above I described the actual version of MTProto, released at the end of 2017. The 1.0 version was based almost on the same scheme,

with few differences, but it was way less secure because, for example, it used SHA1 instead of SHA256 (SHA1 is still present in version 2.0 but Telegram says that it is used only where 'the choice of hash function is irrelevant for security' [2]). Other differences from the new version are the usage of padding bytes, that now are involved in the computation of msg_key and are increased form 0-15 bytes to 12-1024.

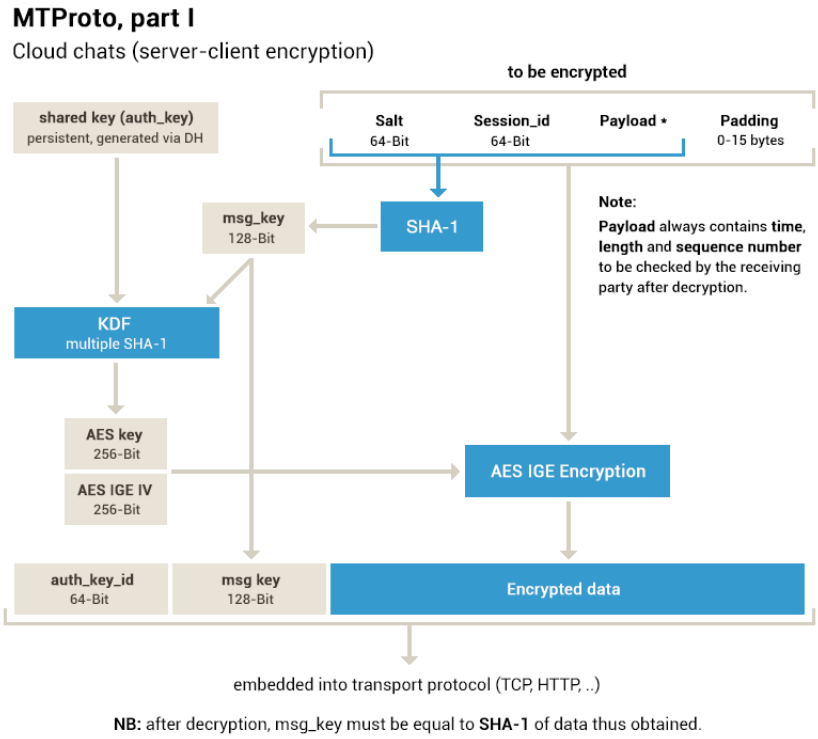

Figure 3: MTProto 1.0

4.3 End-to-End encryption

As already said, end-to-end encryption is not set as default in Telegram, but it can be used in secret chats between users. In this case the encryption keys are held only by the clients. The general structure of the protocol is not much different form the cloud-based one, just the payload message is more complex (and of course the keys are not stored in a server). Secret chat keys are generated using DH algorithm and are changed after some time. Figure

4 describes the algorithm. This schema is the 2.0 version, the older one

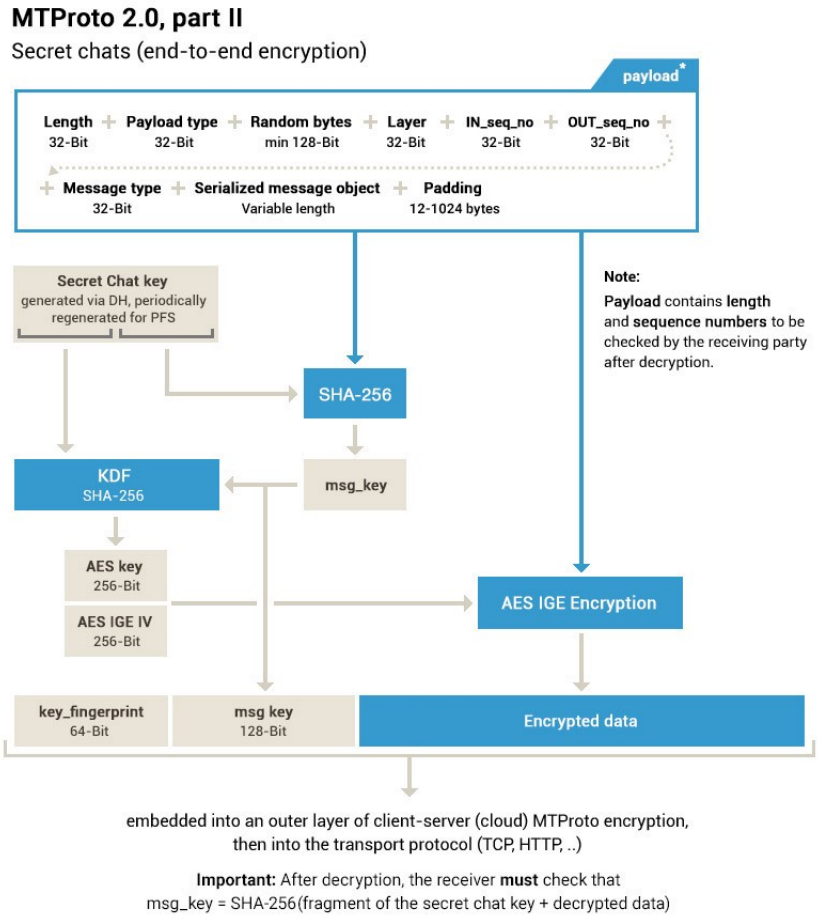

Figure 4: End-to-end encryption scheme

had the same differences that I showed before between the two client-server encryption schemes.

5 Comments

Telegram claims to have the most secure system for messages exchange on the market and to guarantee the privacy of the users like no other app of the kind. They also base this statements on their history and their particular interest in protecting users from governmental data requests. But no all the

experts are convinced by the level of security offered by MTProto and there have been also some controversial facts about the utilization of this app.

5.1 Security

Before the introduction of MTProto 2.0, the encryption scheme had several flaws . For example there were some problems related to how Diffie-Hellmann algorithm was implemented, problems that could lead to possible Man-In-The-Middle attacks. In particular, at the beginning, the DH key was just XORed with a nonce, but in this way an attacker could use different nonce variables for two users. Another big issue raised by cybersecurity experts is the use of SHA1 that is not secure because it is not collision-resistant. With MTProto Telegram claims that SHA1 is used only where it is not essential for security, but, as stated by experts, they should use some more powerful hash functions anyway. AES IGE encryption is not considered as a secure way to encrypt data too. The padding used in the algorithm is non-standard and this doesn't convince some experts.

Even if the flaws may seem a lot, it doesn't mean that the scheme is absolutely insecure. In fact it is known that bad primitive choices doesn't lead necessarily to an insecure protocol (and good primitives don't always guarantee that the protocol is great), but the doubts about the effective security are several.

5.2 Non technical problems

Besides the technical concernings, Telegram has raised also other kind of doubts. One of them is the fact that end-to-end encryption is not enabled by default, while other messaging apps use it extensively. An odd thing is that we have to give Telegram the list of our contacts when we start using the app for the first time and they store them on their servers, so users have to trust the fact that they won't sell them to other entities or that the data is stored safely from unauthorized malicious people. Telegram has also received criticism because of its use by radical groups, that were not banned by the app because of the right for privacy (but in the last years the company started to put efforts into the removal of extremist groups from the app).

5.3 Conclusions

In conclusion, despite all the claims, our data and privacy are not completely safe with Telegram, like with other applications of the same kind. The

doubts about this app are not only relative to its 'homemade' protocol, which may have or not some weak points, but also about how our data are stored and used once in their servers.

References

[1] https://core.telegram.org/mtproto/description

[2] https://core.telegram.org/techfaq#q-where-can-i-read-more-about-the-protocol

[3] https://www.links.org/files/openssl-ige.pdf