Homework #5
Telegram vs WhatsApp Security

CNS Course Sapienza

Riccardo Prinzivalle, 1904064

November 30, 2020

1 Homework Goal

This homework contains a basic introduction to Telegram and WhatsApp security, then a comparison of their security protocols (**MTProto** and **Signal** respectively), some past development and technical issues/vulnerabilities, and what it is possible to find about current threat and vulnerabilities.

2 Telegram Security basics

Telegram uses a security protocol called MTProto, developed by the telegram team. It is a symmetric encryption protocol based on 256-bit symmetric AES encryption, 2048-bit RSA encryption and Diffie–Hellman key exchange. The protocol is divided in 3 layers:

- **High-level** component which defines the method whereby API queries and responses are converted to binary messages.
- **Cryptographic/authorization** layer which defines the method used to encrypt messages prior to being transmitted through the transport protocol.
- **Transport** component, which defines the method for the client and the server to transmit messages over some other existing network protocol.

Let's analyze in the details every section. The high level component sees a client and a server exchanging messages inside a session, which is identified by a user key identifier (a particularity, the session is attached to the client instead of standard protocols such as http/s or tcp). The client can instantiate different connections to the server (the practicality of Telegram stands in the fact that one can open different sessions on many devices such as browsers without having to log in many times once one have the session active), and messages can be

sent from one connection to the other and everything is synchronized server side. The low level message structure can be seen in fig. 1.

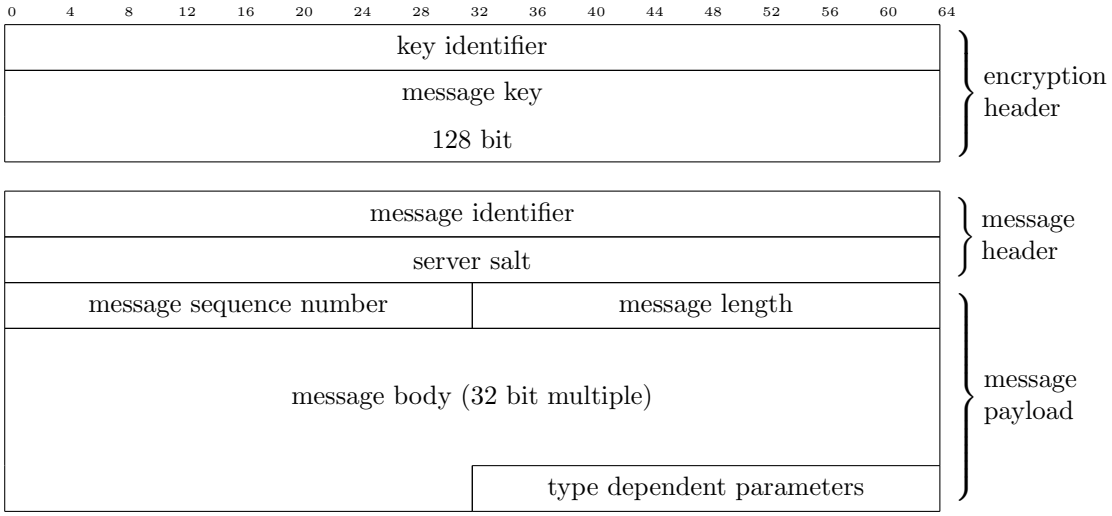

Figure 1: Telegram simplified packet structure

The message header is fixed for every message, the types of parameters does not change, and they are used by the encryption part, like message identifier and server salt.
To be mentioned, all number are saved as little endian, with the exception of large numbers, such as those needed by RSA and DH which are stored as big endian due to openSSL compatibility.
The encryption stuff can be identified in the upper part fig. 1, it is added at the end of encryption as header of the encrypted message. The message is encrypted using a 256-bit key constituted by the message key and the user key and the encryption is performed with AES-256. The message key is defined as the hash using SHA256 on the message body, and taking the 128 middle resulting bits. The user key is generated from the authorization key: it is created once, when the client is first run on the device, and never changes, so this will expose all messages if that key is stolen (even from the device or from server side); different counter measures can be taken:

- Use session keys generated at every session using the Diffie Hellman exchange protocol
- Store the keys on the device and protect them with a password
- Protect all stored and cached data of the device with a password

All these measures cannot protect the user in the case where is the server that is violated or some government agency ask the keys for terrorism prevention (as

example). The complete encryption scheme for every message can be seen in fig 2.

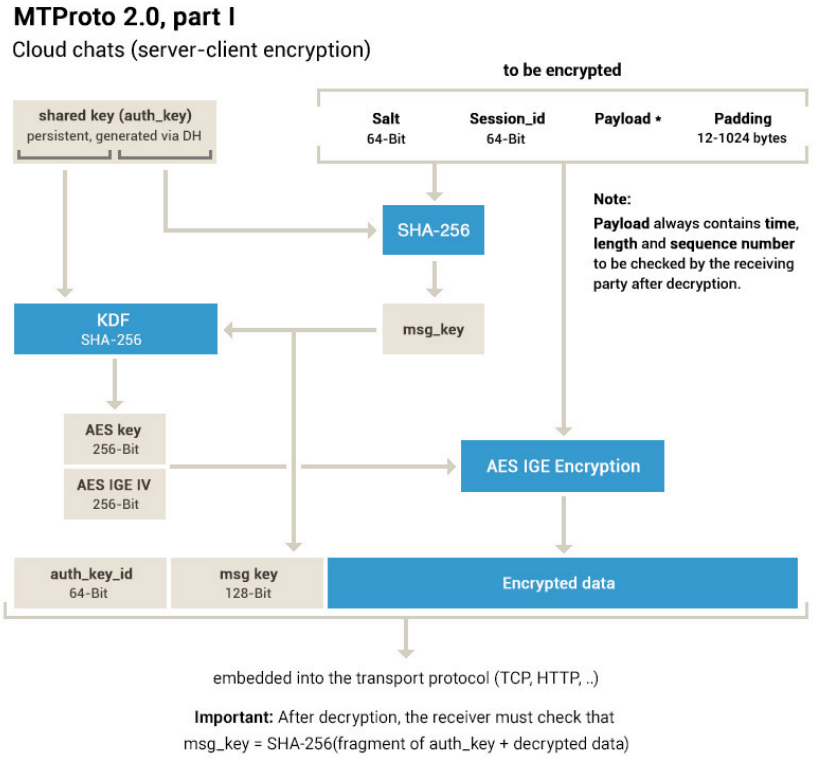

Figure 2: MTProto

The message identifier field is strictly related to time on the sender machine, and the receiver checks it with its time, so if there is a time divergence on sender or receiver message (here it is intended client/server communication and vice versa) then the receiver start to drop messages: in this case, the server (even if its time is not correct) will send a service message asking the client to synchronize its time with the server one. The time synchronization is simply defined as storing the difference between server and client time on client side. If the synchronization fails then the client must start a new session to continue the communication.
The transport layer encapsulated the encrypted message with a secondary protocol header depending on specification of the message contained in the header, this allows to obtain additional services such as *quick ack* or *transport error*. The effective transport then relies on existent transport layer protocols as TCP, websockets or HTTP/S.
This section has been developed using [1] and [6] as references.

3 WhatsApp Security Basics

WhatsApp security relies on **Signal** protocol: it allows end-to-end encryption of every communication through the application. The protocol is based on the following steps:

- Client registration and public keys exchange
- Encrypted session creation
- Message exchange using **Double Ratchet Algorithm** to generate a key for every message

When a new client is registered, some public keys are sent to the server who stores them: these keys are a *public identity key*, a *signed pre key* with its signature and a batch of *one time pre keys*. These elements allow the server to uniquely identify a user identity.
Every communication with a user is treated as an encrypted session: it is established at the first message exchanged and it does not change until no external factor changes the integrity of the session (this can happen due to a device change or an app reinstall). The session is created using the public keys of the receiver retrieved from the server and they are used to generate a master secret by "or" operation of different Diffie Hellman based on Elliptic Curves. The secret key is used by an hashed key derivation function to create a root key and some chain keys. If the recipient is not online at the time of the creation of the session, then the initiator includes the information to start the session with the first message sent, in order for the recipient to start the same encrypted session when he receives the messages.
Every message sent through a session are encrypted by a message key using AES256 and HMAC-SHA256 for authentication. The message key changes for every message and it is ephemeral, so it is not possible to reconstruct an encrypted message from the session state. The key generation is based on the **Double Ratchet Algorithm**: it is based on HMAC-SHA256 to generate the single message key from the ratchet chain key and on the association of ECDH and HKDF to generate the new chain key and root key for the next message (these steps guarantee forward secrecy). This mechanism can cause delay in the messages and shift on the order of the original messages, but since every message is encrypted separately, it is not a problem. The Double ratchet scheme can be seen in fig. 3; the letters A and B are the message sent by Alice or Bob, and the number is the number of the message sent by one of them.
The user can verify the end-to-end encryption keys to avoid a man-in-the-middle attack, the verification can be performed either by a QR code or by a 60 digit number, that the users must compare with the keys stored on their device to ensure no attack has been performed.
For what concerns the transport layer, WhatsApp uses the **Noise** protocol framework: this provides long time connections between the clients and the servers based on *noise pipes* with Curve25519, AES-GCM, and SHA256.

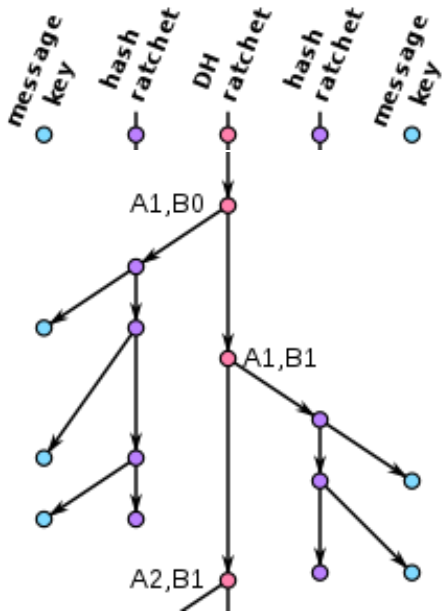

Figure 3: Double Ratchet [5]

Unfortunately, I was not able to find images like fig. 2 for WhatsApp; this is due to the nature of **Signal** protocol, which subdivides every step of security into separated applications, such as the Double Ratchet of fig. 3. This section is based on [3], [7], [4] and [2].

4 Protocols comparison

5 Past Evolution and Fixed Vulnerabilities

6 Current Vulnerabilities and Security Threats

7 Conclusion

After this brief introduction on elliptic curves, it is obvious why they have been widely adopted in many cases of asymmetric encryption: they use less bits for the same level of security, so are more efficient to compute and their base concepts are easier to visualize. As suggested on section , EC can be easily used in hybrid encryption scheme, in the key exchange phase.

References

[1] Mtproto mobile protocol. https://core.telegram.org/mtproto.

[2] Signal technical information. https://signal.org/docs/.

[3] Whatsapp encryption overview. https://scontent.whatsapp.net/v/t39.8562-34/122249142_469857720642275_2152527586907531259_n.pdf/WA_Security_WhitePaper.pdf?ccb=2&_nc_sid=2fbf2a&_nc_ohc=zyXCDnfHxJEAX_DdSo3&_nc_ht=scontent.whatsapp.net&oh=44e5aa8afd1092357784e47a197b5b32&oe=5FD7D019, 2020.

[4] Whatsapp wikipedia page. https://en.wikipedia.org/wiki/WhatsApp#Technical, 2020.

[5] P. kitty111. https://en.wikipedia.org/wiki/Double_Ratchet_Algorithm#/media/File:Axolotl_ratchet_scheme.svg.

[6] Multiple. Telegram wikipedia page. https://en.wikipedia.org/wiki/Telegram_(software), 2020.

[7] A. Panghal. Whatsapp's end to end encryption, how does it work? https://medium.com/@panghalamit/whatsapp-s-end-to-end-encryption-how-does-it-work-80020977caa0, 2018.