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.

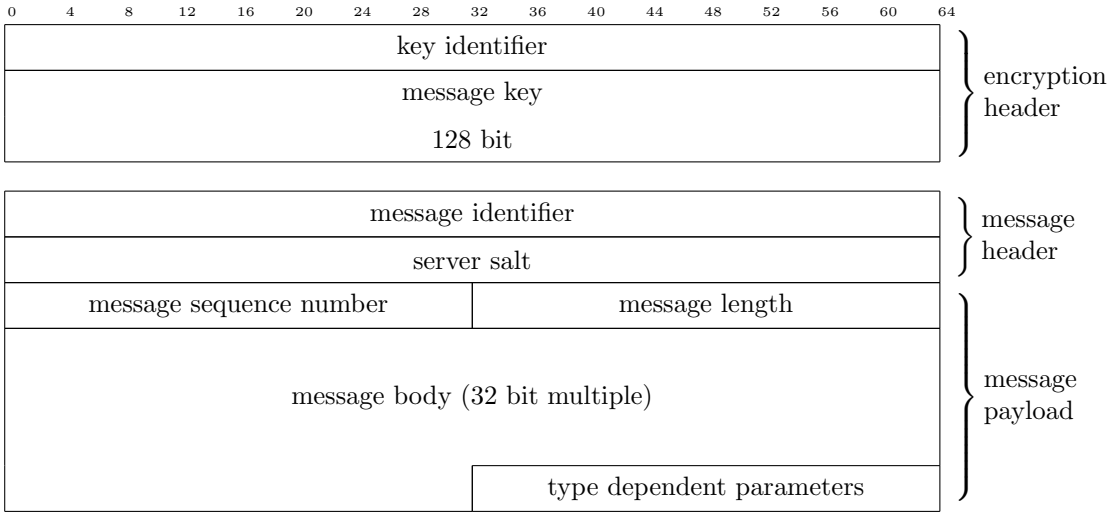

Figure 1: Telegram simplified packet structure

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.

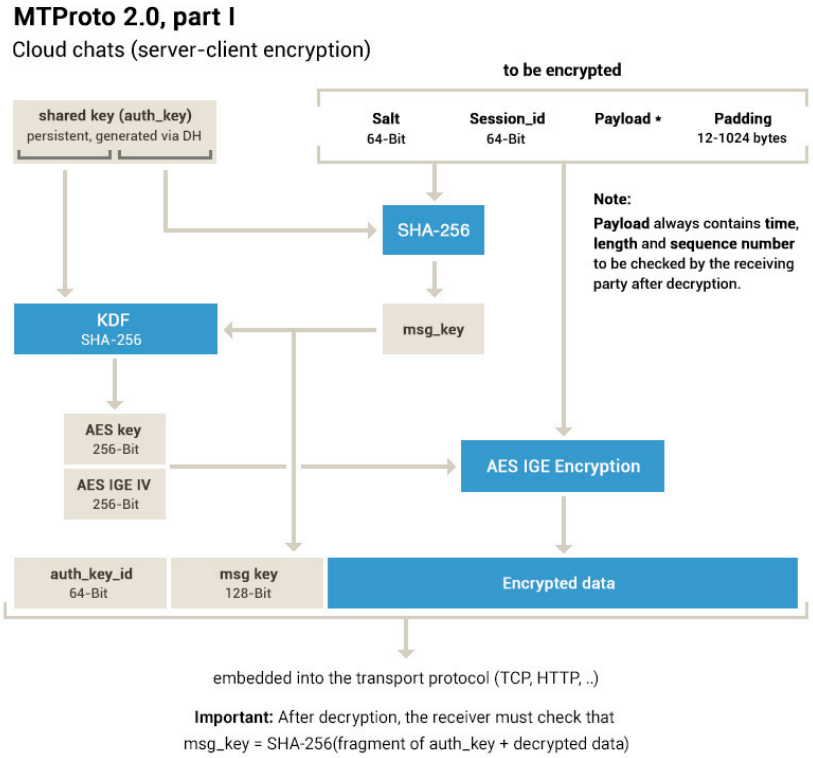

Figure 2: MTProto

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 [7] 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.

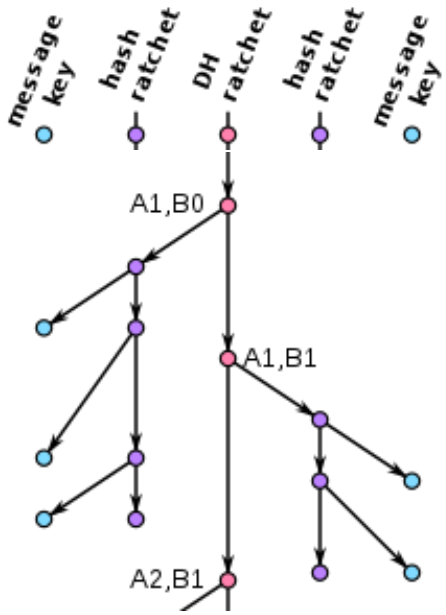

Figure 3: Double Ratchet [6]

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 [4], [9], [5] and [2].

4 Protocols comparison

This section contains some comparisons of both protocols, evidentiating some key aspect and vulnerabilities.
It can be easily seen that telegram hides some big hole in security: the first problem is its usability, to guarantee that, telegram uses session attached to the clients, so the session decades only when the client closes it; in addition one can enter a past session which is still open in a device just by going to the site and the cookies will automatically log in the user to that session. Furthermore, everything sent with telegram (messages, media, links and files) is all stored on server side, this will facilitate the user to obtain everything from every device where he is logged in, but if it is easier for the end user also someone interested in that data will gain access in easier way. Another problem is that all messages are encrypted with a key derived from the authorization key: this key rarely changes, and exposes all messages in the cases it is stolen from a device, this is due to the structure of the key derivation, from which it is possible to decrypt every message once it is stolen.

Furthermore, not all the chats are end-to-end encrypted, only secret chats have this priviledge, so a basic user does not know this difference and he generally uses the standard chats, so his message can be decrypted server side: this introduces another problem of telegram, the server keys are known to the telegram organization, and as it will be seen in the next sections, it is a big trouble for the end user. In addition to that, the server implementation is closed source, so the effectively security of server side cannot be verified by the user or others neutral entities.
Another problem is based on the time syncronization: if one of server or client have a different time, then the messages will began to be discarded, and since it is always the server who send the reference time for the syncronization, if a server is compromised, then an attacker can instantiate a DOS attack.
WhatsApp instead has better masked his security holes. This app uses end-to-end encryption for every communication, a problem stands in the initialization of the encryption: the first messages shared with a new chat/user contains the informations to start the encrypted session, so if the messages are intercepted by someone else other than the effective receiver, one can in some way listen to the session from that moment on, since the session keys changes only for rare events, such as app reinstall or device changes. An attacker can also become active if he has the session keys and inject false messages in the session. For what concerns the usability on browser, the opened session expires after some small time of inactivity.
In both cases the message key changes for every message, WhatsApp uses a more reliable message key generation, but if the session key is stolen then it is possible to rebuild every message key. If a message key is stolen, then only the associated message can be decrypted, but none of past and future message keys can be derived from another message key.
For the transport layer, Telegram uses a proprietary protocol on top of the standard ones, so transport security relies on the security of open transport layer protocols. WhatsApp instead relies completely on the Noise Protocol, which is open as in the other case of Telegram.
WhatsApp has both client and server closed source, so it has the same problem of Telegram when it comes about security verification by third party neutral entities.
The problem on both apps is that they are ruled by companies whose policies are not so secure: in one case the Telegram Messenger Inc has doubtly policies [3] and server keys are in their hand; on the contrary WhatsApp [**?**] is in the hand of Facebook, not one of the clearest companies in the world.
To recap, a small list of both protocols' problems:

- both have some pieces closed source, so no security verification
- both protocols have problems with the initialization of the session, if the session keys are stolen then all the messages can be decrypted
- telegram has also problem woth time syncronization and server side encryption (the latter only for not secret chat)

- both have problems with browser session, Telegram problem is a bit huge w.r.t. the WhatsApp counterpart

The comparison has taken in exam only the standard one-to-one chat, another analysis can comprehend also media, group chats and call/videocall security, which in the case of WhatsApp can be of big concerns [8].

5 Past Evolution and Fixed Vulnerabilities

The first app was WhatsApp, founded in 2009, at that time the app sent all the messages as plaintext, the first form of cryptography was added back in 2012

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] Telegram privacy policy. `https://telegram.org/privacy`.

[4] 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.

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

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

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

[8] K. O'Flaherty. Whatsapp security: Is this hidden flaw a new reason to quit? https://www.forbes.com/sites/kateoflahertyuk/2020/02/29/whatsapp-security-is-this-hidden-flaw-a-new-reason-to-leave/, 2020.

[9] 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.