A survey on the security protocols employed by mobile messaging applications.

Ștefania Andrieș [*] Andrei-Daniel Miron [†] Andrei Cristian [‡]
Emil Simion [§]

January 24, 2022

Abstract

Recently, there has been an increase in the popularity of messaging applications that use end-to-end encryption. Among them were Telegram (in October 2021 it has 550 million active users [1]), Signal (in January 2022 it has over 50 million downloads in the Google Play Store [2]), WhatsApp (according to Statista [1], in 2021 it has over 2 billion active users), Wire (until January 2022 it has been downloaded for over 1 million times on Android devices [3]). Two distinct protocols underlying these applications are noted: MTProto (developed in Russia by Nikolai Durov) and Signal (developed in the US by Moxie Marlinspike). This paper presents the two protocols and examines from the point of view of the primitive cryptographic security used and how the authenticated encryption, key derivation and asynchronous messaging are performed.
Keywords MTProto, Signal, End-to-End Encryption, Messaging apps.

[*]Faculty of Computer Science, Alexandru Ioan Cuza University of Iași
Email: stefania.andries21@gmail.com
[†]Faculty of Computer Science, Alexandru Ioan Cuza University of Iași
Email: danielandrei161@gmail.com
[‡]Faculty of Computer Science, Alexandru Ioan Cuza University of Iași
Email: andrecristian6@protonmail.com
[§]Politehnica University of Bucharest, Email: emil.simion@upb.ro

1 Introduction

Revelations about mass surveillance of communications have made consumers more privacy-aware. In response, scientists and developers have proposed techniques which can provide security for end users even if they do not fully trust the service providers. For example, the popular messaging service WhatsApp was unable to comply with Brazilian government demands for users' plaintext messages because of its end-to-end encryption [4].

Perhaps the first secure instant message protocol to achieve widespread adoption was Apple's iMessage [5], a proprietary protocol that provides end-to-end encryption. A notable characteristic of iMessage is that it automatically manages the distribution of users' long-term keys, and in particular (as of this writing) users have no interface for verifying friends' keys. iMessage, unfortunately, had a variety of flaws that seriously undermine its security[6].

Nowadays, two of the most used secure instant message protocols are Telegram Messenger Inc's **MTProto** [7] and Signal Foundation's **Signal** [8].

2 MTProto Protocol

MTProto is a set of cryptographic protocols, created by Telegram, designed for implementing fast, scalable and secure message exchange without relying on the security of the underlying transport protocol. It is formed from protocols that are tasked with the creation of shared keys between clients and server, the creation of session keys between two clients for end-to-end encryption in secret chats, the rekeying of secret chats and the encryption of all messages.[9]

MTProto provides two different encryption protocols for cloud chats and secret chats. As of december 2017, Telegram has started to phase out the MTProto 1.0 in favor of the MTProto 2.0, that brings important changes like using SHA-256 instead of SHA-1 and the 12..1024 padding bytes instead of the 0..15 bytes. [7]

2.1 Protocol Overview

Before a message (or a multipart message) is transmitted over a network using a transport protocol, it is encrypted in a certain way, and an external header is added at the top of the message that consists 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**. [7]

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).[7]

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. [7]

2.2 Cloud chat system

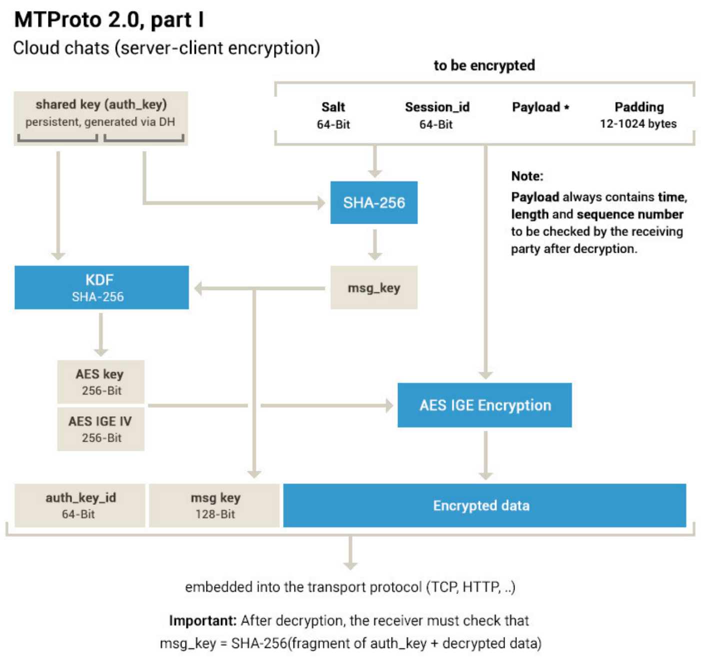
MTProto 2.0, part I
Cloud chats (server-client encryption)
to be encrypted
shared key (auth_key)
persistent, generated via DH
Salt
64-Bit
Session_id
64-Bit
Payload *
Padding
12-1024 bytes
SHA-256
Note:
Payload always contains time, length and sequence number to be checked by the receiving party after 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 1: MTProto Cloud Encryption (*Telegram Documentation* [7])

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.[7]

On the server, the messages are decrypted, to be encrypted again and sent to the receiver again, as presented in *Figure 2*.

2.3 Secret chats

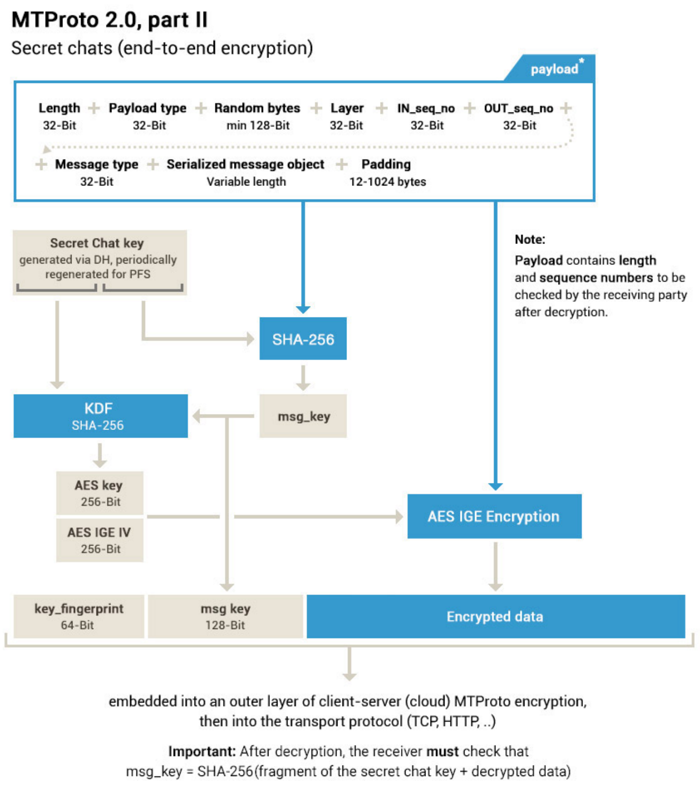
MTProto 2.0, part II
Secret chats (end-to-end encryption)
payload*
Length 32-Bit
Payload type 32-Bit
Random bytes min 128-Bit
Layer 32-Bit
IN_seq_no 32-Bit
OUT_seq_no 32-Bit
Message type 32-Bit
Serialized message object Variable length
Padding 12-1024 bytes
Secret Chat key generated via DH, periodically 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)

Figure 2: MTProto E2E Encryption (*Telegram Documentation* [7])

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. 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. [7]

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. [7]

2.4 Encryption

The data will be encrypted with a 256-bit key and a 256-bit IV, using AES-256 with infinite garble extension (IGE). The encryption key fingerprint and the message key, msg_key, will be added on top of the resulting byte array. The ciphertext will be computed as [10]:

$$c_i \leftarrow F_k(m_i \oplus c_{i-1}) \oplus m_{i-1}$$

where 1 ≤ i ≤ l and l represents the number of blocks from the payload of size B and F is a pseudorandom permutation. The final output will also include the msg_key:

$$c = (\text{msg\_key}, c_1, ..., c_l)$$

2.5 Decryption

The ciphertext c will be parsed as:

$$c = (\text{msg\_key}, c_1, ..., c_l)$$

The short-term key k and the initialization blocks c_i and m_i will obtained as:

$$(k, c_0, m_0) = \text{KDF}(K, \text{msg\_key})$$

The payload is recovered by computing:

$$c_i = m_{i-1} \oplus \text{F}_k^{-1}(m_1 \oplus c_{i-1})$$

and the $(c_1, ..., c_l)$ will be parsed into the components that were concatenated.

When an encrypted message is received, the msg_key will be checked to see if it is in fact equal to the 128 middle bits of the SHA-256 hash of the decrypted message, prepended by 32 bytes from the shared key. If this check fails, the message is dropped, and the sender is not notified of any error.[10] [11]

3 Signal Protocol

Developed in the 2013 in the Signal Technology Foundation non-profit organization (former Open Whisper Systems), Signal protocol provides End-to-End Encryption (E2EE) for many instant messaging applications. The core of the protocol consists in the combination of the Extended Triple Diffie-Hellman (X3DH) [12] key exchange protocol and the Double Ratchet algorithm [13]. Properties like future secrecy, post-compromise security or asynchronous communication are describing the Signal Protocol through the before mentioned core components.

3.1 Protocol Key Features

Signal Protocol offers end-to-end encrypted chats to the users and incorporates the well known security goals, **confidentiality**, **integrity** and **authenticity**. The protocol also provides other key features that should be presented before the building blocks to fully understand the what they accomplish. Several authors have highlighted some specific features of the Signal protocol with concrete security demonstrations. [14–19]

- **Future Secrecy** is a feature key that ensures via a key derivation function (KDF), the intruder incapability to decrypt any future messages using a compromised message key.
- **Forward Secrecy** is another feature key similar to the previous one that prevents an attacker from finding any past messages using also a compromised message key.
- **Message Unlinkability** is the trait that ensures that although a message has been associated with a user, no other message can be linked from that point. The messages are encrypted and authenticated with different one-time keys and therefore no proofs should be found to associate multiple messages to one party.
- **Offline Deniability** similarly with the previous property, a user can deny his involving in a conversation as long as the metadata are not saved or the keys are not compromised.
- **Asynchrony** a dominant trait for a messaging protocol, it allows users to initiate communications even when the receiver is not online. The asynchronous key agreement protocol X3DH enables parties to participate in the communication session at different times.

3.2 Building Blocks

The Signal protocol can be divided into three main blocks: the registration phase, the session setup and the ratchet steps, and each block will be presented below.

1. **Registration phase:** Each user has to run a device setup before engaging in any conversations using the protocol. This step will produce the prekeys used further in the X3DH protocol [12]. The registration is a single incipient phase or can be re-executed whenever the user wants. Therefore, each party must generate and transmit the following to the key server:
   - n one-time prekey pairs $\{(ospk^i_X, oppk^i_X)\}i$, for party X, where i is a unique id
   - A semi ephemeral key pair $\{(sspk^j_X, sppk^j_X)\}j$, for party X, where j is a unique id

   - A long-term identity key pair (isk_X, ipk_X) for party X.

   Every time a user, say Alice, wants to initiate a new communication session with another party, Bob, she has to request his key-bundle from the server key. The so called key-bundle associated with Bob is computed from the previous keys that Bob sent on the registration and contains the following:

   - The user's registration id (id_X)
   - A public one-time key preceded by a unique id $(i, oppk_X^i)$. The one-time keys are deleted by the server upon transmission.
   - The semi-ephemeral public key signed with the long-term identity key and also identified by a unique id $(j, sppk_X^j, sig_X^j)$, where sig_X^j ¸$sign(isk_X, sppk_X^j)$
   - the public long-term identity key (ipk_X)

2. **Session setup:** The second main block of Signal Protocol sets the cryptographic keys for the symmetric encryption scheme used in the upcoming conversation session. The Extended Triple Diffie-Hellman key agreement protocol, X3DH, is used in this scope, and the resulting shared secret will be used on the next component. Once Alice received the bundle-key associated with Bob, she can start calculating three or four keys applying Diffie-Hellman on combinations of her new generated base key and Bob's keys from server.

$$(bsk_A, bpk_A) \leftarrow gen()$$

$$k1 \leftarrow DH(isk_A, sppk_B^j)$$

$$k2 \leftarrow DH(bsk_A, ipk_B)$$

$$k3 \leftarrow DH(bsk_A, sppk_B^j)$$

$$k4 \leftarrow DH(bsk_A, oppk_B^i)$$

   The last key, $k4$, is optional as the Bob's one-time key can be missing from the key-bundle if the server used and deleted all one-time keys generated from the registration phase. All the results from the Diffie-Hellman calculations are concatenated in one master key which is used to open a session with Bob.

$$sk \leftarrow KDF(k1||k2||k3(||K4))$$

   Bob will receive from Alice in the first messages her base key, her identity and an identifier of the Bob's prekeys she used and then Bob will be able also to compute the master secret.

$$k_1 \leftarrow DH(sspk_B^j, ispk_A)$$

$$k_2 \leftarrow DH(isk_B, bpk_A)$$

$$k_3 \leftarrow DH(sspk_B^j, bpk_A)$$

$$k_4 \leftarrow DH(ospk_B^i, bpk_A)$$

3. **Ratchet steps:** Since the X3DH key agreement protocol was run, a secret shared key was calculated. Any encrypted message based on it can be exchanged between the parties using the Double Ratchet algorithm[13]. This component ensures that each party generates new keys at every step, providing Future Secrecy and Forward Secrecy properties. The Double Ratchet algorithm involves three Key Derivation Functions (KDF) [20] chains for each user: a root chain, a sending chain, and a receiving chain. There are two ratchet stages in which the chains are advanced: the symmetrical Ratchet and the asymmetrical Ratchet.

    - Asymmetrical Ratchet: the root chain refreshes the other ones. The output is a new chain key and the message key

      $$(rk, ck) \leftarrow KDF(rk, k_n)$$

      The first ratchet will take the secret shared key as input for this KDF and the next ones will take the second part of the output of the previous ratchet. k_n is generated on each party using Diffie-Hellman on an ephemeral key pair generated by Alice and Bob's signed prekey:

      $$(esk_A, epk_A) \leftarrow gen()$$

      $$k_n \leftarrow DH(esk_A, sppk_B)$$
      $$k_n \leftarrow DH(sppk_B, epk_A)$$

      When asymmetric ratchets take place, the current symmetric key chains are potentially discarded if there are not messages in transit needed for decryption. New sending and receiving chain pair are created, providing freshness and future secrecy.

    - Symmetrical Ratchet: To ratchet the sending and receiving chains, a KDF will be run on the first part of the output of the previous execution. The second part of the output will be used as a key for a new message encryption/decryption operation.

      $$(ck, mk) \leftarrow KDF(ck)$$

      This step allows the users to discard symmetric keys after using them to encrypt or decrypt a message and so a possible adversary that compromise a key is prevented from decrypting past messages.

3.3 Protocol Usage

Signal Protocol is an open-source implementation that is present in many chat applications, many of which are used in daily basis. Through the most popular instant messaging applications that use Signal Protocol for End-to-End Encryption we can mention WhatsApp, Facebook Messenger and without a doubt Signal itself.

Signal Technology Foundation launched the Signal platform for instant messages in 2014 and offers a free, open source software that covers all messages with End-to-End Encryption. The source code is in a public GitHub repository [21] to allow any curious party to analyse, test and improve the security and the correctness.

WhatsApp integrate the Signal protocol in its implementation on 2016 in order to provide encrypted messages exchange and all features mentioned in section 3.1. WhatsApp published a technical white paper [22] to acknowledge the integration and explains how and where the protocol is used.

Facebook adopted Signal Protocol in 2016 for Messenger to introduce the Secret Conversations feature. The technical white paper [24] recognises this new integration. Compared to Signal App and WhatsApp which offer end-to-end encryption for all sessions by default, Messenger did not enabled this feature on each conversation. However Facebook announced that wants to implement Secret Conversation as default.

4 Comparison of Signal and MTProto

As our interest is in the comparision of the end-to-end encryption, in this section we will refer MTProto as MTProto's secret chats, if not explicitly specified otherwise.

4.1 Cryptographic Primitives

Both Signal and MTProto use 256-bit AES encryption [26], SHA256 . Signal uses one of two elliptic curves to implement X3DH: curve X25519 or curve X448 [27], while MTProto uses a 2048-bit RSA key for DH.

4.2 Forward Secrecy

Signal renews the keys used for message encryption after Alice receives a message from Bob, while MTProto (actually 'official Telegram clients') will initiate re-keying once a key has been used to decrypt and encrypt more than 100 messages, or has been in use for more than one week, provided the key has been used to encrypt at least one message[28].

In Signal compromising the current key will only compromise a set of messages until the the next message from the other party arrives - possible 0. [18] In Telegram (MTProto), less than 100 messages can be recovered using the compromised key.

4.3 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.
- 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 Signal this is not possible, because all data are only available on the unique client (and the linked devices).

5 Conclusions

Signal and MTProto protocols are the underlying substrate in some of the most used messaging applications in the world right now.

For key derivation, Signal has a good story to tell: compromising the current key will only compromise a set of future messages (possibly zero; until the next message from the other party arrives). However, in MTProto, compromising the authorization key (which is the closest equivalent key) allows all past and future messages to be compromised.

Speaking of the applications, Signal, which is owned by **Signal Technology Foundation** an American non-profit organization, keeps the messages on your device, while Telegram, owned by **Telegram Messenger Inc** which was founded by two Russian brothers, by using it's default cloud-messaging mode keeps your encrypted messages in the cloud, accessible for the people having the decryption keys.

References

[1] *Most popular social networks worldwide as of October 2021, ranked by number of active users.* URL: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/.

[2] URL: https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms.

[3] URL: https://play.google.com/store/apps/details?id=com.wire.

[4] Marcelo Santos and Antoine Faure. "Affordance is power: Contradictions between communicational and technical dimensions of WhatsApp's end-to-end encryption". In: *Social Media+ Society* 4.3 (2018), p. 2056305118795876.

[5] Scott E Coull and Kevin P Dyer. "Traffic analysis of encrypted messaging services: Apple imessage and beyond". In: *ACM SIGCOMM Computer Communication Review* 44.5 (2014), pp. 5–11.

[6] Xiaoyu Shi. 2016. URL: https://www.cs.tufts.edu/comp/116/archive/fall2016/xshi.pdf.

[7] *Telegram MTProto Documentation.* URL: https://core.telegram.org/mtproto.

[8] *Signal Website.* URL: https://signal.org/docs/.

[9] Marino Miculan and Nicola Vitacolonna. "Automated Symbolic Verification of Telegram's MTProto 2.0". In: (2021).

[10] Sungsook Kim Jeeun Lee Rakyong Choi and Kwangjo Kim. "Security Analysis of End-to-End Encryption in Telegram". In: (2017).

[11] Jakob Jakobsen and Claudio Orlandi. "On the CCA (in)security of MTProto". In: (2015).

[12] Moxie Marlinspike Trevor Perrin. "The X3DH Key Agreement Protocol". In: (2016). DOI: https://signal.org/docs/specifications/x3dh/x3dh.pdf.

[13] Moxie Marlinspike Trevor Perrin. "The Double Ratchet Algorithm". In: (2016). DOI: https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[14] Michael Schliep and Nicholas Hopper. "End-to-end secure mobile group messaging with conversation integrity and deniability". In: *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society.* 2019, pp. 55–73.

[15] Sara Stadler et al. "Hybrid Signal protocol for post-quantum email encryption". In: *Cryptology ePrint Archive* (2021).

[16] Matthew L Jansen. "A Security Analysis of the Signal Protocol's Group Messaging Capabilities in Comparison to Direct Messaging". In: (2020).

[17] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. "The double ratchet: security notions, proofs, and modularization for the signal protocol". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 129–158.

[18] Katriel Cohn-Gordon et al. "A formal security analysis of the signal messaging protocol". In: *Journal of Cryptology* 33.4 (2020), pp. 1914–1983.

[19] Paul Rösler, Christian Mainka, and Jörg Schwenk. "More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema". In: *2018 IEEE European Symposium on Security and Privacy (EuroS P)*. 2018, pp. 415–429. DOI: 10.1109/EuroSP.2018.00036.

[20] Hugo Krawczyk. *Cryptographic Extraction and Key Derivation: The HKDF Scheme*. Cryptology ePrint Archive, Report 2010/264. https://ia.cr/2010/264. 2010.

[21] *Signal open-source Repository*. URL: https://github.com/signalapp.

[22] WhatsApp. *WhatsApp Encryption Overview Technical white paper*. URL: https://scontent.whatsapp.net/v/t39.8562-34/271639644_1080641699441889_2201546141855802968_n.pdf/WhatsApp_Security_Whitepaper.pdf?ccb=1-5&_nc_sid=2fbf2a&_nc_ohc=XagoRV_7TsgAX92_ZMT&_nc_ht=scontent.whatsapp.net&oh=01_AVw83188FFhEpYclFBdMZCwrZn_no8eF_TAjEPOLVxhjEA&oe=61E46CBE.

[23] Tom Carpay and Pavlos Lontorfos. "WhatsApp End-to-End Encryption:Are Our Messages Private?" In: (2019). DOI: https://www.os3.nl/_media/2018-2019/courses/rp1/p25_report.pdf.

[24] Facebook Inch. *Messenger Secret Conversations Technical Whitepaper*. URL: https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf.

[25] *Facebook Messenger deploys Signal Protocol for end-to-end encryption*. URL: https://signal.org/blog/facebook-messenger/.

[26] Vincent Rijmen and Joan Daemen. "Advanced encryption standard". In: *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology* 19 (2001), p. 22.

[27] Adam Langley, Mike Hamburg, and Sean Turner. "Elliptic curves for security". In: *Internet Engineering Task Force* (2016).

[28] URL: https://core.telegram.org/api/end-to-end#perfect-forward-secrecy.