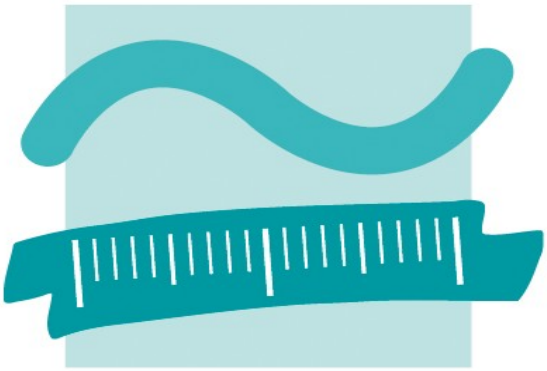


BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences

# End To End Encryption

## Analyze and comparison between MtProto and Signal protocol

**Supervised by M.Eng. Tarkan Yavas**

**By**
**Mourad Fredj: 868632**
**&**
**Houssem Mhadhbi: 870918**

# 1-Introduction

End-to-end encryption (E2EE) is a method of secure communication that prevents third-parties from accessing data while it's transferred from one end system or device to another.

In E2EE, the data is encrypted on the sender's system or device and only the recipient is able to decrypt it. Nobody in between,  Internet service provider, application service provider or hacker, can read it or tamper with it.

# 2 – E2EE

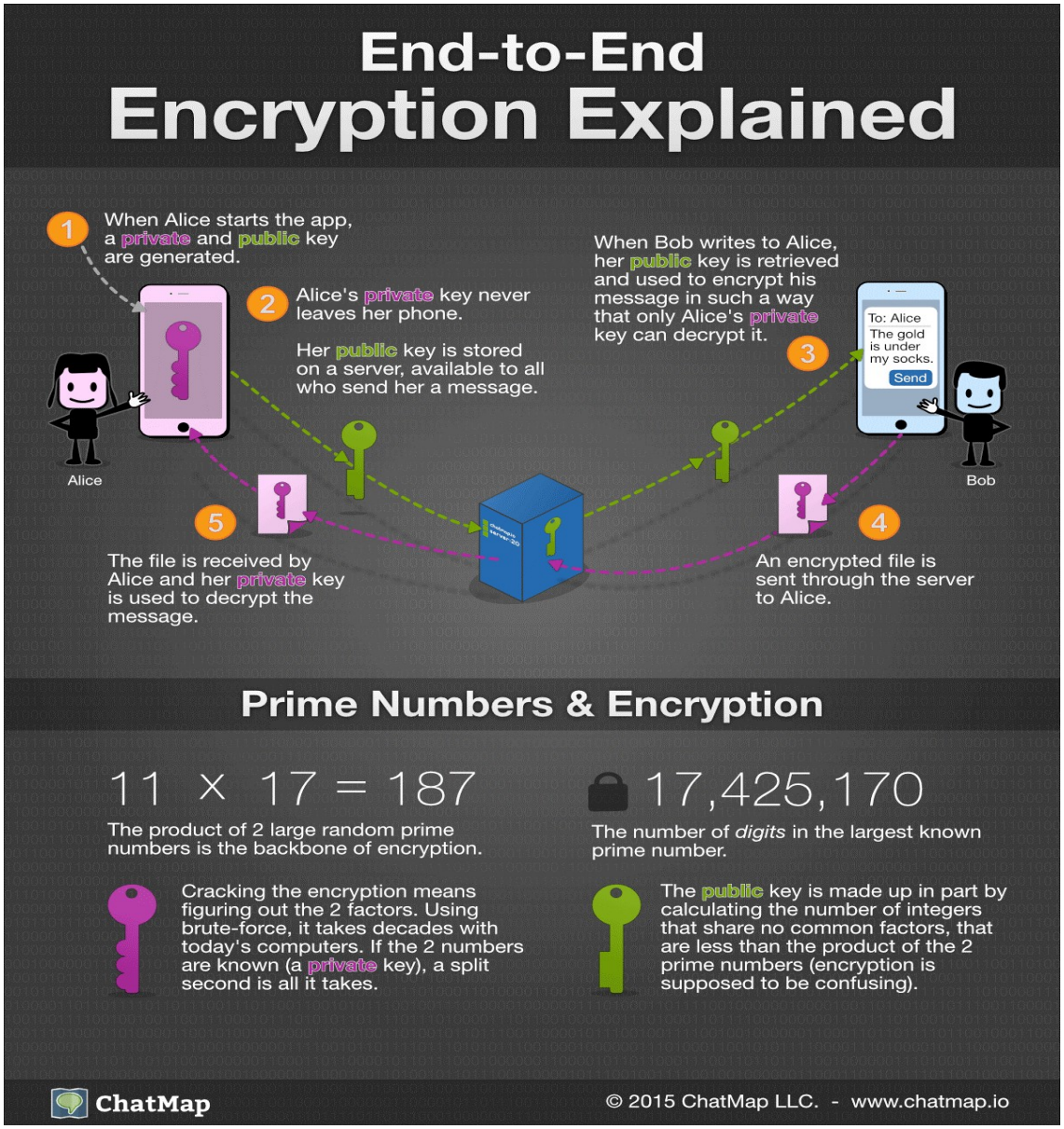



**Fig1. E2EE Encryption**

The cryptographic keys used to encrypt and decrypt the messages are stored exclusively on the endpoints, a trick made possible through the use of public key encryption. Although the key exchange in this scenario is considered unbreakable using known algorithm (**Diffie-Hellman**) and currently obtainable computing power, there are at least two potential weaknesses that exist outside of the mathematics. First, each endpoint must obtain the public key of the other endpoint, but a would-be an attacker who could provide one or both endpoints with the attacker's public key could execute a man-in-the-middle attack. Additionally, all bets are off if either endpoint has been compromised such that the attacker can see messages before and after they have been encrypted or decrypted.

## 3-Diffie-Hellman

Diffie-Hellman is an algorithm which is used to establish a shared secret between two parties.

It is a method of secure exchanging of cryptographic keys over a public channel which is used in End-to-End encryption

To start the communication Alice and Bob agree publicly to use a modulo **p** (**p** is prime) and a base **g** where **g** is primitive root modulo **p**.

2 Public keys are generated (**A**, **B**) where:

**A= $g^a$ mod p**, **B=$g^b$ mod p** and they are known to anyone

a= Alice`s private key, known only to Alice

b= Bobs private key, known only to Bob.

Alice and Bob change their Public Keys **A** and **B**

Alice and Bob know now each others public key and can then calculate the shared secret key as:

**S=$B^a$ mod p = $A^b$ mod p** and use it to open the message.

Example:

p=17, g=5

Alice chooses a secret integer a=2 then sends Bob A= $5^2$ mod 17= 8

Bob chooses a secret integer b=3 and sends Alice B= $5^3$ mod 17= 6

Alice computes S= $B^a$ mod p= $6^2$ mod 17= 2

Bob computes S= $A^b$ mod p= $8^3$ mod 17= 2

(we chosen small numbers just for the explanation but in the practice big prime numbers are chosen, generally 2048bits prime numbers).

# 4 – Application

## 4.1 MT Protocol (Telegram)

### 4.1.1 Payload encryption process:

MTProto uses Diffie-Hellman (DH) key exchange, Secure Hash Algorithm (SHA-256), Key Derivation Function (KDF), and AES-256 in IGE mode as cryptographic primitives and the overall process is described in Figure 2.

**SHA-256:** After its generation the shared secret key **s** must be hashed with secure hash algorithm SHA-256 to ensure its integrity by comparing the hash value of the shared secret keys.

The SHA (Secure Hash Algorithm) is one of a number of cryptographic hash functions.

A cryptographic hash is like a signature for a data set. If you would like to compare two sets of raw data (source of the file, text or similar) it is always better to hash it and compare SHA256 values.

**Msg_key:** Msg_key is defined as 128 middle bits of the SHA-256 of the message body prepended by 32 bits from the authorization key.

**KDF (key derivation function):** used generally to drive one or more secret keys from a secret value or to stretch keys into a longer keys. In this case KDF is used to obtain a key of required format by converting the result of Diffie-Hellman key exchange into a symmetric key for use with AES ( AES IGE ).

The encryption will be executed with AES.

**AES Key (Advanced Encryption Standard):**

Block cypher like AES is one of the most important cryptographic standard and widely used to encrypt data, its operation work on a fixed- length group of bits called a block, there can be many approaches to combine repeated operations for multiple block.

### 4.1.2 Key generation:

The DH key exchange is used for generating an ephemeral key. After key exchange, the sender and the receiver share the same 2048-bit symmetric key K. In order to protect past communications, secret key is regenerated once a key has been used for more than 100 messages or more than a week. The payload x is generated by concatenating some auxiliary information, random bytes, message, and padding such that $|x| \bmod B = 0$ where B is the block length. Then the payload except padding is computed by hash function SHA-256 whose output named tag. This tag is hashed again by KDF for generating AES key and IV. The input of KDF is (K, tag) and the output is (k, c0, m0) of the length (k, B, B) where $\kappa = 256$ bits and $B = 128$ bits.

### 4.1.3 Encryption:

The AES-256 in IGE mode is used for encryption. Let x1,...,xl be the l blocks of the payload, each of length B, then ciphertext is computed as below: $ci \leftarrow Fk(mi \oplus ci-1) \oplus mi-1$ where F is a pseudo random permutation, e.g., AES. The final output of the encryption is c including other information.

c = (tag, c1,...,cl)

### 4.1.4 Decryption:

Given cipher-text c, tag is used again in KDF. Using (tag, K), KDF output is (k, c0, m0) same as encryption. Also, the IGE mode is used again for decryption and the payload x is recovered.

$m_i \leftarrow F^{-1}_k(c_i \oplus m_{i-1}) \oplus c_{i-1}$

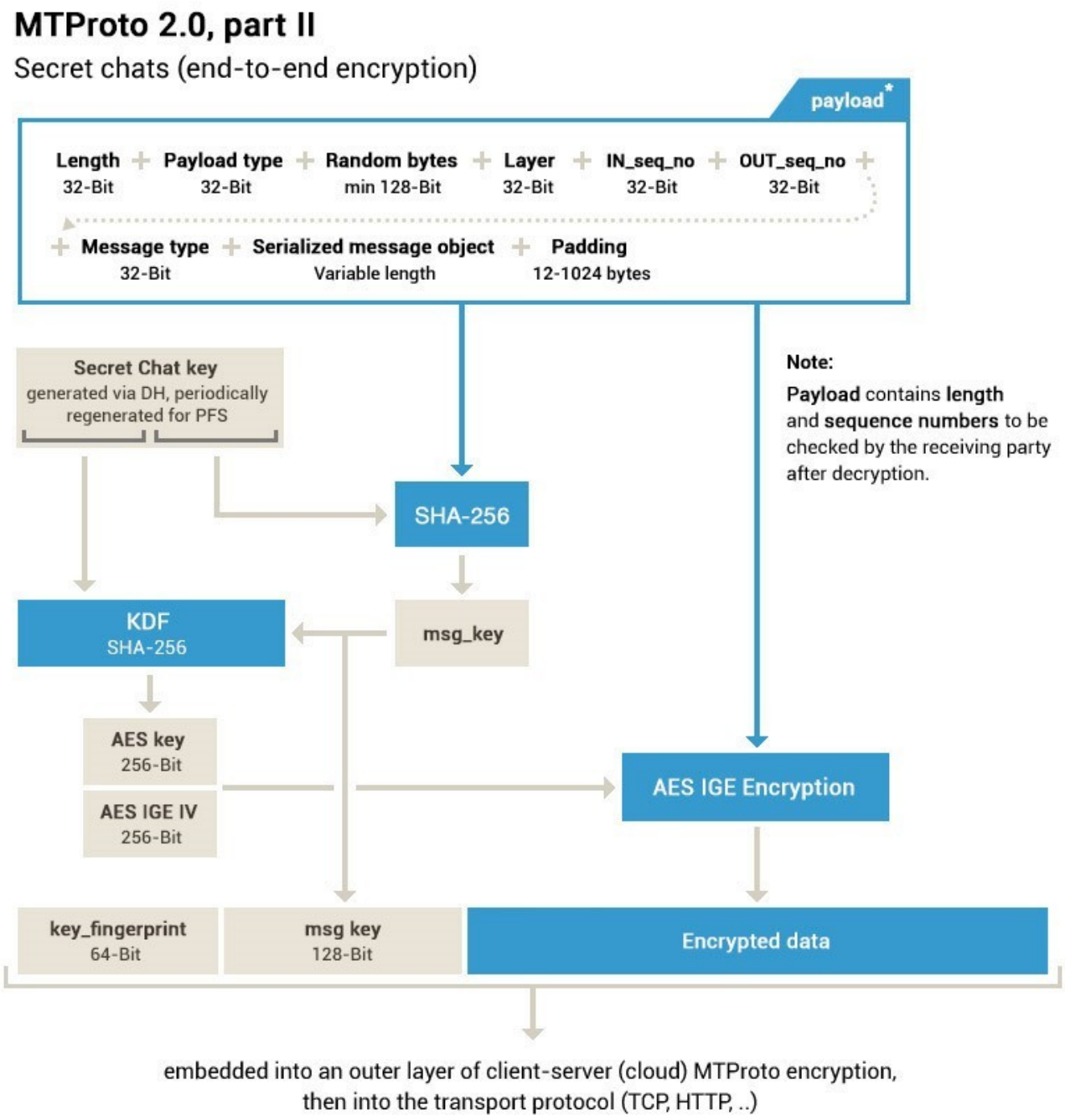



**Fig2. Payload of MTProto in E2EE**

Notice: After decryption, the receiver must check that the msg_key=SHA-256(fragment of the secret chat key + decrypted data).

### 4.1.5 Known Attacks on MTProto:

**IND CCA (INDistinguishability under adaptive Chosen Ciphertext Attack)**

for a probabilistic polynomial-time adversary ,A(Attacker) win always the folowing game:

1. A outputs different messages $M_0$ and $M_1$ of the same length

2. C (cryptographer) chooses b C {0,1} randomly and outputs the cyphertext $C_b \leftarrow E(M_b)$

3. A appends 128 Bits random block $c_r$ to $C_b$ and ask C to decrypt C` = $C_b$ || $c_r$

4. C returns M´ where M´ =$M_b$ for any b

5. A guess b as 0 if M´ = $M_0$, 1 otherwise.

This attack is possible since extra padding on ciphertext yields only the extension of the padding of plaintext without changing the plaintext. Obviously, A gets a ciphertext C´ = E($M_b$) for $M_b$ and this ciphertext has not been previously produced by the sender because of the random padding.

**Notice:**

A probabilistic polynomial-time algorithm is a probabilistic algorithm that may only perform a polynomial amount of operations including at most a polynomial number of coin-flips.

To be secure against this attack, it is necessary to check the length of padding in M0 by modifying the decryption process. Decryption algorithm will discard the message if the length of padding is larger than the block size.

**Last Block substitution Attack**

Since the padding is not authenticated in the process of MTProto, it is possible to make a collision with a non-negligible probability by modifying the last 128-bit (16-byte) blocks.

For a probabilistic polynomial-time adversary, A wins the following game with a probability at most $2^{-8}$, i.e., MTProto is not INT-CTXT (integrity of ciphertext)secure under the following game.

1. A outputs a message M whose length in bytes is equal to b mod 16.

2. The challenger C hashes M into the message key msg key ← SHA-1(M) to provide integrity of the plaintext.

3. Before encryption, 16−b random bytes of padding r are added to M, then sends C = E(M||r).

4. A modifies last 16-byte blocks of C` to get C` =! C.

5. A outputs C`.

To make the protocol secure against this attack, it is sufficient to add padding to the computation of the authentication tag. But, since this requires to change the whole encryption with authentication process, it becomes impossible to communicate with the older versions of the protocol due to version compatibility.

## 4.2 Signal Protocol (WhatsApp)

### 4.2.1 introduction

The signal protocol is cryptographic messaging protocol that provide end to end encryption for instant messaging in WhatsApp, Wire and Facebook messenger. The protocol was developed by open whisper system in 2013.

Signal provides confidentiality, integrity, authentication and other uncommon security proprieties such as future secrecy and post compromise security that are enable due to a new technology called **ratcheting,** in which session keys are updated with every message sent.

The encryption process of the signal protocol is divided into 2 steps:

- X3DH (extended triple Diffie-Hellman) which is used for key exchange

- Ratcheting to generate forward secret keys and to create symmetric encryption key

#### 4.2.2 X3DH (extended triple Diffie-Hellman)

X3DH establishes a shared secret key between two parties who authenticate each other based on public keys. X3DH assure forward secrecy and cryptographic deniability.

X3DH is designed to provide an asynchronous communication session between 2 users.

User Bob is offline but published some information to a server, Alice use this information to establish a secure communication with Bob.

X3DH uses some defined keys called elliptic curve public keys:

$IK_A$: Alice´s identity key

$EK_A$: Alice´s ephemeral key

$IK_B$: Bob identity key

$SPK_B$: Bob´s signed prekey

$OPK_B$: Bob´s one time prekey

**X3DH process:**

Bob have to upload a set of keys on a server which are the elliptic curve public keys

$IK_B$, $Spk_B$, $Sig(IK_B, Encode(SPk_B))$

**notice**: sig is signtaure for the identity key and signed prekey

Encode is a function to encode $SPK_B$ into byte sequence

$SPK_B$ and the prekey signature must be uploaded at some time interval (once a week or once a month).

$OPK_B$ may be also at other time published

**X3DH key agreement:**

to perform X3DH key agreement Alice should communicate the server and look up for Bob´s set of keys.

2 scenario are possible that depends on Bob´s $OPK_B$.

If $OPK_B$ dos not exist within set of keys that are published by Bob on the server, Alice calculate the Diffie-Helleman key as following:

DH1 = DH ($IK_A$, $SPK_B$)

DH2 = DH ($EK_A$, $IK_B$)

DH3 = DH ($EK_A$, $SPK_B$)

The secret key SK will be result of KDF (key derivation function) of the 3 concatenated DH.

**SK = KDF (DH1 || DH2 || DH3)**

if the $OPK_B$ exists within the published set of keys SK the calculation will include the Fourth key DH4:

DH4 = DH($EK_A$, $OPK_B$)

**SK = KDF (DH1|| DH2|| DH3|| DH4).**

Alice send now a ciphertext to Bob (message post X3DH) that contains several information:

$IK_A$, $EK_A$, identifiers stating which of Bob´s prekeys Alice used.

After receiving Alice´s initial message Bob calculate the DH and SK (using his loaded private keys IKB and SPKB). After the derivation of SK Bob must delete the DH.

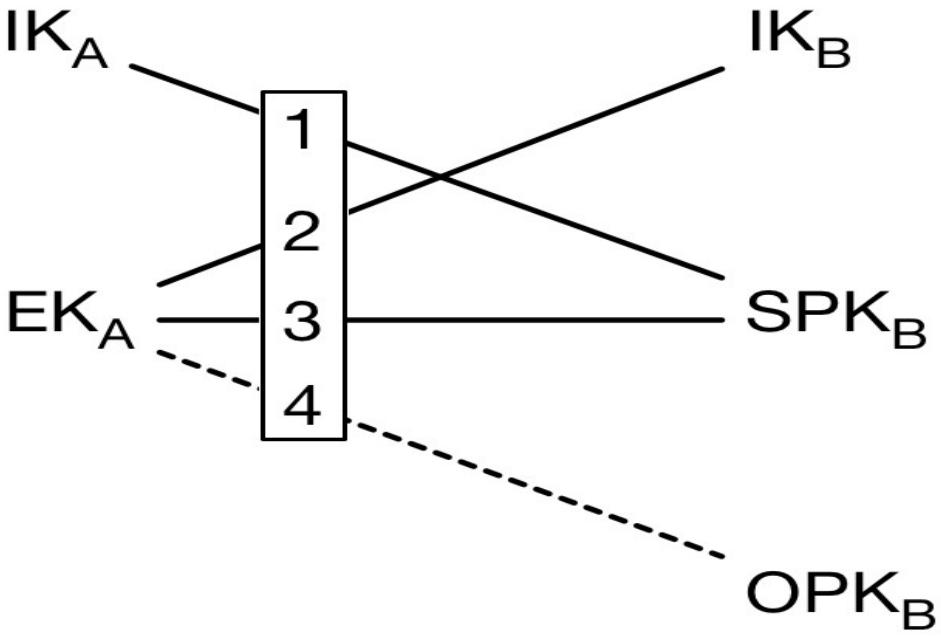
Bob try now to decrypt the initial text using AD and SK. If the initial ciphertext fails to decrypt the X3DH will be aborted and SK will be deleted.

**Notice:**

AD (associated data is calculated by Alice before sending the ciphertext to Bob and it contain identity information for both users $IK_B$, $IK_A$. AD is actually the concatenating of sequences of byte of $IK_A$ and IKB

**AD= Encode ((Ika)|| (Ikb)).** AD must be by Bob calculated also.

If the cyphertext is successfully decrypted the protocol is executed, Bob delete now any OPK to assure forward secrecy.




**X3DH key exchange**

### 4.2.3 Ratcheting

**introduction**

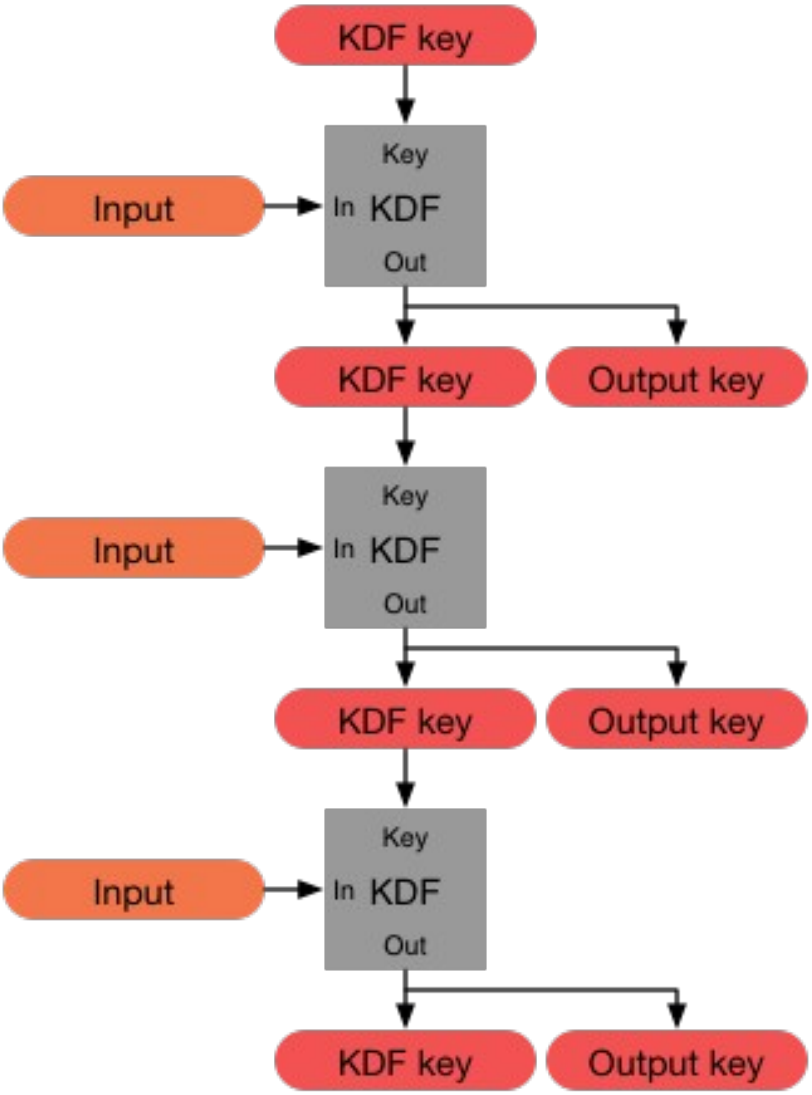
Based on the double ratchet algorithm, which is used by two parties to exchange encrypted messages. The two parties send and receive encrypted messages after sharing the secret key.

News keys are derived for every double ratchet messages so that the new one can not be calculated form the previous one.

#### 4.2.3.1 KDF chains

A KDF (key derivation function, explained above within MTProto) is a main concept in the Double Ratchet algorithm, it has an input KDF key and another input and generates two new keys, an output key and a KDF key, which can then be used with another input for the next key derivation and so on.

This processing can be represented in the below diagram:




In double ratchet session between Alice and Bob each party stores a KDF key for the 3 chains:

root chain, sending chain and receiving chain. Alice´s sending chain matches Bob´s receiving chain and vice versa.

The double Ratchet is a combination between two algorithms called the Diffie-Hellman ratchet and the symmetric-key ratchet.

#### 4.2.3.2 Symmetric-key ratchet

Every sent or received message is encrypted with a unique message key. As explained above, Alice and Bob have each a sending and a receiving chain. Lets say Alice wants to send a message to Bob so she has to start a new sending chain. She receives then a message key, crypts the message and sends it to Bob and deletes the key finally. Bob starts a receiving chain doing the same steps as

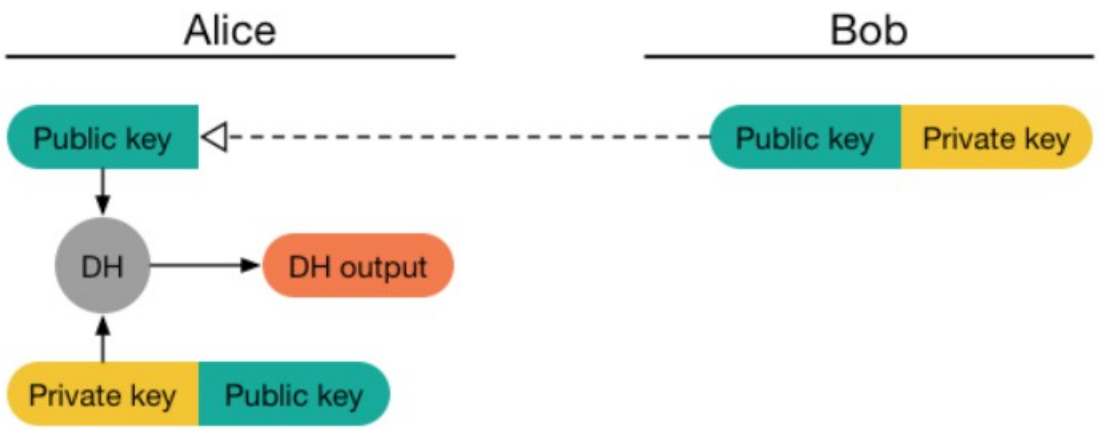
Alice and gets then the same message key, decrypt the message and finally deletes the key. This procedure assure a forward secrecy, that all the old messages stay secure, even if after a successful attack. The problem here is that if an attacker steals the KDF key, will be able to decrypt all the future messages. To prevent this, the symmetric-key ratchet is combined with a Diffie-Hellmann Ratchet which updates chain keys based on Diffie-Hellman outputs.

#### 4.2.3.3 Diffie-Hellman Ratchet

Due this algorithm, the keys pair of the sending and receiving chains will be changed regularly with new keys pair.

Each party of the communication generates a DH key pair composed of a Diffie-Hellman public key and private key, this pairs are the current ratchet key pairs. Every message from either party begins with a header which contains the sender´s current ratchet public key. When a new ratchet public key is received from the remote party, a DH ratchet step is performed which replace the local party´s current ratchet key pair with a new key pair.

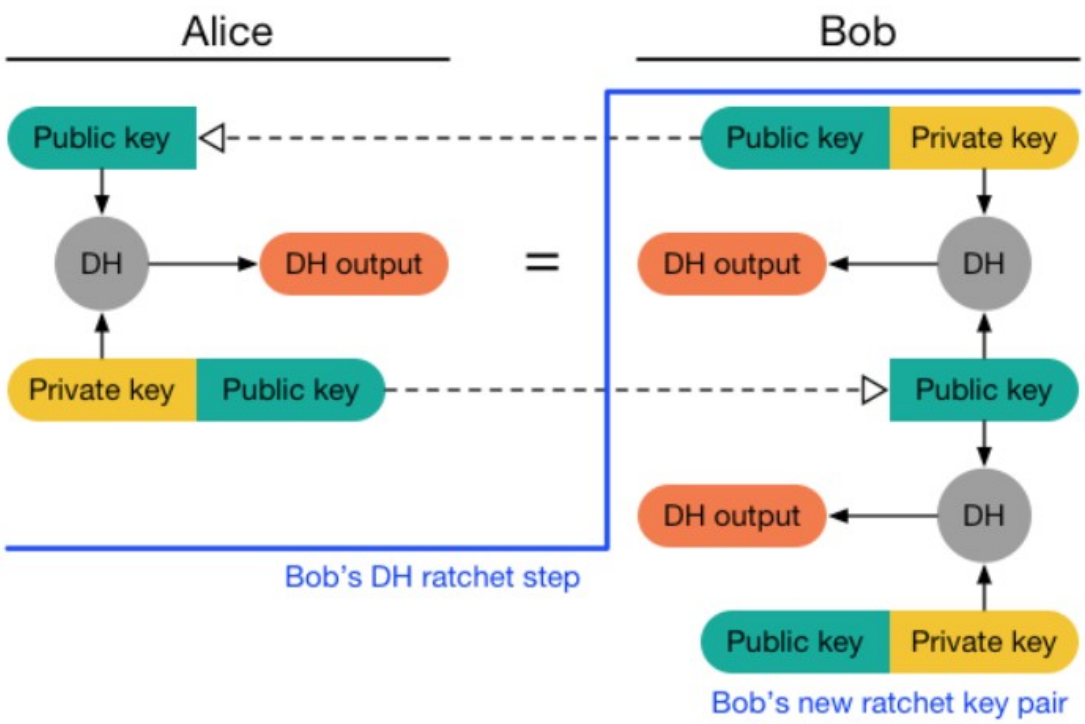
The following diagram describes how the DH ratchet steps are executed.



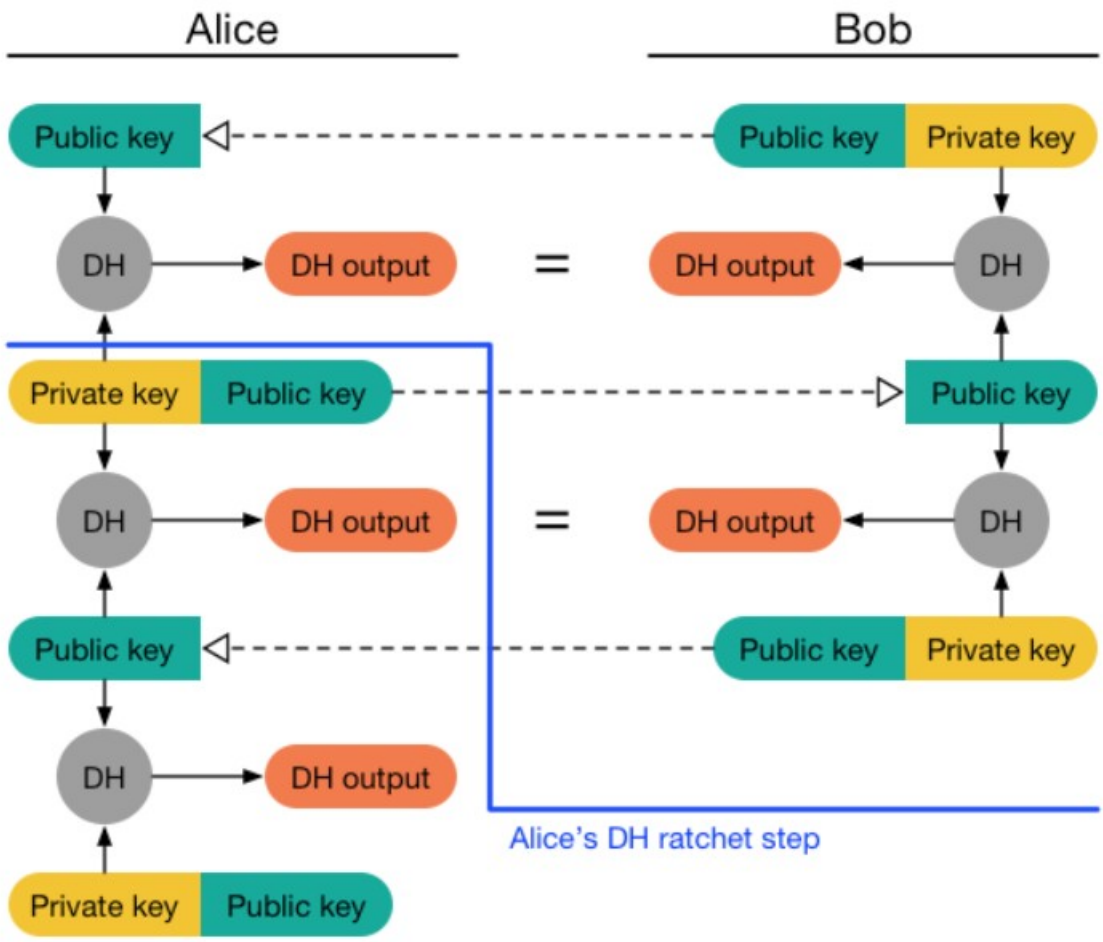

To start the communication Alice received Bob´s ratchet public key, Alice calculates then the first

DH output using Bob´s ratchet public key and her ratchet private key.

Now is Bob´s turn to perform DH calculation when receiving Alice´s initial message which include her public ratchet key. Using his private key Bob´s is able to calculate his first DH output which is equal to Alice´s DH output. Bob replace then his ratchet key pair and calculate a new DH output.

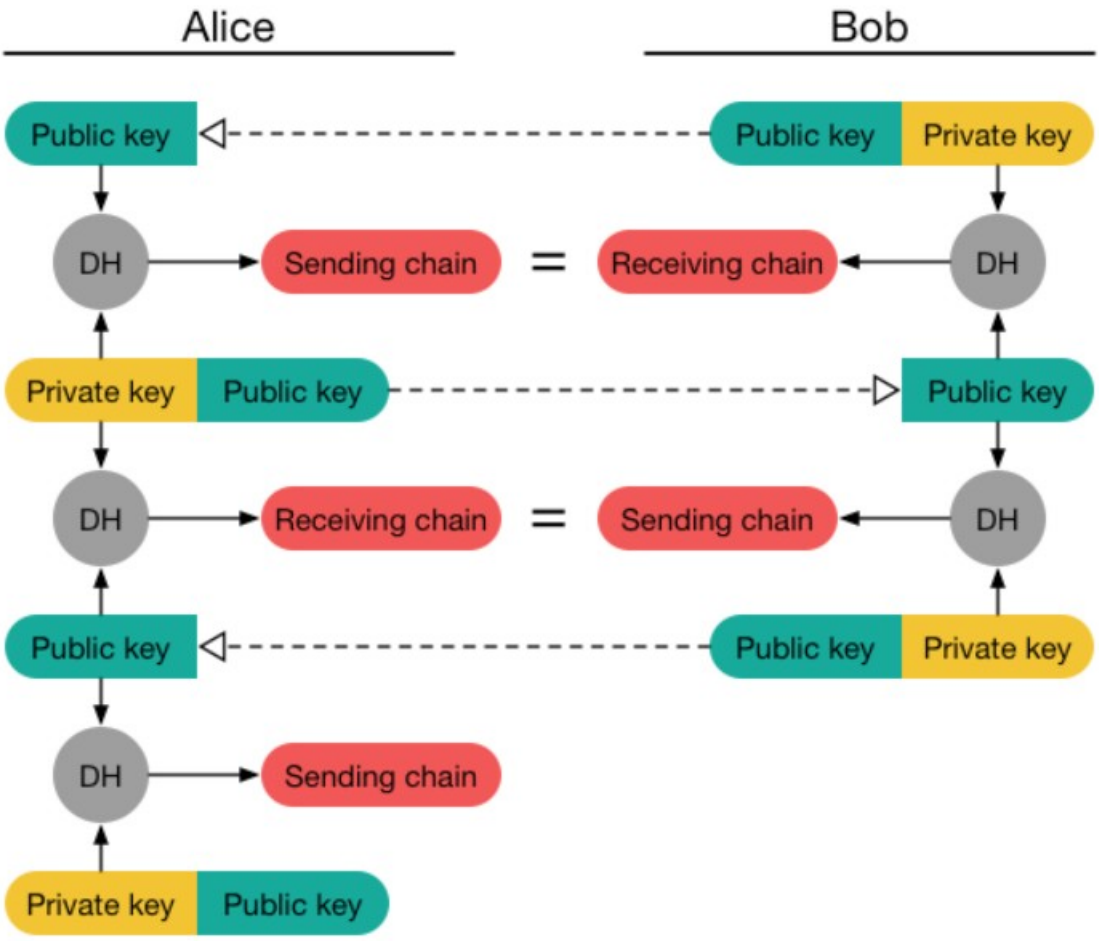



Messages sent by Bob include his new public key. Alice will receive one of those messages and executed her DH ratchet step, generating a new ratchet key pair and calculating 2 DH output one of them matches Bob´s DH.

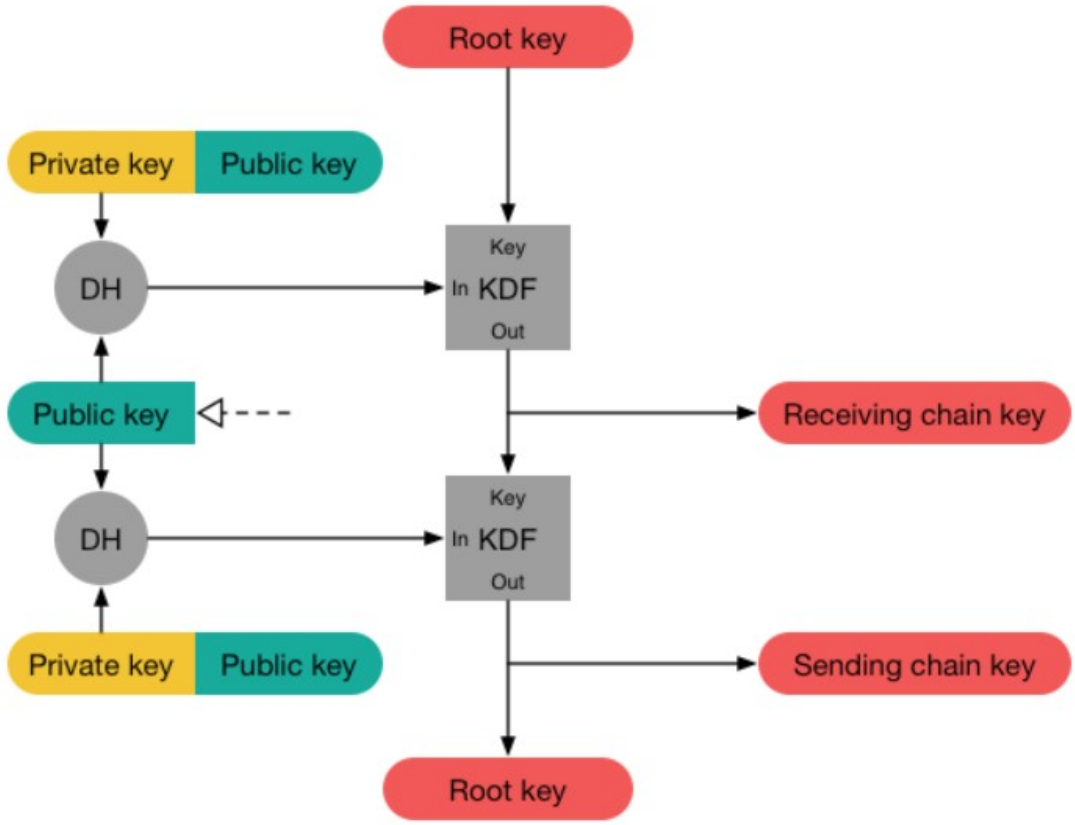


This result in a “Ping Pong” behavior as the parties takes turns replacing ratchet key pairs. An attacker who compromise one of the parties might learn the value of a current ratchet private key, but the private key will eventually be replaced with an unccompromised one. At that point the DH calculation between the ratchet key pairs will define a new DH output unknown to the attacker.

Using the DH outputs in every DH ratchet step, new sending and receiving chain keys are generated. The following diagram shows how Alice with her first DH output derives a first sending chain that matches Bob´s receiving chain. Then Bob uses his second DH output to derive a new sending chain that matches also Alice´s receiving chain and so on.



The above diagram explained the derivation of the sending and receiving chain keys directly form the DH outputs. But instead of that a root chain is used where the DH outputs are the KDF inputs to this root chain, and the KDF outputs from this root chain are then the sending and receiving chain keys. This is described in the following picture.




#### 4.2.3.4 Double Ratchet

The Double ratchet is the combination between the symmetric key and the DH ratchet to assure forward and future secrecy.

When Alice sends or receives a message she executes a symmetric key ratchet step to derive the message key. Before this step, when she receives a public ratchet key from Bob, she executes then a Diffie-Hellman ratchet step to calculate DH output which will be the input to the root chain where it will be derived by the KDF and as output Alice finally gets new key for the sending or receiving chain.

#### 4.2.3.5 encryption & decryption premitives

**-curve 25519**

Given Bob’s 32-byte private key, Curve25519 generates his 32-byte public key. Given Bob’s 32-byte private key and Alice’s 32-byte public key, Curve25519 generates the master secret key shared by the two parties. The secret is subsequently used to authenticate and start encrypting messages between them. This algorithm was carefully designed to allow all 32-byte strings as Diffie-Hellman public keys. The Signal protocol leverages Curve25519 for all asymmetric cryptographic operations.

**-AES-256**

This is a symmetric block cipher to protect and encrypt sensitive data. This cipher encrypts and decrypts data in blocks of 256-bits. Symmetric ciphers use the same key for encrypting and decrypting data, therefore Bob and Alice must both know, and use, the same secret key. There are a total of 14 rounds of 256-bit keys — one round consisting of several processing steps that include substitution, transposition, and randomly mixing the plaintext  to output a ciphertext .

**-HMAC-SHA256 (Hash-Based Message Authentication Code)**

That is a specific type of message authentication code involving a cryptographic hash function and a secret cryptographic key. It also verifies the data integrity, as well as the authentication of a message. This type of keyed hash algorithm is constructed from the SHA-256 hash function. This algorithm mixes a master secret key with the message data, hashes the result with the hash function then mixes that hash value with the secret key again, and finally invokes the hash function again. The output hash is 256 bits in length.

## Comparison

After the analyses of the two protocols we try now to distinct the difference between them.

First of all the protocols are based on Difffie-Hellman key generation but different levels.

To generate a shared secret key the MTProto(telegram) uses a classic Diffie-Hellman key exchange, based on the private and public keys of the two parties of the communication.

The shared secret key is calculated using a mod function and is changed every-time it has been used to decrypt and encrypt more than 100 messages, or has been used for more one week. On the other side the signal protocol uses an X3DH where the secret key is regenerated in every communication step by changing the key pair (private key, public key).

Both protocols assure a forward secrecy by regenerating the key and securely discard or delete the old ones, they just differ in the method.

To assure its future secrecy the Signal protocol combines the Diffie-Hellman ratchet to his symetric-key ratchet. In MtProto, this secrecy is assured automatically just by recalculating the key.

## Conclusion

we can conclude that both of MTProto and Signal protocol are two of the most secure massaging protocols despite the known attacks on Telegram which stay just threats or the WhatsApp viedeo calling attack which the company has solved by developing the security mechanism and updating the App.

Both Apps use the EE2E with two different protocols but based on the same cryptography primitives. To decide that one App is more secure than the other is not really evident at least from a cryptographic point of view.

## References

https://core.telegram.org/api/end-to-end

https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

https://www.poweradmin.com/blog/a-quick-guide-to-encrypted-messaging/

Cryptography and Information Security Naha, Japan, Jan. 24 - 27, 2017 The Institute of Electronics, Information and Communication Engineers. SCIS 2017.

https://signal.org/docs/

CISPA Helmholtz Center for Information Security, Germany.