On the CCA (in)Security of MTProto

Jakob Jakobsen
jakjak@cs.au.dk

Claudio Orlandi
orlandi@cs.au.dk

Department of Computer Science
Aarhus University
Denmark

ABSTRACT

Telegram is a popular messaging app which supports end-to-end encrypted communication. In Spring 2015 we performed an audit of Telegram's Android source code. This short paper summarizes our findings.

Our main discovery is that the symmetric encryption scheme used in Telegram – known as MTProto – is not IND-CCA secure, since it is possible to turn any ciphertext into a different ciphertext that decrypts to the same message.

We stress that this is a theoretical attack on the definition of security and we do not see any way of turning the attack into a full plaintext-recovery attack. At the same time, we see no reason why one should use a less secure encryption scheme when more secure (and at least as efficient) solutions exist.

The take-home message (once again) is that well-studied, provably secure encryption schemes that achieve strong definitions of security (e.g., *authenticated-encryption*) are to be preferred to home-brewed encryption schemes.

1. INTRODUCTION

Telegram. Telegram is an instant messaging service designed for mobile devices, that has had (as of May 2015) 62 million monthly active users[1] and is used to deliver 10^{10} messages daily[2]. It offers two modes, the first being the regular chat mode in which all messages can be read by the server and are stored, allowing for synchronization between devices and group chats. The second mode, called secret chat, is an end-to-end encrypted chat which supports at most two parties. Messages are sent through the server in encrypted form only, and are claimed not to be stored.

The official Telegram client application is open source, allowing for full auditing, but the server software is not. According to the Electronic Frontier Foundation[3], Telegram was audited in February 2015 and the secret chat mode achieved full marks for its security.

Our Analysis. In Spring 2015 we performed an independent audit of the Telegram source code. The analysis is based on the official Telegram source code for version 2.7.0, downloaded in mid April 2015 from github[4].

Instead of using proven cryptographic constructs, Telegram opted to create its own original protocol for symmetric encryption, known as MTProto. Our main finding is that MTProto does not satisfy the definitions of *authenticated encryption (AE)* or *indistinguishability under chosen-ciphertext attack (IND-CCA)* [KL14]. Those definitions are designed to guarantee security of symmetric encryption schemes in the presence of active attacks (i.e., attacks where the adversary freely manipulates ciphertexts and has access to a decryption oracle). We note that MTProto includes some kind of integrity mechanism, which means that active attacks are included in Telegram's *threat-model.* Alas, the non-standard countermeasures that MTProto implements are not enough to satisfy current standards for security of encryption schemes.

In a nutshell, MTProto uses a block-cipher structure and therefore includes padding. Unfortunately, neither *the length* nor *the content* of the padding are checked for integrity during decryption. This means that given any ciphertext it is possible to create different ciphertexts that decrypt to the same message, thus breaking the definition of security. In fact, we found two ways of doing this: either by *adding one more random block at the end of the ciphertext* or by *replacing the last block with a random block.* The first method produces a ciphertext which decrypts correctly with probability 1, while the second method produces a ciphertext which decrypts correctly with probability exponentially small in the amount of payload contained in the last block (i.e., the block length minus the padding length).

Our findings were communicated to the Telegram team on the 3rd of September 2015. The Telegram team has not fixed the problem, but has changed the *description* of the protocol instead.

2. MTPROTO

From a high-level point of view, Telegram allows two devices to exchange a long term key using Diffie-Hellman key exchange. Afterwards, the two devices can use this key to

[1]http://techcrunch.com/2015/05/13/telegram-says-its-hit-62-maus-and-messaging-activity-has-doubled/.
[2]https://telegram.org/blog/10-billion.

[3]https://www.eff.org/secure-messaging-scorecard.
[4]https://github.com/DrKLO/Telegram.

SPSM'16, October 24 2016, Vienna, Austria

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4564-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2994459.2994468

exchange encrypted messages using a symmetric encryption scheme known as MTProto. In a nutshell MTProto is a combination of:

1. A lesser known mode of operation, namely *Infinite Garble Extension (IGE)*;
2. A short term key derivation mechanism;
3. An integrity check on the plaintext;

Here is an abstract description of MTProto:

2.1 MTProto Encryption

Long-Term key: The sender and the receiver share a long term-key K of length λ. (In MTProto $\lambda = 2048$ bits and the key is obtained by running the Diffie-Hellman Key-Exchange over a multiplicative subgroup of $\mathbb{Z}_p^*$, where p and $p - 1/2$ are primes);

Payload: The payload $x = (aux, rnd, msg, pad)$ is obtained by concatenating:

1. some auxiliary information aux;
2. a random salt rnd (in MTProto this has length 128 bits)[5];
3. the message msg;
4. some arbitrary padding pad such that $|x| \bmod B = 0$, where B is the block length; For technical reasons this is never more than 96 bits (and not 120 bits as described by the official documentation).

Authentication Tag: MTProto employs a hash function

$$\mathsf{Tag} : \{0,1\}^* \to \{0,1\}^h$$

for authentication purposes. This is used to compute

$$tag = \mathsf{Tag}(aux, rnd, msg)$$

(crucially, pad is not part of the input of the authentication tag). In MTProto Tag = SHA-1 and $h = 128$ bits;

Short-Term Key Derivation: MTProto employs a hash function

$$\mathsf{KDF} : \{0,1\}^{\lambda+h} \to \{0,1\}^{\kappa+2B}$$

for key derivation purposes. This is used to compute

$$(k, x_0, y_0) = \mathsf{KDF}(K, tag)$$

of length (κ, B, B) respectively. In MTProto the output of KDF is obtained by (a permutation of) the output of four SHA-1 invocations, which in turn take as input (a permutation of) the tag and (different) portions of the long term key K.

IGE Encryption: MTProto employs an efficiently invertible pseudo-random permutation (PRP)

$$\mathsf{F}, \mathsf{F}^{-1} : \{0,1\}^\kappa \times \{0,1\}^B \to \{0,1\}^B$$

to implement the IGE mode of operation. Let $x_1, \dots, x_\ell$ be the ℓ blocks of the payload, each of length B, then IGE computes

$$y_i = \mathsf{F}_k(x_i \oplus y_{i-1}) \oplus x_{i-1}$$

(where x_0, y_0 are defined in the previous step). In MTProto F =AES with $\kappa = 256$ bits and $B = 128$ bits;

Resulting Ciphertext: The output of MTProto is

$$c = (tag, y_1, \dots, y_\ell)$$

(as well as other information which is not relevant for our presentation).

2.2 MTProto Decryption

Input Parsing: The ciphertext c is parsed as

$$c = (tag, y_1, \dots, y_\ell)$$

Short-Term Key Re-computation: The short-term key and the initialization blocks are recomputed as

$$(k, x_0, y_0) = \mathsf{KDF}(K, tag)$$

IGE Decryption: The payload is recovered by computing:

$$x_i = y_{i-1} \oplus \mathsf{F}_k^{-1}(y_i \oplus x_{i-1})$$

and $(x_1, \dots, x_\ell)$ is parsed as (aux, rnd, msg, pad). The padding pad is stripped from the payload by reading the byte length from the auxiliary data, and removing any bytes beyond this length.

Tag Verification: The decryption checks if

$$tag \overset{?}{=} \mathsf{Tag}(aux, rnd, msg)$$

and, if so, outputs msg (or $\perp$ otherwise).

Interestingly, if a Telegram client receives an ill-formed ciphertext, it simply drops the message and *does not* notify the sender about the error. This implies that it seems very hard for a network attacker to detect whether a decryption was performed successfully or not (and therefore it does not seem possible to run a Bleichenbacher-style attack [Ble98]). However, we cannot exclude that an attacker which sits on the same client and shares resources (such as memory, CPU, etc.) might be able to detect if (and how) decryption fails using other channels.

3. MTPROTO IS NOT IND-CCA SECURE

Here we briefly present the two attacks that show that MTProto is not IND-CCA secure. We assume the reader to be familiar with the notion of IND-CCA security. More details on the attacks (and experimental validations) can be found in [Jak15].

Once again, we stress that the attacks are only of theoretical nature and we do not see a way of turning them into full-plaintext recovery. Yet, we believe that these attacks are yet another proof (see e.g., [JN15]) that designing your own crypto rarely is a good idea.

[5]Note that this design choice is very different than standard constructions of modes of operations such as CTR, CBC, etc. Instead of adding rnd as an IV, in MTProto the randomness is appended to the message to be encrypted.

3.1 Attack #1: Padding-Length Extension

Given a ciphertext

$$c = (tag, y_1, \dots, y_\ell)$$

which encrypts a payload

$$x = (aux, rnd, msg, pad)$$

the attacker picks a random block $r \leftarrow \{0,1\}^B$ and outputs

$$c' = (tag, y_1, \dots, y_\ell, r)$$

When this is run via the decryption process, the resulting payload is

$$x' = (aux, rnd, msg, pad')$$

where $pad' = (pad, pad^*)$ where: 1) pad^* is randomly distributed and 2) the length of $|pad'|$ is larger than the block length B. However, since the length of the pad' is not checked during decryption, and pad' is not an input to the authentication function Tag, decryption outputs msg with probability 1.

Mitigation. This attack can be mitigated simply by checking the length of pad' during decryption, and letting decryption abort if this is larger than the block size. Note that since honest clients never produce ill-formed ciphertexts, patching the decryption process in this way will not prevent clients running older versions of the software to communicate with clients running the patched version.

3.2 Attack #2: Last Block Substitution

Given a ciphertext

$$c = (tag, y_1, \dots, y_\ell)$$

which encrypts a payload

$$x = (aux, rnd, msg, pad)$$

the attacker picks a random block $r \leftarrow \{0,1\}^B$ and outputs

$$c' = (tag, y_1, \dots, y_{\ell-1}, r)$$

When this is run via the decryption process, the resulting payload is

$$x' = (aux, rnd, msg', pad')$$

Let $P = |pad|$ be the length of the original pad, then it holds that the last $B - P$ bits of msg' are randomly distributed (due to the use of the PRP in the IGE decryption). Hence we have $msg' = msg$ with probability 2^{P-B}, and in this case the verification of the authentication tag succeeds and the decryption outputs msg. According to the official description of MTProto the padding length is between 0 and 120 bits, which means that in the best case scenario the above attack succeeds with probability 2^{-8}. However experimental validation shows that this is not the case and the length of the padding is in fact between 0 and 96 bits (this is due to the way the objects are serialized in chunks of four bytes in Java).

So we conclude that in the best case this attack succeeds with probability 2^{-32} which, while still being too high to satisfy the definition of security, makes this attack more cumbersome than expected.

Mitigation. This attack can be mitigated by adding pad to the computation of the authentication tag. However, since this requires to change the encryption process as well, patched clients will not be able to communicate with unpatched clients.

4. CONCLUSION

We described two simple attacks which show that MTProto, the symmetric encryption scheme used by Telegram, fails to achieve desirable notions of security such as *indistinguishability under chosen-ciphertext attack* or *authenticated encryption*. While the first attack can be mitigated by applying a patch which does not affect backward compatibility with unpatched (but honest) senders, the second one can only be patched in a way that only allows communication between patched clients. Therefore it is fair to ask whether the next version of Telegram should use a patched version of MTProto or a better studied mode of operation which provably implements *authenticated encryption*.

We believe that the second option is the only sensible choice since while a patched version of MTProto would not be susceptible to the attacks described in this note, this gives *no guarantees* that no other attacks exist. On the other hand well studied *authenticated encryption schemes* offer much stronger security guarantees. In addition, MTProto does not seem to be more efficient than the many available options for *authenticated encryption* such as e.g., *encrypt-then-MAC* using AES in counter mode (which can be parallelized better than IGE) together with e.g., HMAC.

Acknowledgements. Orlandi is supported by the Danish National Research Foundation (grant 61361136003).

5. REFERENCES

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 1–12, 1998.

[Jak15] Jakob Jakobsen. *A practical cryptanalysis of the Telegram messaging protocol.* Master Thesis, Aarhus University (Available on request)., 2015.

[JN15] Philipp Jovanovic and Samuel Neves. Dumb crypto in smart grids: Practical cryptanalysis of the open smart grid protocol. FSE, 2015. http://eprint.iacr.org/2015/428.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography.* CRC Press, 2014.

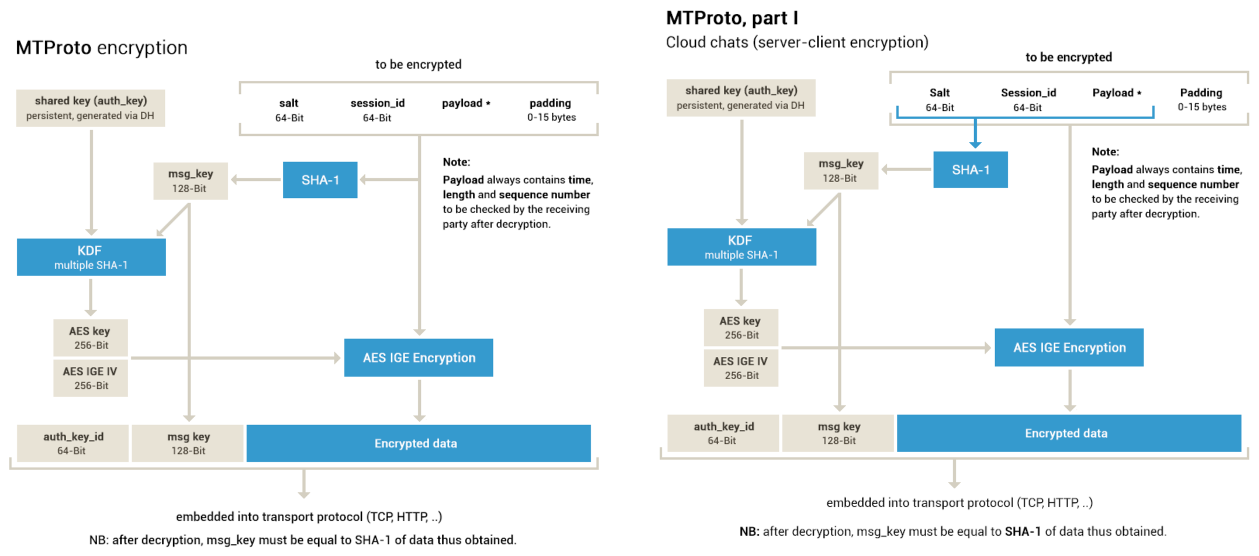
MTProto encryption
to be encrypted
shared key (auth_key)
persistent, generated via DH
salt
64-Bit
session_id
64-Bit
payload *
padding
0-15 bytes
msg_key
128-Bit
SHA-1
Note:
Payload always contains time, length and sequence number to be checked by the receiving party after decryption.
KDF
multiple SHA-1
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 transport protocol (TCP, HTTP, ..)
NB: after decryption, msg_key must be equal to SHA-1 of data thus obtained.
MTProto, 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
0-15 bytes
msg_key
128-Bit
SHA-1
Note:
Payload always contains time, length and sequence number to be checked by the receiving party after decryption.
KDF
multiple SHA-1
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 transport protocol (TCP, HTTP, ..)
NB: after decryption, msg_key must be equal to SHA-1 of data thus obtained.

Figure 1: MTProto description prior (left) and after (right) our work. On the left hand-side the padding is feed into the SHA-1 block, while of the right-hand side the padding is no longer feed into the SHA-1 block.

APPENDIX

Historical Notes and Telegram Response

Prior to our study the technical FAQ section of Telegram's website[6] showed the pictorial description of MTProto reported on the left-hand side of Figure 3.2. In the figure the padding is given as input to the hash function and is therefore authenticated. At the same time the FAQ contained a section about CCA security and claimed that *"[MTProto] negates known CCAs"*.

Our audit showed a discrepancy between this protocol description and the implemented protocol, where the padding *is not authenticated*.

After the notification of our findings in September 2015 (and the first public release of this short paper in December 2015) the Telegram team changed the description of their protocol (as shown on the right-hand side of Figure 3.2) so that now the protocol description matches the implemented protocol. A paragraph arguing for practical irrelevance of IND-CCA security was added to the FAQ as well.

There is of course no way of knowing whether the lack of padding authentication is part of the original design of MTProto (meaning that we merely uncovered a "bug" in the original pictorial description) or whether the padding was supposed to be authenticated (meaning that we uncovered a "bug" in their implementation, and that the protocol design was retroactively changed to match the faulty implementation).

[6]https://web.archive.org/web/20150323142815/https://core.telegram.org/techfaq