Four Attacks and a Proof for Telegram

RWC 2022
April 14, 2022

Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, Igors Stepanovs
Royal Holloway, University of London; ETH Zürich

Based on the paper to appear at IEEE S&P 2022.
More information at: https://mtpsym.github.io/
Important advantages of Telegram:
- Support of public and private group chats for up to 200’000 people (Signal up to 1’000; WhatsApp up to 256).
- Pseudonimity: can use a pseudonym, not revealing phone number to others (not supported in Signal and WhatsApp).
- Other features: anonymous polls; disappearing messages; timed or scheduled messages; ability to delete messages sent by others.

Common use cases: large public groups up to 50’000 members, and small private groups.

Collective Information Security in Large-Scale Urban Protests: the Case of Hong Kong

Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková, Royal Holloway, University of London

Telegram was the predominant messaging application used in the Hong Kong protests in 2019-2020.

Telegram was perceived to provide more security than its competitors.

### Monthly active users in Jan 2022:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Active Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>WhatsApp</td>
<td>2000 · 10^6</td>
</tr>
<tr>
<td>WeChat</td>
<td>1263 · 10^6</td>
</tr>
<tr>
<td>FB Messenger</td>
<td>988 · 10^6</td>
</tr>
<tr>
<td>QQ</td>
<td>574 · 10^6</td>
</tr>
<tr>
<td>Snapchat</td>
<td>557 · 10^6</td>
</tr>
<tr>
<td>Telegram</td>
<td>550 · 10^6</td>
</tr>
</tbody>
</table>

According to Statista 2022.
Cloud Chats and Secret Chats

<table>
<thead>
<tr>
<th></th>
<th>Cloud Chats</th>
<th>Secret Chats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group communication</td>
<td>✓</td>
<td>❌</td>
</tr>
<tr>
<td>1-on-1 communication</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Type of encryption</td>
<td>client-server</td>
<td>end-to-end</td>
</tr>
<tr>
<td>Enabled by default?</td>
<td>✓</td>
<td>❌</td>
</tr>
</tbody>
</table>

The **MTProto** protocol – Telegram’s equivalent of the TLS record protocol.

**Cloud Chats** encrypt and authenticated messages using **MTProto**.

**Secret Chats** add another layer of **MTProto** encryption, i.e. messages are double-encrypted.

The **MTProto** protocol is not well-studied:

- **2013**: Telegram launched with MTProto 1.0.
- **2016**: Jakobsen and Orlandi showed that MTProto 1.0 is not CCA-secure.
- **2017**: Telegram released MTProto 2.0 that addressed the security concerns.
- **2017**: Sušánka and Kokeš reported an attack based on improper validation in the Android client.
- **2018**: Kobeissi reported input validation bugs in Telegram’s Windows Phone client.
- **2020**: Miculan and Vitacolonna proved MTProto 2.0 secure in a symbolic model, assuming ideal building blocks.

The focus in the literature has been on the **Secret Chats**. We focus on the security of the **Cloud Chats**.

---

“Q: Why are you not using X? (insert solution)
While other ways of achieving the same cryptographic goals, undoubtedly, exist, we feel that the present solution is both robust and also succeeds at our secondary task of beating unencrypted messengers in terms of delivery time and stability.”

Telegram FAQ ([https://core.telegram.org/techfaq](https://core.telegram.org/techfaq))

Why not use TLS instead of MTProto?
The Design of MTProto 2.0

**MTProtoEncrypt**

```
MAC(mk, p)
msg_key ← SHA-256(mk||p)[64 : 192]
Return msg_key

KDF(kk, msg_key)
(kk0, kk1) ← kk
k0 ← SHA-256(msg_key||kk0)
k1 ← SHA-256(kk1||msg_key)
k ← k0∥k1 ; Return k
```

Payload p:

<table>
<thead>
<tr>
<th>64 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>32 bits</th>
<th>12-1024 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_salt</td>
<td>session_id</td>
<td>msg_seq_no</td>
<td>msg_length</td>
<td>msg_data</td>
</tr>
</tbody>
</table>

Infinite Garble Extension (IGE) mode:
The Design of MTProto 2.0

\[
\text{MAC}(mk, p) \\
\text{msg_key} \leftarrow \text{SHA-256}(mk\|p)[64 : 192] \\
\text{Return msg_key}
\]

\[
\text{KDF}(kk, \text{msg_key}) \\
(kk_0, kk_1) \leftarrow kk \\
k_0 \leftarrow \text{SHA-256}(\text{msg_key}\|kk_0) \\
k_1 \leftarrow \text{SHA-256}(kk_1\|\text{msg_key}) \\
k \leftarrow k_0\|k_1; \text{ Return } k
\]

\[
\text{payload } p: \\
\begin{array}{cccccc}
\text{server_salt} & \text{session_id} & \text{msg_seq_no} & \text{msg_length} & \text{msg_data} & \text{padding}
\end{array}
\]

MTProto.Encrypt

\[
\begin{array}{c}
\text{MAC} \\
\text{KDF} \\
\text{SE.Enc}
\end{array}
\]

Infinite Garble Extension (IGE) mode:

\[
\begin{array}{c}
IV_c \\
E_K \\
\cdots \\
E_K \\
IV_m \\
c_1 \\
c_2 \\
c_{t-1} \\
c_t
\end{array}
\]

Used to encrypt messages from client to server.

\[
\begin{array}{c}
IV_c \\
E_K \\
\cdots \\
E_K \\
IV_m \\
c_1 \\
c_2 \\
c_{t-1} \\
c_t
\end{array}
\]

Used to encrypt messages from server to client.

\[
\begin{array}{c}
kk_0 (288 \text{ bits}) \\
kk_1 (288 \text{ bits}) \\
mk (256 \text{ bits})
\end{array}
\]

raw \(g^x\) value (2048-bit long)

\[
\begin{array}{c}
kk_0 (288 \text{ bits}) \\
kk_1 (288 \text{ bits}) \\
mb (256 \text{ bits})
\end{array}
\]

64 bits 32 bits 96 bits 256 bits
The Design of MTProto 2.0

\[
\text{MAC}(mk,p) \\
\text{msg_key} \leftarrow \text{SHA-256}(mk||p)[64:192] \\
\text{Return msg_key}
\]

\[
\text{KDF}(kk,\text{msg_key}) \\
(kk_0, kk_1) \leftarrow kk \\
k_0 \leftarrow \text{SHA-256}(\text{msg_key}||kk_0) \\
k_1 \leftarrow \text{SHA-256}(kk_1||\text{msg_key}) \\
k \leftarrow k_0||k_1; \text{Return } k
\]

**MTProto.Encrypt**

- **payload p:** server_salt session_id msg_seq_no msg_length msg_data padding

- Infinite Garble Extension (IGE) mode:

- Used to encrypt messages from client to server.

- Used to encrypt messages from server to client.

**SHA-256 compression function**

\[
h_{256}(H_{i-1}, x_i) = H_{i-1} + \text{SHACAL-2}(x_i, H_{i-1})
\]

**block cipher**

\[
\text{SHACAL-2}(x_i, H_{i-1})
\]

- **raw g^{xy} value (2048-bit long)**

- **kk0 (288 bits)**

- **kk1 (288 bits)**

- **mk (256 bits)**

- **1024 bits**

- **12-1024 bytes**
We found 4 weaknesses in MTProto. Reported to Telegram on April 16, 2021. Telegram acknowledged receipt soon after. Acknowledged the behaviours on June 8, 2021. Agreed on disclosure on July 16, 2021.

No security or bugfix releases except for immediate post-release crash fixes. Did not wish to issue security advisories at the time of patching. Did not commit to release dates for specific fixes.

Fixes were rolled out as part of regular updates:
- 7.8.1 for Android
- 7.8.3 for iOS
- 2.8.8 for Desktop

2. Message reordering attack. // Technically trivial; easy to exploit.
3. Timing side-channel attacks against clients. // Plaintext recovery; infeasible in practice.
4. Timing side-channel attack against servers. // MitM on key exchange; infeasible in practice.

Telegram awarded a bug bounty for side-channel attacks and overall analysis.
Four Attacks Against Telegram

We found 4 weaknesses in MTProto.
Reported to Telegram on April 16, 2021.
Telegram acknowledged receipt soon after.
Acknowledged the behaviours on June 8, 2021.
Agreed on disclosure on July 16, 2021.
No security or bugfix releases except for
immediate post-release crash fixes.
Did not wish to issue security advisories
at the time of patching.
Did not commit to release dates for specific fixes.
 Fixes were rolled out as part of regular updates:
  7.8.1 for Android
  7.8.3 for iOS
  2.8.8 for Desktop

2. Message reordering attack. // Technically trivial; easy to exploit.
3. Timing side-channel attacks against clients. // Plaintext recovery; infeasible in practice.
4. Timing side-channel attack against servers. // MitM on key exchange; infeasible in practice.

Telegram awarded a bug bounty for side-channel attacks and overall analysis.
Four Attacks Against Telegram

Reported to Telegram on April 16, 2021.
Acknowledged the behaviours on June 8, 2021.
Agreed on disclosure on July 16, 2021.

No security or bugfix releases except for immediate post-release crash fixes.
Did not wish to issue security advisories at the time of patching.
Did not commit to release dates for specific fixes.

Fixes were rolled out as part of regular updates:
- 7.8.1 for Android
- 7.8.3 for iOS
- 2.8.8 for Desktop

2. Message reordering attack. // Technically trivial; easy to exploit.
3. Timing side-channel attacks against clients. // Plaintext recovery; infeasible in practice.
4. Timing side-channel attack against servers. // MitM on key exchange; infeasible in practice.

Telegram awarded a bug bounty for side-channel attacks and overall analysis.
Four Attacks Against Telegram

We found 4 weaknesses in MTProto. Reported to Telegram on April 16, 2021. Telegram acknowledged receipt soon after. Acknowledged the behaviours on June 8, 2021. Agreed on disclosure on July 16, 2021. No security or bugfix releases except for immediate post-release crash fixes. Did not wish to issue security advisories at the time of patching. Did not commit to release dates for specific fixes. Fixes were rolled out as part of regular updates:

- 7.8.1 for Android
- 7.8.3 for iOS
- 2.8.8 for Desktop

**MTProto.Encrypt**

1. IND-CPA attack

\[ c_0 \leftarrow \text{MTPEnc}(p) \]
\[ c_1 \]
\[ c_2 \leftarrow \text{MTPEnc}(p) \]

Client expects \( c_1 \) be an encryption of ACK. Otherwise, it re-encrypts the payload. Birthday-bound collision in msg_key causes the first 2 blocks of \( c_0, c_2 \) be the same. IND-CPA thus breaks privacy of \( c_1 \).

2. Message reordering attack. // Technically trivial; easy to exploit.
3. Timing side-channel attacks against clients. // Plaintext recovery; infeasible in practice.
4. Timing side-channel attack against servers. // MitM on key exchange; infeasible in practice.

Telegram awarded a bug bounty for side-channel attacks and overall analysis.
Four Attacks Against Telegram

We found 4 weaknesses in MTProto. Reported to Telegram on April 16, 2021. Telegram acknowledged receipt soon after. Acknowledged the behaviours on June 8, 2021. Agreed on disclosure on July 16, 2021.

No security or bugfix releases except for immediate post-release crash fixes. Did not wish to issue security advisories at the time of patching. Did not commit to release dates for specific fixes.

Fixes were rolled out as part of regular updates:
- 7.8.1 for Android
- 7.8.3 for iOS
- 2.8.8 for Desktop

1. **IND-CPA attack**
   - Client expects $c_1$ to be an encryption of ACK.
   - Otherwise, it re-encrypts the payload.
   - Birthday-bound collision in `msg_key` causes the first 2 blocks of $c_0, c_2$ to be the same.
   - IND-CPA thus breaks privacy of $c_1$.

2. **Message reordering attack**
   - Attack against IND-CPA security. // Theoretical.
   - Message reordering attack. // Technically trivial; easy to exploit.
   - Timing side-channel attacks against clients. // Plaintext recovery; infeasible in practice.
   - Timing side-channel attack against servers. // MitM on key exchange; infeasible in practice.

Telegram awarded a bug bounty for side-channel attacks and overall analysis.
Timing Side-Channel Attacks Against Clients

3 official clients (Android; Desktop; iOS)
ran a sanity check on a decrypted payload
prior to verifying its hash msg_key.
Timing Side-Channel Attacks Against Clients

```
If (msg_length > length) then ...  // Android
Outcome of comparison depends on 32 bits on msg_length.
If comparison fails: two conditional jumps added.
```

![Diagram of timing side-channel attacks against clients]

- `c_1`, `c_2`, ..., `c_i`
- `m_1`, `m_2`, ..., `m_i`
- `IV_m`, `IV_c`
- `E_K^{-1}`, `E_K`
- `c_{i-1}`, `m_{i-1}`
- `c_i`, `m_i`
- `server_salt`, `session_id`, `msg_seq_no`, `msg_length`, `msg_data`, `padding`

64 bits 64 bits 96 bits 32 bits

3 official clients (Android; Desktop; iOS)
ran a sanity check on a decrypted payload
prior to verifying its hash `msg_key`. 
Timing Side-Channel Attacks Against Clients

If \( \text{msg.length} > \text{length} \) then \ldots \hspace{1em} /\hspace{0.5em} \text{Android}
Outcome of comparison depends on 32 bits on \text{msg.length}.
If comparison fails: two conditional jumps added.

If \( \text{msg.length} > 2^{24} \) then \ldots \hspace{1em} /\hspace{0.5em} \text{Desktop}
Outcome of comparison depends on 8 bits on \text{msg.length}.
If comparison fails: MAC verification is omitted.

3 official clients (Android; Desktop; iOS)
ran a sanity check on a decrypted payload
prior to verifying its hash \text{msg.key}.
Timing Side-Channel Attacks Against Clients

If \((\text{msg\_length} > \text{length})\) then \ldots \hspace{1em} \text{// Android}
Outcome of comparison depends on 32 bits on \(\text{msg\_length}\).
If comparison fails: two conditional jumps added.

If \((\text{msg\_length} > 2^{24})\) then \ldots \hspace{1em} \text{// Desktop}
Outcome of comparison depends on 8 bits on \(\text{msg\_length}\).
If comparison fails: MAC verification is omitted.

If not \((12 \leq \ell - \text{msg\_length} \leq 1024)\) then \ldots \hspace{1em} \text{// iOS}
Outcome of comparison depends on 32 bits on \(\text{msg\_length}\).
If comparison fails: MAC verification takes a shorter input.

3 official clients (Android; Desktop; iOS)
ran a sanity check on a decrypted payload
prior to verifying its hash \(\text{msg\_key}\).
Timing Side-Channel Attacks Against Clients

If \((\text{msg} \text{. length} > \text{length})\) then \(...\) \text{Android}

Outcome of comparison depends on 32 bits on \text{msg} \text{. length}.
If comparison fails: two conditional jumps added.

If \((\text{msg} \text{. length} > 2^{24})\) then \(...\) \text{Desktop}

Outcome of comparison depends on 8 bits on \text{msg} \text{. length}.
If comparison fails: MAC verification is omitted.

If not \((12 \leq \ell - \text{msg} \text{. length} \leq 1024)\) then \(...\) \text{iOS}

Outcome of comparison depends on 32 bits on \text{msg} \text{. length}.
If comparison fails: MAC verification takes a shorter input.

### Plaintext Recovery Attacks Against SSH

Martin R. Albrecht, Kenneth G. Paterson and Gaven J. Watson

Assume we know contents of \(m_1\) and \(m_i-1\).
Want to learn the contents of \(m_i\).
Set \(c_2 := (c_i \oplus m_{i-1}) \oplus m_1\).
Get \(m_2 = (m_i \oplus c_{i-1}) \oplus c_1\).
Infer bits of \(m_2\) from timing side-channel.
Derive the corresponding bits of \(m_i\).

3 official clients (Android; Desktop; iOS)
rung a sanity check on a decrypted payload
prior to verifying its hash \text{msg} \_\text{key}.

- server_salt
- session_id
- msg_seq_no
- msg_length
- msg_data
- padding

64 bits | 64 bits | 96 bits | 32 bits | \(\ell\) bytes
Timing Side-Channel Attacks Against Clients

If \( \text{msg.length} > \text{length} \) then \( \ldots \)  // **Android**
Outcome of comparison depends on 32 bits on \text{msg.length}. If comparison fails: **two conditional jumps added**.

If \( \text{msg.length} > 2^{24} \) then \( \ldots \)  // **Desktop**
Outcome of comparison depends on 8 bits on \text{msg.length}. If comparison fails: **MAC verification is omitted**.

If not \((12 \leq \ell - \text{msg.length} \leq 1024)\) then \( \ldots \)  // **iOS**
Outcome of comparison depends on 32 bits on \text{msg.length}. If comparison fails: **MAC verification takes a shorter input**.

### Plaintext Recovery Attacks Against SSH

Martin R. Albrecht, Kenneth G. Paterson and Gaven J. Watson

Assume we know contents of \( m_1 \) and \( m_{i-1} \).
Want to learn the contents of \( m_i \).
Set \( c_2 := (c_i \oplus m_{i-1}) \oplus m_1 \).
Get \( m_2 = (m_i \oplus c_{i-1}) \oplus c_1 \).
Infer bits of \( m_2 \) from timing side-channel.
Derive the corresponding bits of \( m_i \).

The attack fails because \text{server_salt}, \text{session_id} are secret.

---

3 official clients (Android; Desktop; iOS)
ran a sanity check on a decrypted payload prior to verifying its hash \text{msg.key}.
Timing Side-Channel Attacks Against Clients

If \((\text{msg\_length} > \text{length})\) then \(\ldots\) // Android
Outcome of comparison depends on 32 bits on \text{msg\_length}. If comparison fails: two conditional jumps added.

If \((\text{msg\_length} > 2^{24})\) then \(\ldots\) // Desktop
Outcome of comparison depends on 8 bits on \text{msg\_length}. If comparison fails: MAC verification is omitted.

If not \((12 \leq \ell - \text{msg\_length} \leq 1024)\) then \(\ldots\) // iOS
Outcome of comparison depends on 32 bits on \text{msg\_length}. If comparison fails: MAC verification takes a shorter input.

Plaintext Recovery Attacks Against SSH

Martin R. Albrecht, Kenneth G. Paterson and Gaven J. Watson

Assume we know contents of \(m_1\) and \(m_{i-1}\).
Want to learn the contents of \(m_i\).
Set \(c_2 := (c_i \oplus m_{i-1}) \oplus m_1\).
Get \(m_2 = (m_i \oplus c_{i-1}) \oplus c_1\).
Infer bits of \(m_2\) from timing side-channel.
Derive the corresponding bits of \(m_i\).

The attack fails because \text{server\_salt}, \text{session\_id} are secret.

3 official clients (Android; Desktop; iOS)
ran a sanity check on a decrypted payload prior to verifying its hash \text{msg\_key}.

Our attack highlights the brittle design.
Stems from using Encrypt-and-MAC.
Operates with a decryption key on untrusted data.
Would be safer to protect integrity of ciphertext.
Timing Side-Channel Attack Against Servers

We attack **Telegram**'s key exchange.

Telegram uses textbook **RSA** encryption.

\[ m := \text{SHA-1(data)}||\text{data}||\text{padding} \]
We attack Telegram’s key exchange.

Telegram uses textbook RSA encryption.

\[ m := \text{SHA-1(data)} || \text{data} || \text{padding} \]

After RSA decryption but before SHA-1 verification, Telegram parses \( m \) to validate its format and might omit the computation of SHA-1.
Timing Side-Channel Attack Against Servers

We attack Telegram’s key exchange.

Telegram uses textbook RSA encryption.

\[
m := \text{SHA-1(data)} || \text{data} || \text{padding}
\]

After RSA decryption but before SHA-1 verification, Telegram parses \( m \) to validate its format and might omit the computation of SHA-1.

We recover data by solving noisy linear equations via lattice reduction.

Can use data to recover server_salt and session_id. Can use data to run a MitM attack against the (encrypted) DH exchange.

The attack is infeasible in practice because:
- The timing side-channel is very small.
- Recovering session_id requires additional \( 2^{64} \) computation.
- The key exchange would time out before MitM can be completed.
### Timing Side-Channel Attack Against Servers

We attack Telegram’s key exchange.

\[ \text{RSA.Enc}(pk, m) \]

Telegram uses textbook RSA encryption.

\[ m := \text{SHA-1(data)} \| \text{data} \| \text{padding} \]

We recover data by solving noisy linear equations via lattice reduction.

Can use data to recover server_salt and session_id.
Can use data to run a MitM attack against the (encrypted) DH exchange.

The attack is infeasible in practice because:
- The timing side-channel is very small.
- Recovering session_id requires additional \( 2^{64} \) computation.
- The key exchange would time out before MitM can be completed.

“Publishing the server code doesn’t guarantee privacy, because unlike with the client-side code - there’s no way to verify that the same code is run on the servers. [..]

So why not publish the server code anyway, even if it is only a publicity stunt? 3 years ago I learnt that an authoritarian regime [..] was looking for a way to obtain Telegram’s server code. Their plan was to launch their own equally convenient local app and then to shut down all other social media in the country.”

Pavel Durov ([https://t.me/durovschat/515221](https://t.me/durovschat/515221))

After RSA decryption but before SHA-1 verification, Telegram parses \( m \) to validate its format and might omit the computation of SHA-1.
Large parts of **Telegram**’s design remain **unstudied**:

- Secret chats (including encrypted voice and video calls).
- The key exchange.
- Multi-user security.
- Forward secrecy.
- Telegram Passport.
- Bot APIs.
- The higher-level message processing.
- Control messages.
- Encrypted CDNs.
- Cloud storage.

These are pressing topics for future work.