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/

Background

Monthly active users in Jan 2022:

According to Statista 2022.

WhatsApp	$2000 \cdot 10^6$
WeChat	$1263 \cdot 10^6$
FB Messenger	$988 \cdot 10^6$
QQ	$574 \cdot 10^6$
Snapchat	$557 \cdot 10^6$
Telegram	$550 \cdot 10^6$

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.

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.

Cloud Chats and Secret Chats

	Cloud Chats	Secret Chats
Group communication	✓	✗
1-on-1 communication	✓	✓
Type of encryption	client-server	end-to-end
Enabled by default?	✓	✗

"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)

Why not use TLS instead of MTProto?

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**.

The Design of MTProto 2.0

MAC(mk, p)

msg_key ← SHA-256$(mk \| p)[64 : 192]$
Return msg_key

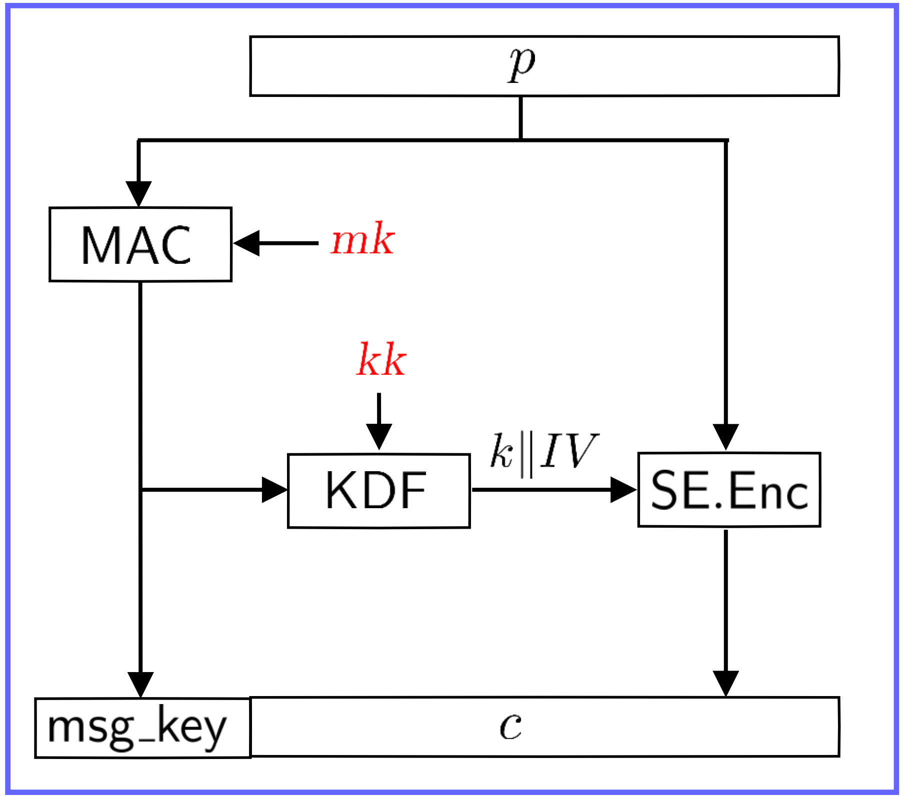
KDF$(kk,$ msg_key$)$

$(kk_0, kk_1) \leftarrow kk$
$k_0 \leftarrow$ SHA-256(msg_key$\|kk_0)$
$k_1 \leftarrow$ SHA-256$(kk_1\|$msg_key$)$
$k \leftarrow k_0 \| k_1$; Return k

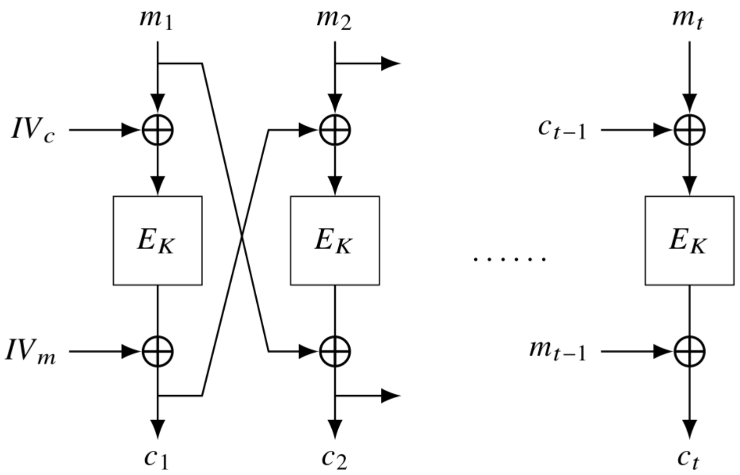
payload p:

64 bits	64 bits	96 bits	32 bits	12-1024 bytes
server_salt	session_id	msg_seq_no	msg_length	msg_data padding

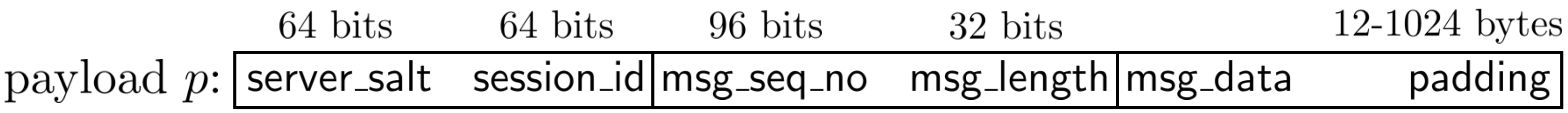
MTProto.Encrypt



Infinite Garble Extension (IGE) mode:



The Design of MTProto 2.0



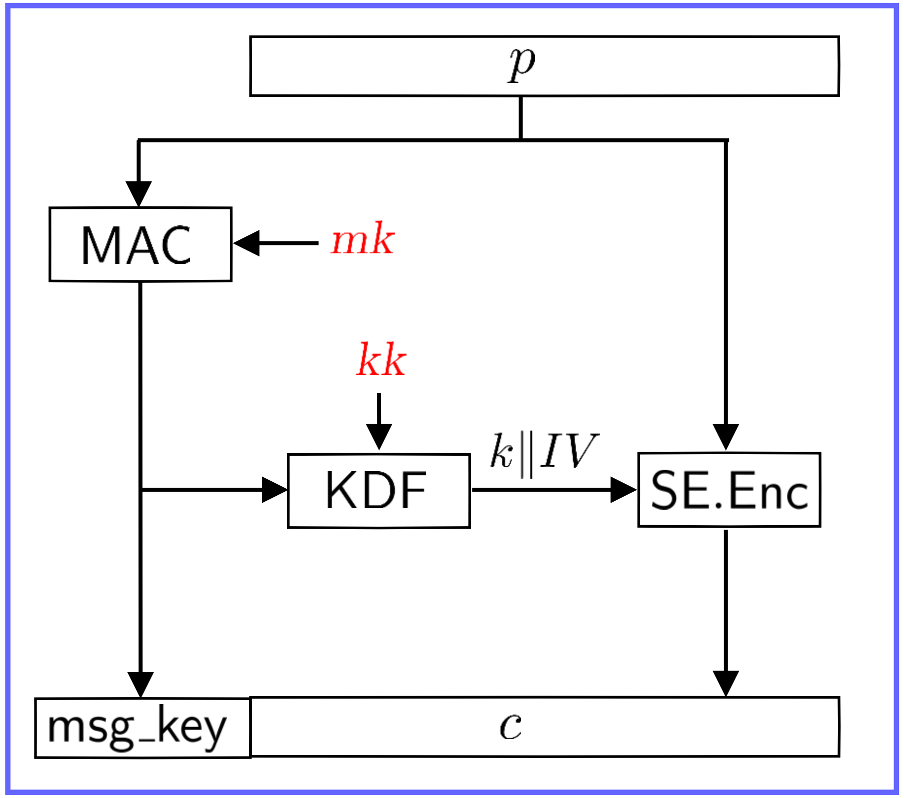
$\mathsf{MAC}(mk, p)$

msg_key $\leftarrow$ SHA-256$(mk \| p)[64:192]$
Return msg_key

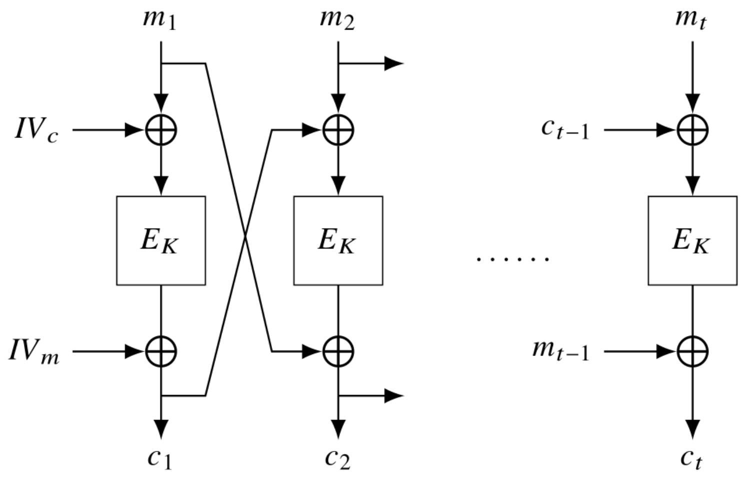
$\mathsf{KDF}(kk, \text{msg\_key})$

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

MTProto.Encrypt

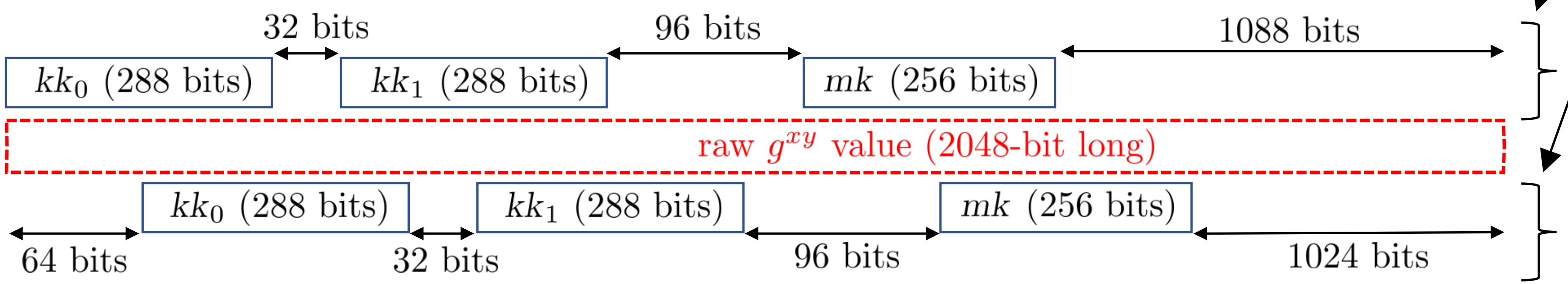


Infinite Garble Extension (IGE) mode:

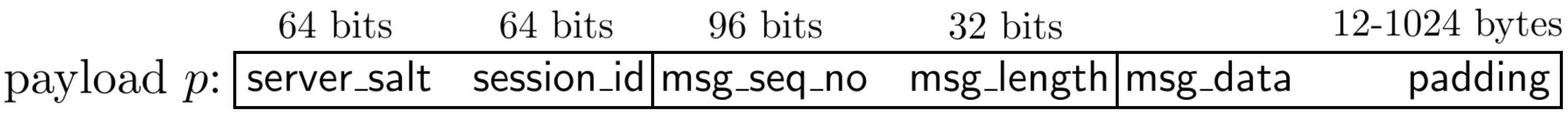


Used to encrypt messages from client to server.

Used to encrypt messages from server to client.



The Design of MTProto 2.0

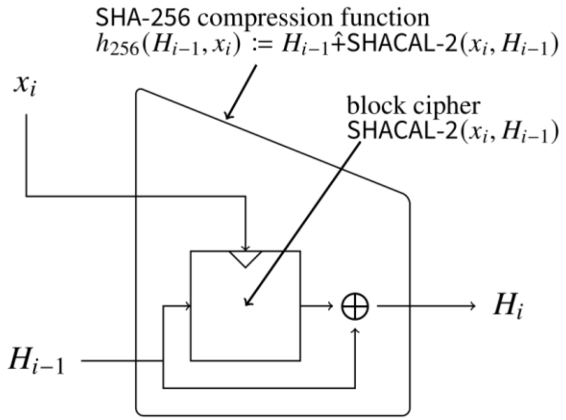


$\mathsf{MAC}(mk, p)$

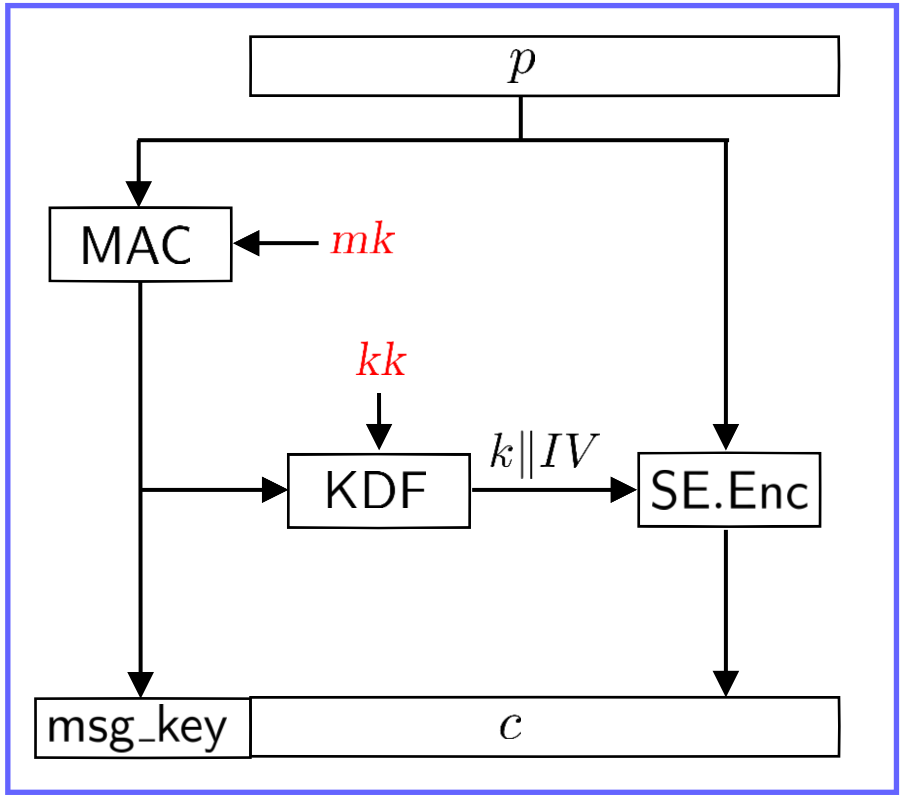
msg_key ← SHA-256$(mk \| p)[64 : 192]$
Return msg_key

$\mathsf{KDF}(kk, \mathsf{msg\_key})$

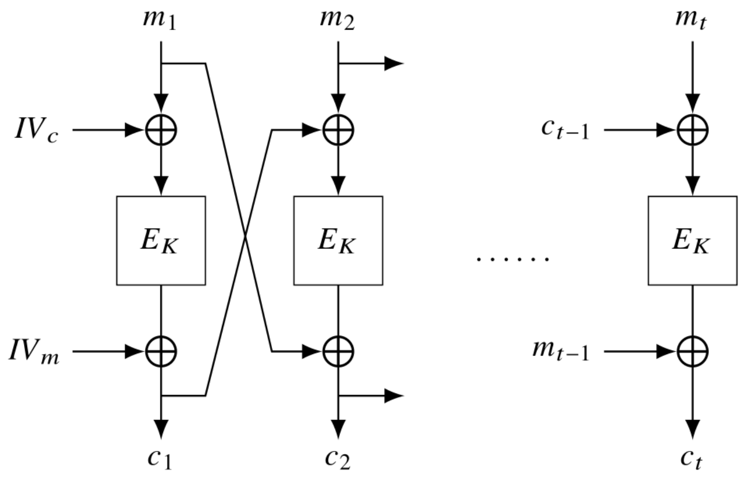
$(kk_0, kk_1) \leftarrow kk$
$k_0 \leftarrow$ SHA-256(msg_key$\| kk_0$)
$k_1 \leftarrow$ SHA-256($kk_1 \|$msg_key)
$k \leftarrow k_0 \| k_1$; Return k



MTProto.Encrypt

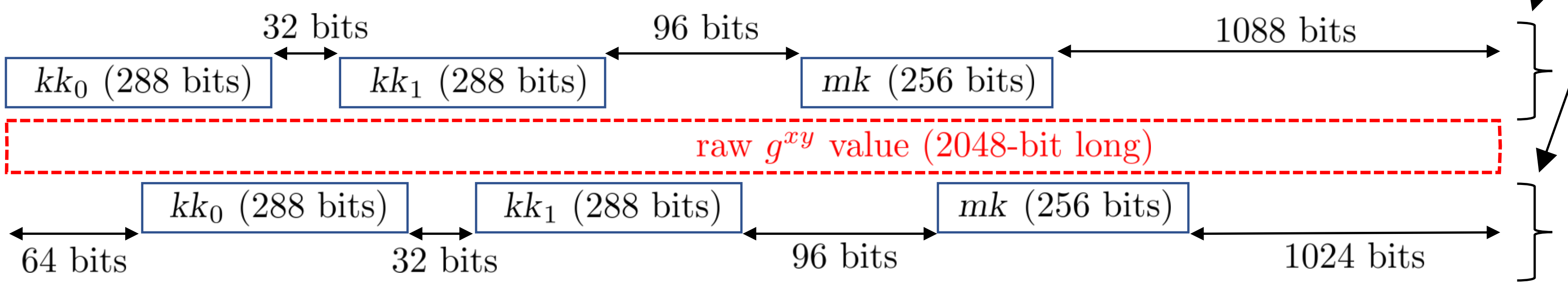


Infinite Garble Extension (IGE) mode:



Used to encrypt messages from client to server.

Used to encrypt messages from server to client.



Four Attacks Against Telegram

64 bits	64 bits	96 bits	32 bits		12-1024 bytes
server_salt	session_id	msg_seq_no	msg_length	msg_data	padding

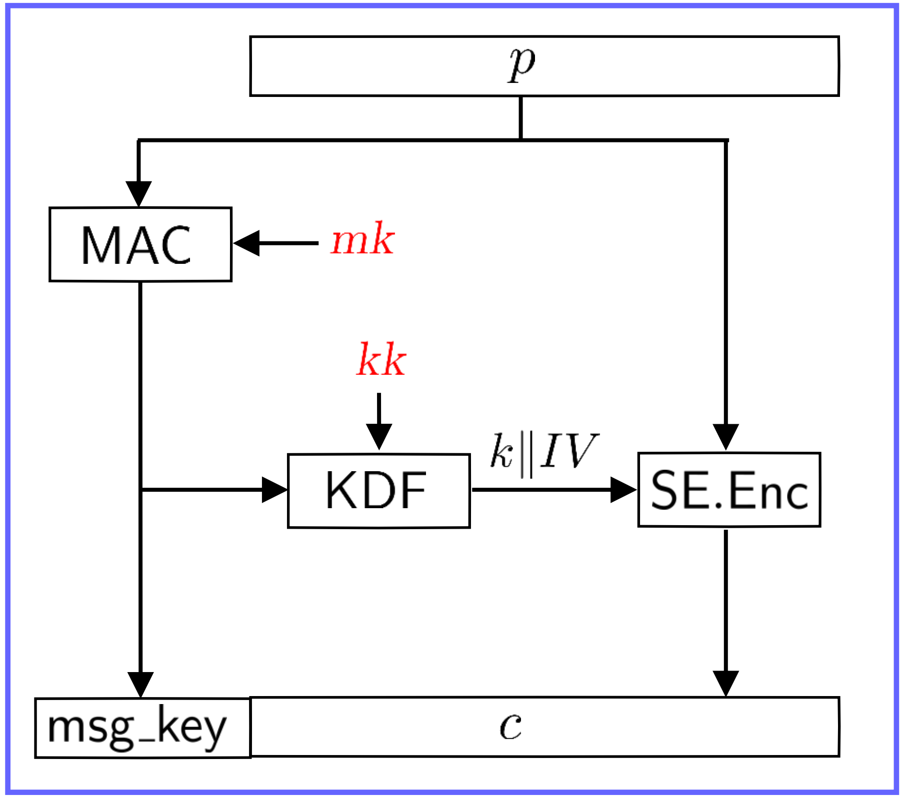
We found 4 weaknesses in MTProto.
Reported to Telegram on April 16, 2021.
Telegram acknowledged receipt soon after.
Acknowledged the behaviours on June 8, 2021.
Agreed on disclosure on July 16, 2021.

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. Attack against IND-CPA security. // Theoretical.
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

64 bits	64 bits	96 bits	32 bits	12-1024 bytes	
server_salt	session_id	msg_seq_no	msg_length	msg_data	padding

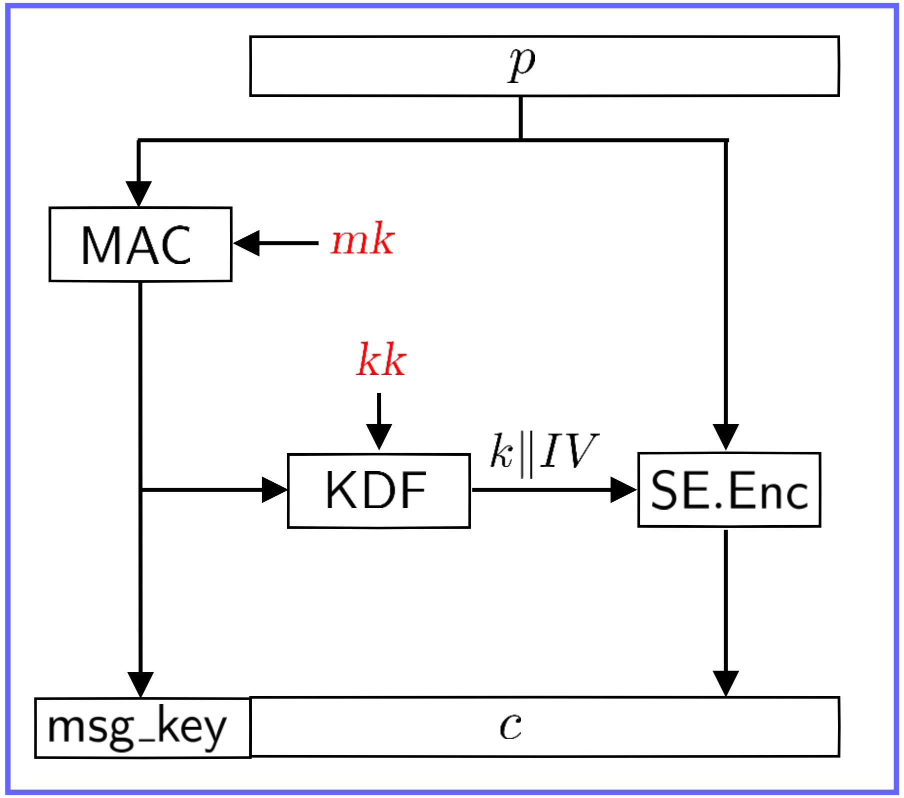
We found 4 weaknesses in MTProto.
Reported to Telegram on April 16, 2021.
Telegram acknowledged receipt soon after.
Acknowledged the behaviours on June 8, 2021.
Agreed on disclosure on July 16, 2021.

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.

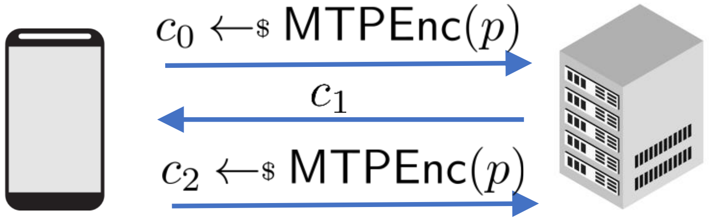
Fixes were rolled out as part of regular updates:

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

MTProto.Encrypt



1. IND-CPA attack

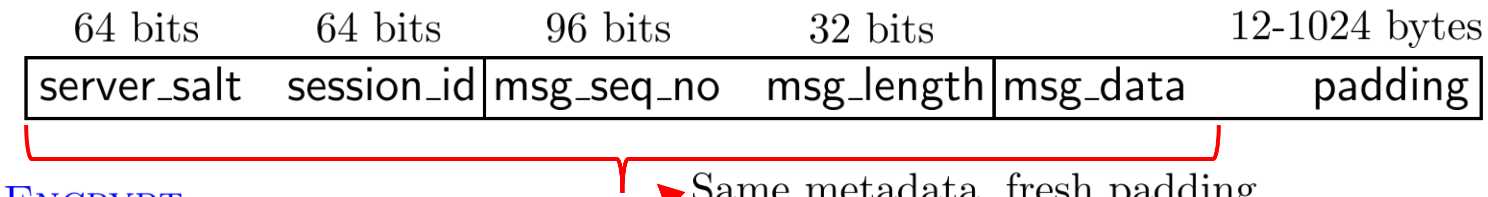


Client expects c_1 be an encryption of ACK.
Otherwise, it re-encrypts the payload.

1. Attack against IND-CPA security. // Theoretical.
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



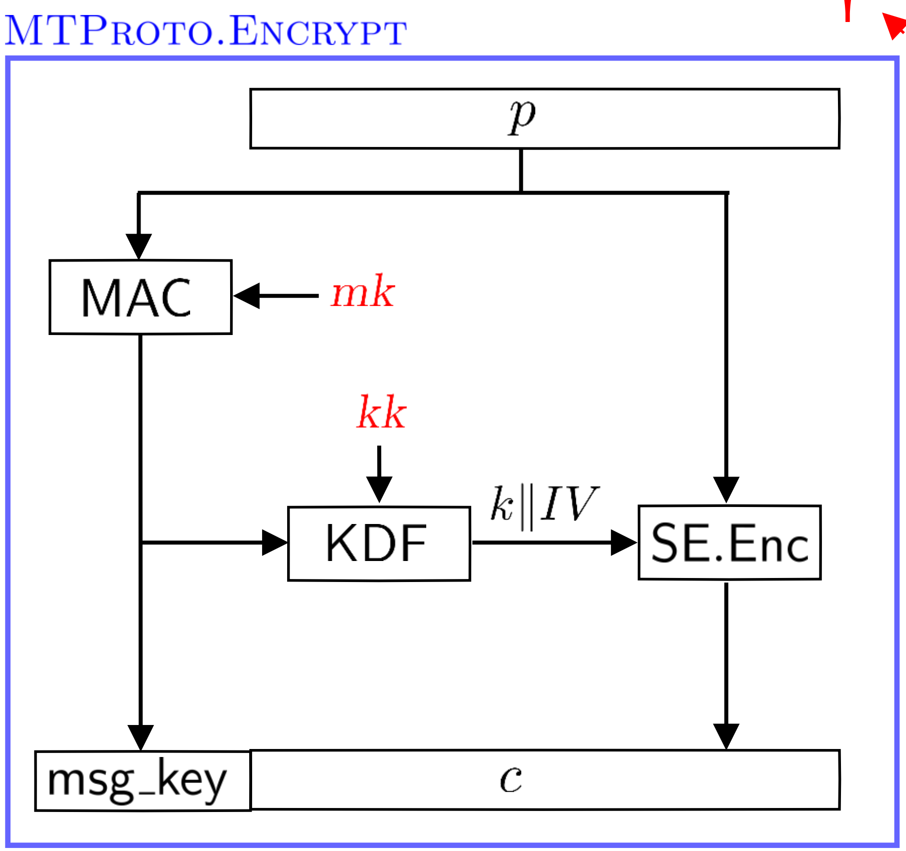
Same metadata, fresh padding.

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.

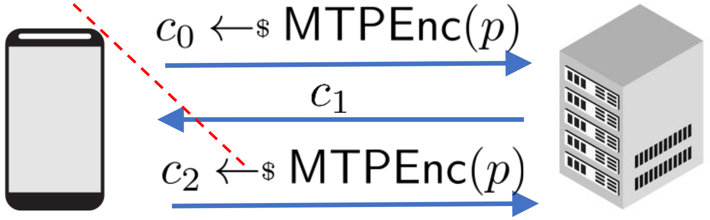
No security or bugfix releases except for immediate post-release crash fixes.
Did not wish to issue security advisories at the time of patching.
Did not commit to release dates for specific fixes.

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

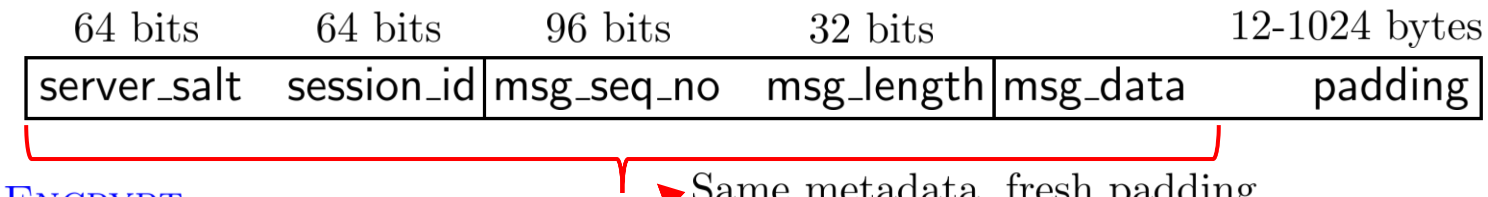


Client expects c_1 be an encryption of ACK.
Otherwise, it re-encrypts the payload.

1. Attack against IND-CPA security. // Theoretical.
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

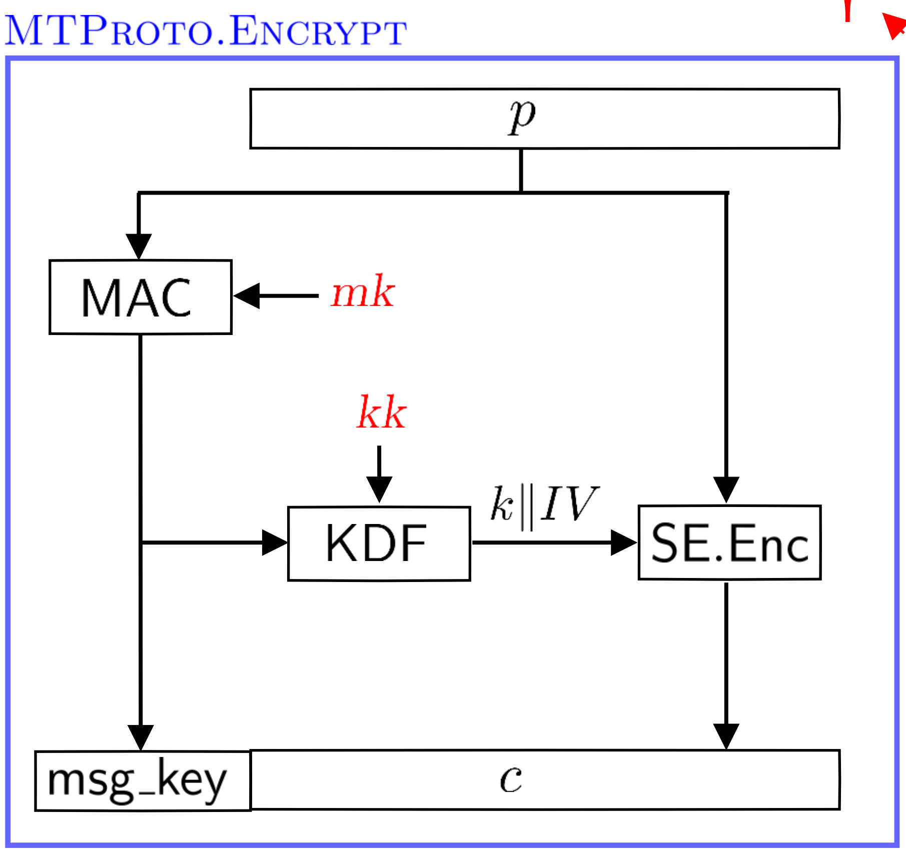


We found 4 weaknesses in MTProto.
Reported to Telegram on April 16, 2021.
Telegram acknowledged receipt soon after.
Acknowledged the behaviours on June 8, 2021.
Agreed on disclosure on July 16, 2021.

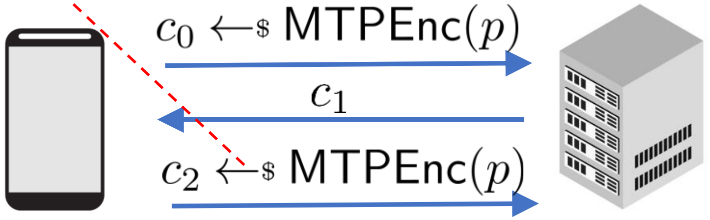
No security or bugfix releases except for immediate post-release crash fixes.
Did not wish to issue security advisories at the time of patching.
Did not commit to release dates for specific fixes.

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

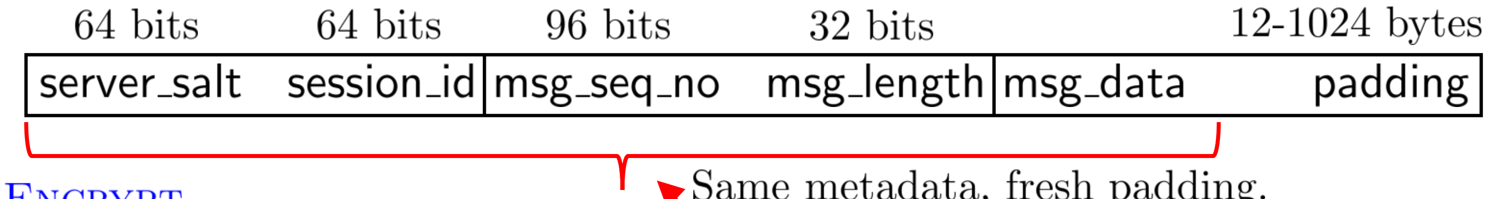


Client expects c_1 be an encryption of ACK.
Otherwise, it re-encrypts the payload.
Birthday-bound collision in msg_key causes the first 2 blocks of c_0, c_2 be the same.
IND-CPA thus breaks privacy of c_1.

1. Attack against IND-CPA security. // Theoretical.
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

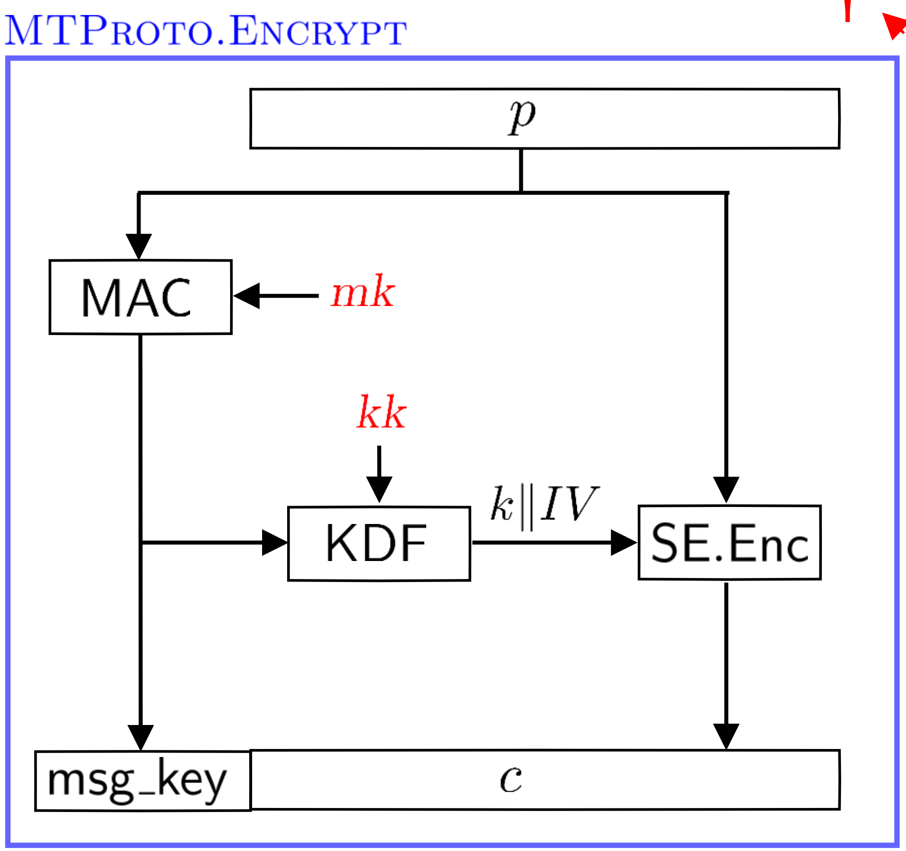


We found 4 weaknesses in MTProto.
Reported to Telegram on April 16, 2021.
Telegram acknowledged receipt soon after.
Acknowledged the behaviours on June 8, 2021.
Agreed on disclosure on July 16, 2021.

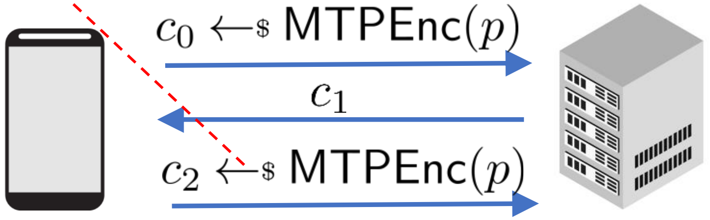
No security or bugfix releases except for immediate post-release crash fixes.
Did not wish to issue security advisories at the time of patching.
Did not commit to release dates for specific fixes.

Fixes were rolled out as part of regular updates:

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

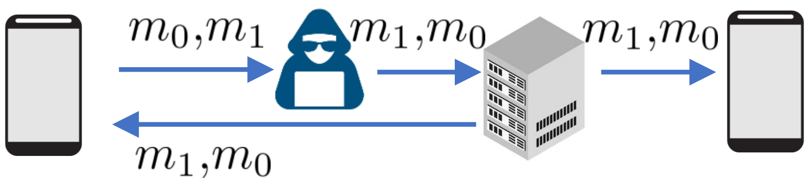


1. IND-CPA attack



Client expects c_1 be an <u>encryption</u> 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.

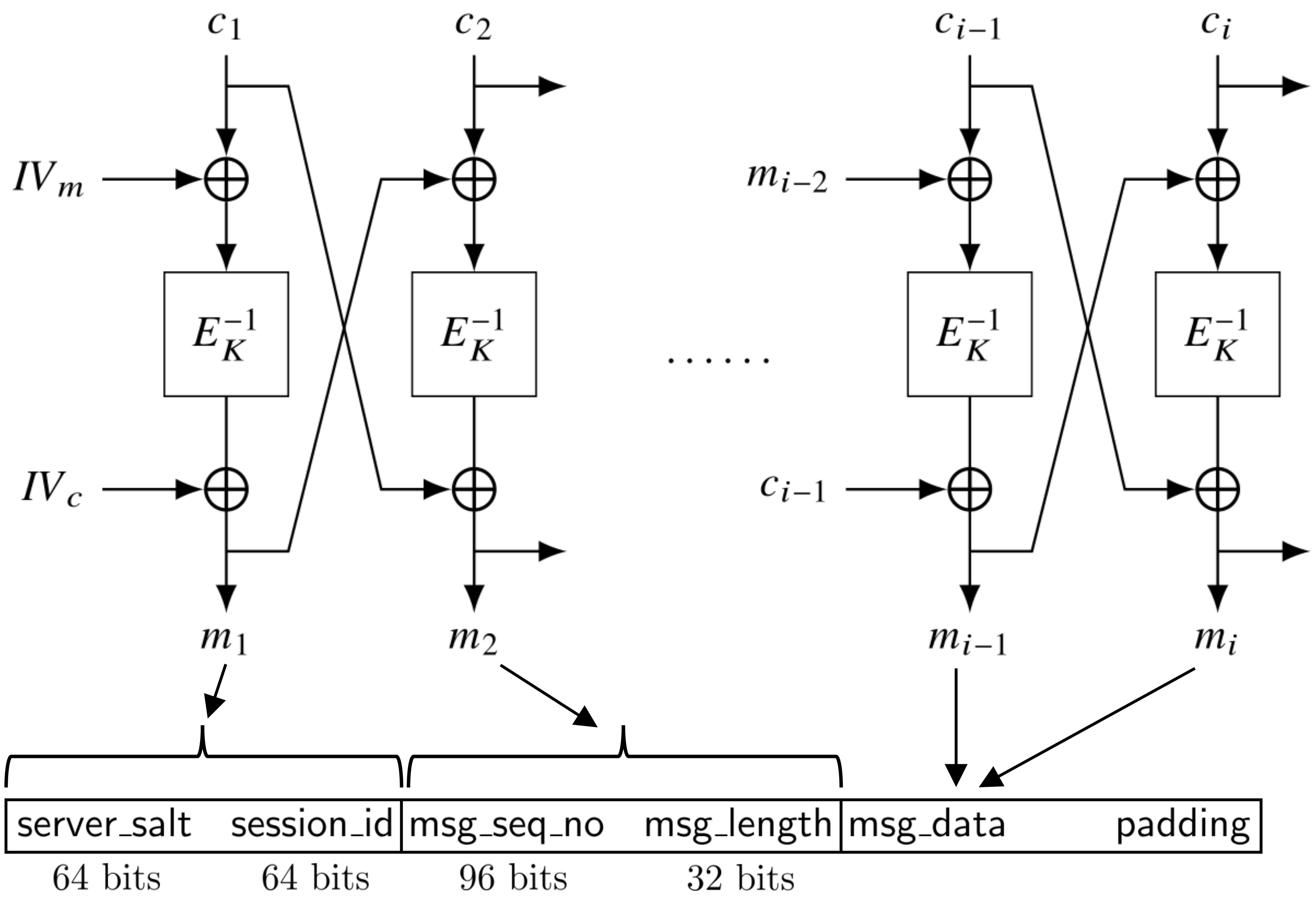
2. Message reordering attack



1. Attack against IND-CPA security. // Theoretical.
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.

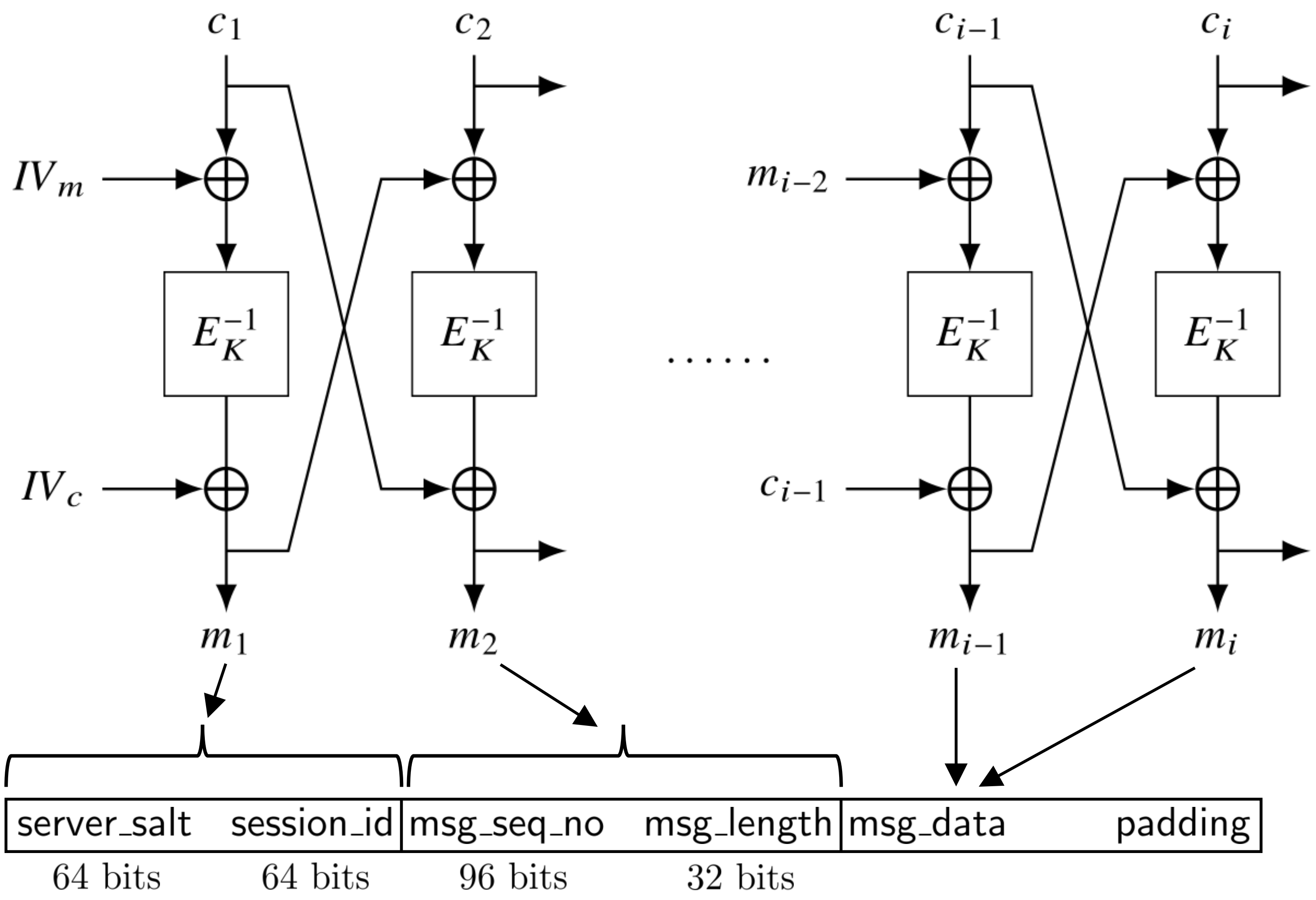
Telegram awarded a bug bounty for side-channel attacks and overall analysis.

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



supplied by attacker

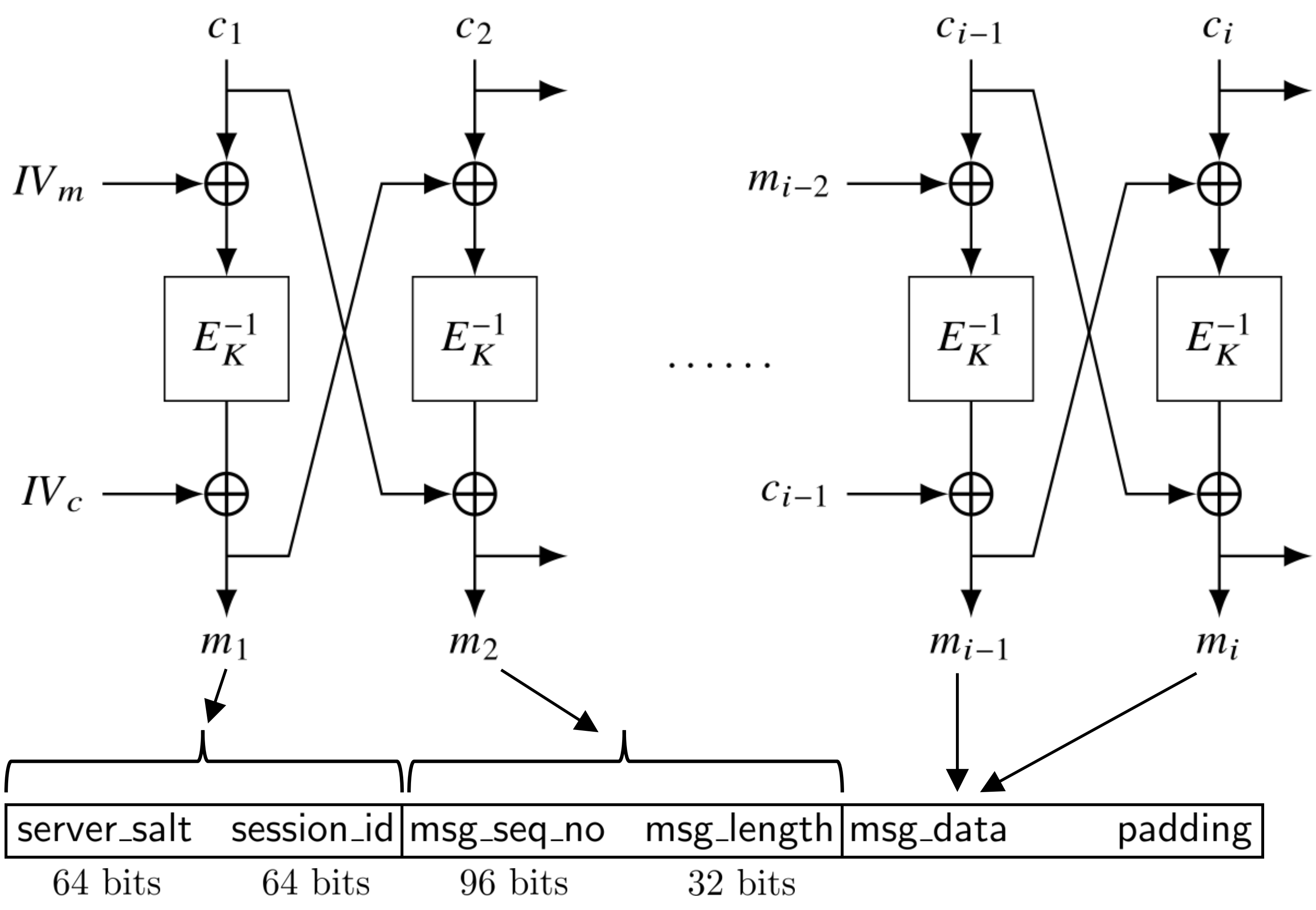
If (msg_length > length) then . . . // **Android**

Outcome of comparison depends on 32 bits on msg_length.

If comparison fails: two conditional jumps added.

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



supplied by attacker

If (msg_length > length) then . . . // **Android**

Outcome of comparison depends on 32 bits on msg_length.

If comparison fails: two conditional jumps added.

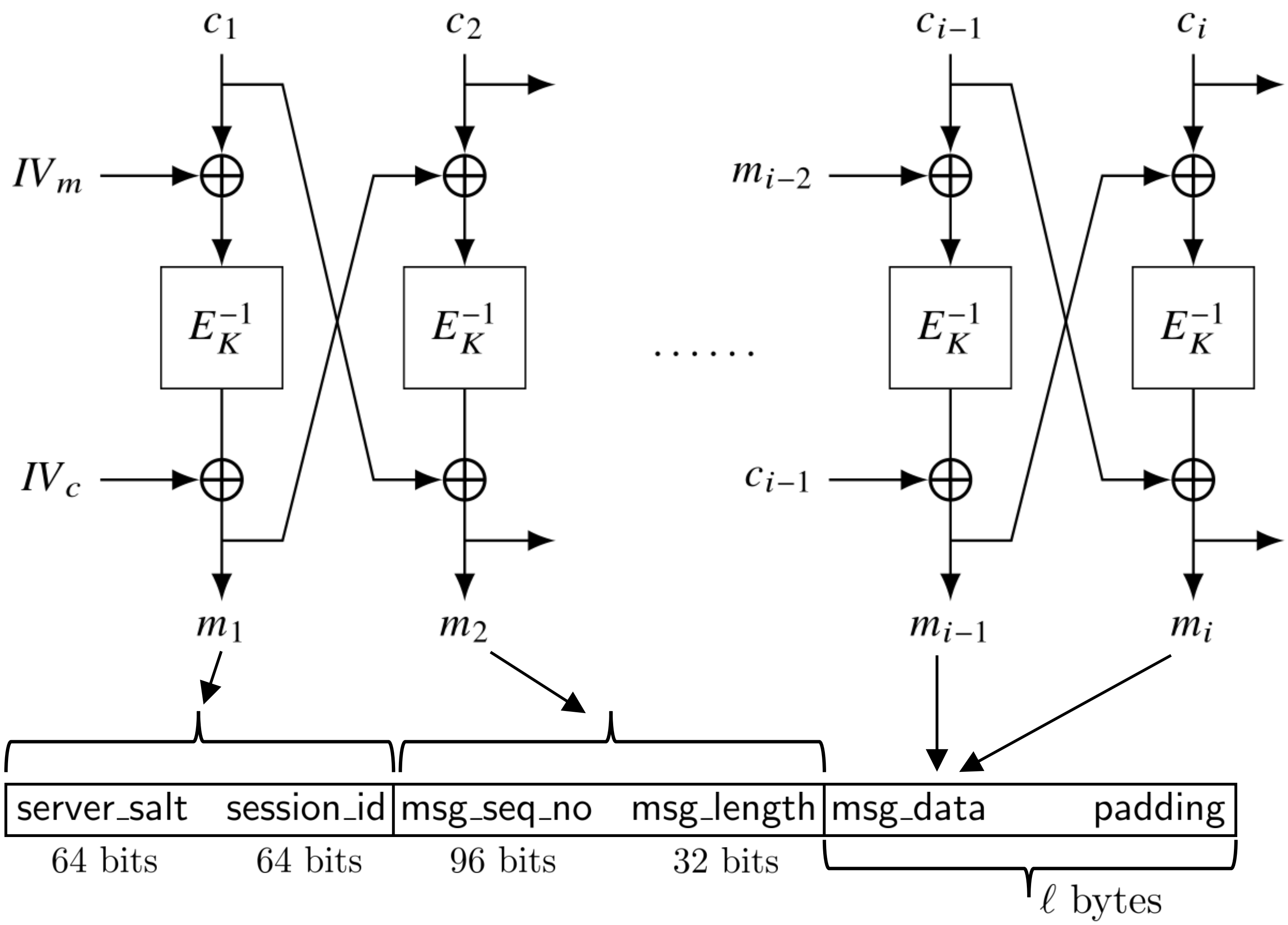
If (msg_length > 2^{24}) then . . . // **Desktop**

Outcome of comparison depends on 8 bits on 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 msg_key.

Timing Side-Channel Attacks Against Clients



supplied by attacker

If (msg_length > length) then … // **Android**

Outcome of comparison depends on 32 bits on msg_length.

If comparison fails: two conditional jumps added.

If (msg_length > 2^{24}) then … // **Desktop**

Outcome of comparison depends on 8 bits on msg_length.

If comparison fails: MAC verification is omitted.

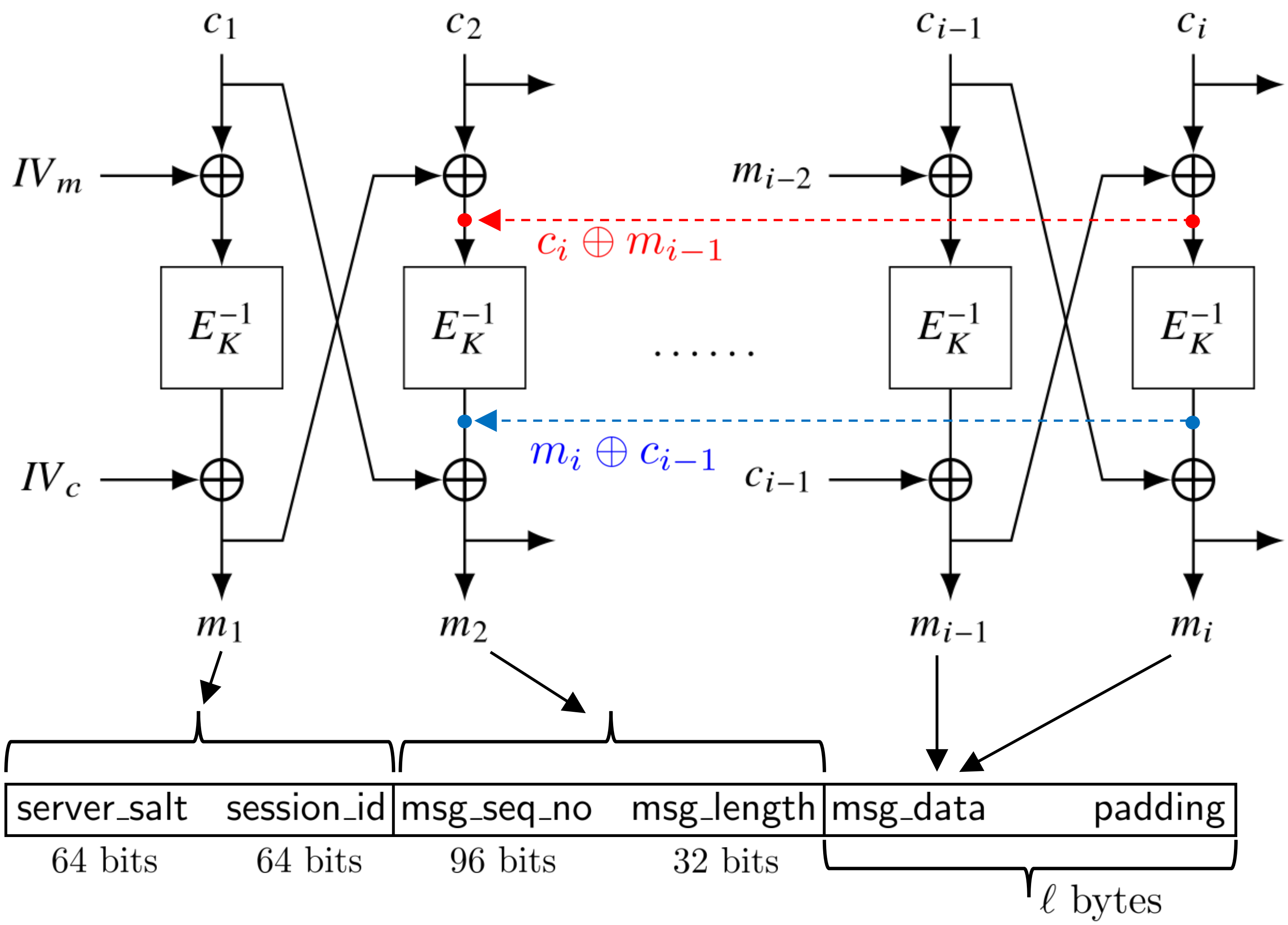
If not ($12 \le \ell -$ msg_length ≤ 1024) then … // **iOS**

Outcome of comparison depends on 32 bits on 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 msg_key.

Timing Side-Channel Attacks Against Clients



supplied by attacker

If (msg_length > length) then ... // **Android**

Outcome of comparison depends on 32 bits on msg_length.

If comparison fails: two conditional jumps added.

If (msg_length > 2^{24}) then ... // **Desktop**

Outcome of comparison depends on 8 bits on msg_length.

If comparison fails: MAC verification is omitted.

If not ($12 \le \ell -$ msg_length ≤ 1024) then ... // **iOS**

Outcome of comparison depends on 32 bits on msg_length.

If comparison fails: MAC verification takes a shorter input.

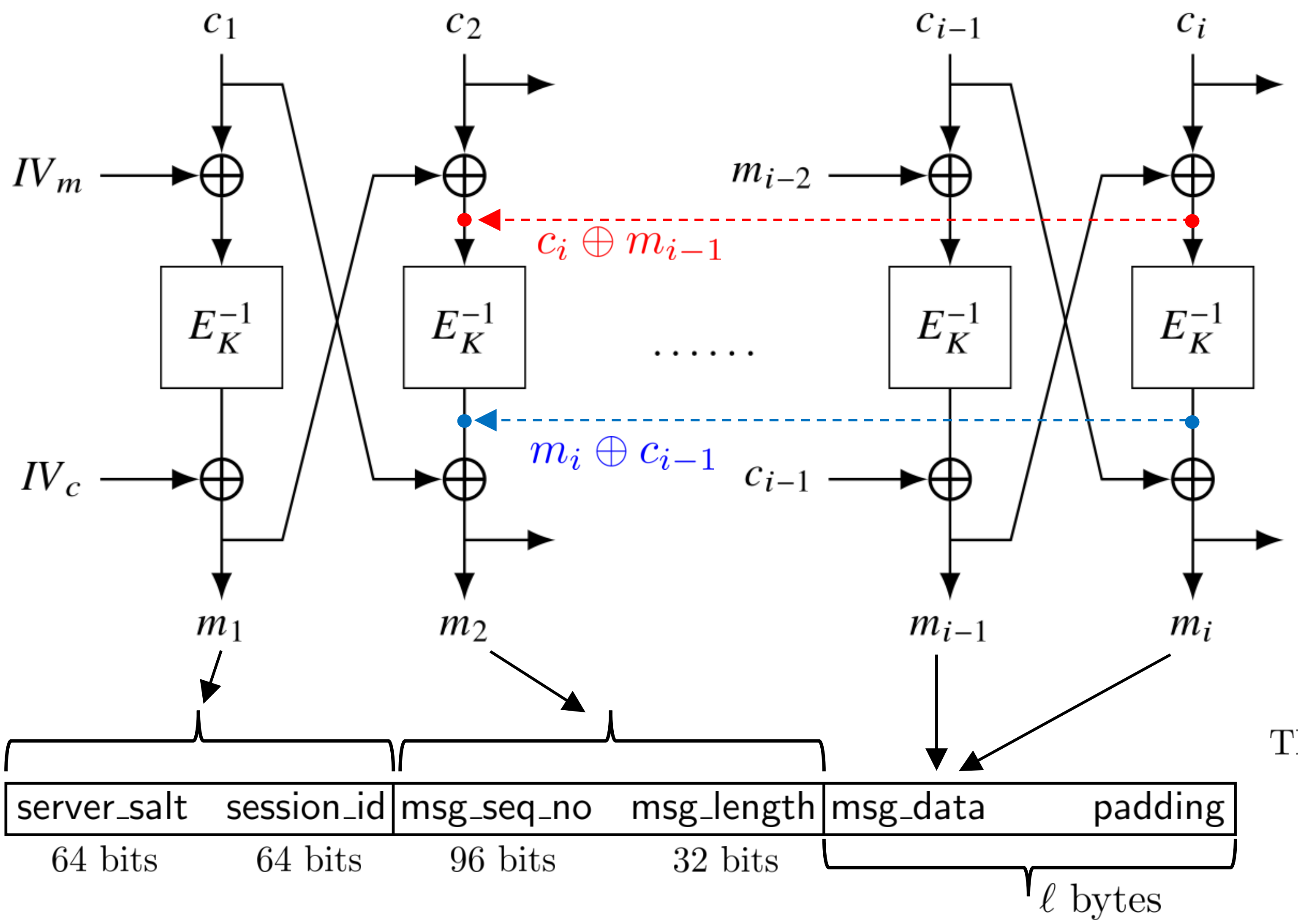
Plaintext Recovery Attacks Against SSH

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.

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



supplied by attacker

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

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

If not ($12 \leq \ell -$ msg_length ≤ 1024) then . . . // **iOS**
Outcome of comparison depends on 32 bits on msg_length.
If comparison fails: MAC verification takes a shorter input.

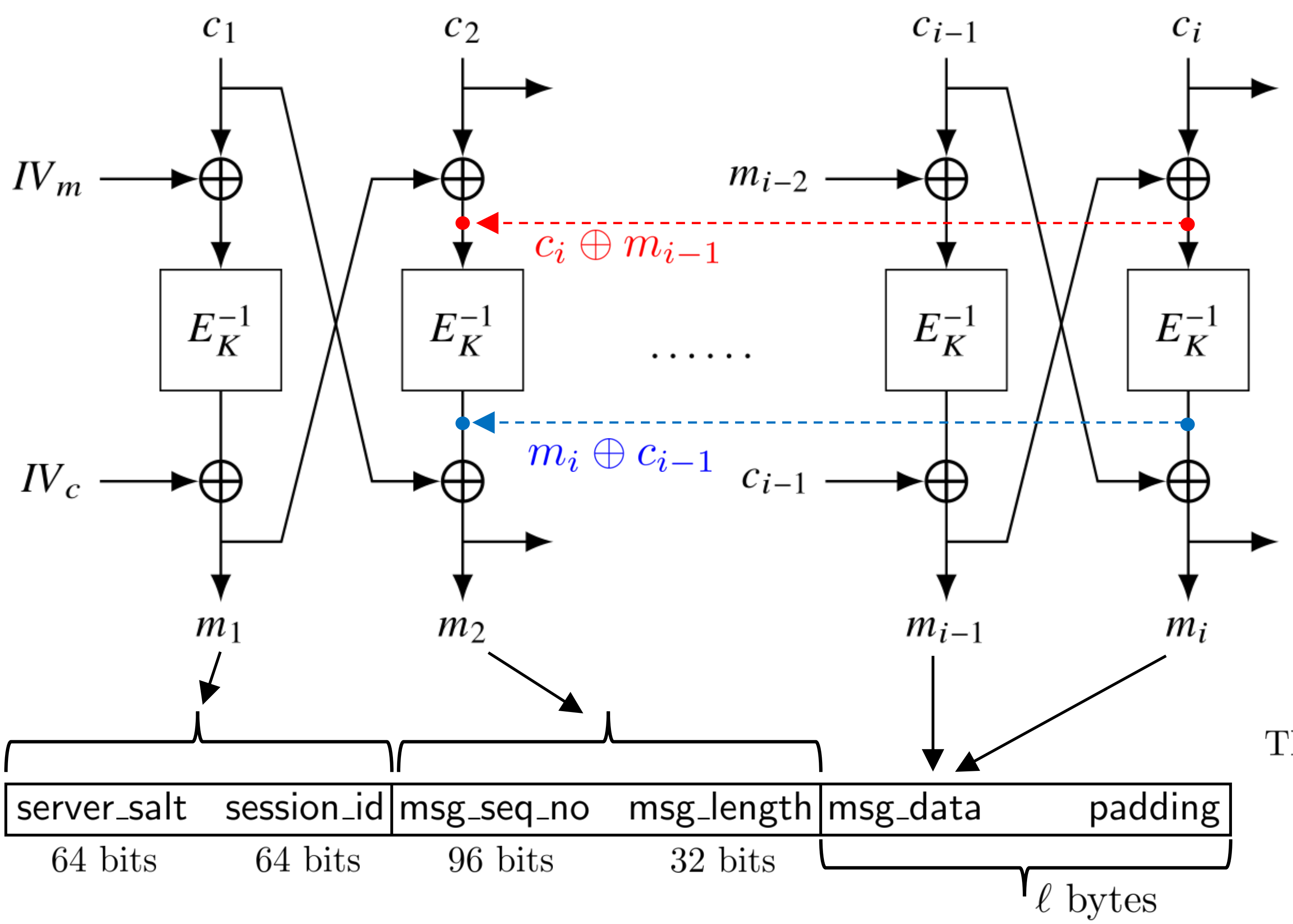
Plaintext Recovery Attacks Against SSH
Martin R. Albrecht, Kenneth G. Paterson and Gaven J. Watson

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 server_salt, session_id are secret.

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



supplied by attacker

If (msg_length > length) then ... // **Android**

Outcome of comparison depends on 32 bits on msg_length.
If comparison fails: two conditional jumps added.

If (msg_length > 2^{24}) then ... // **Desktop**

Outcome of comparison depends on 8 bits on msg_length.
If comparison fails: MAC verification is omitted.

If not ($12 \leq \ell -$ msg_length ≤ 1024) then ... // **iOS**

Outcome of comparison depends on 32 bits on 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.

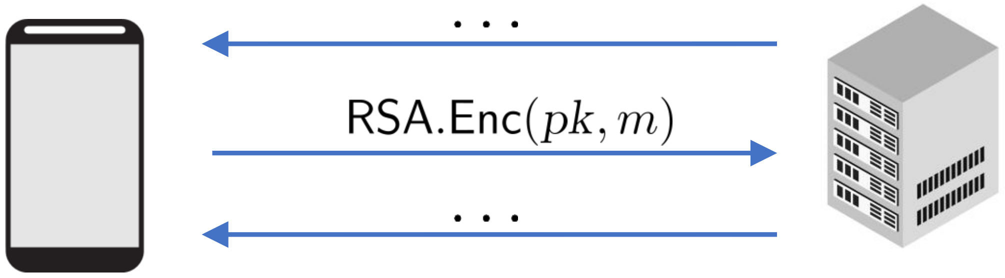
The attack fails because server_salt, session_id are secret.

3 official clients (Android; Desktop; iOS) ran a sanity check on a decrypted payload prior to verifying its hash 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.

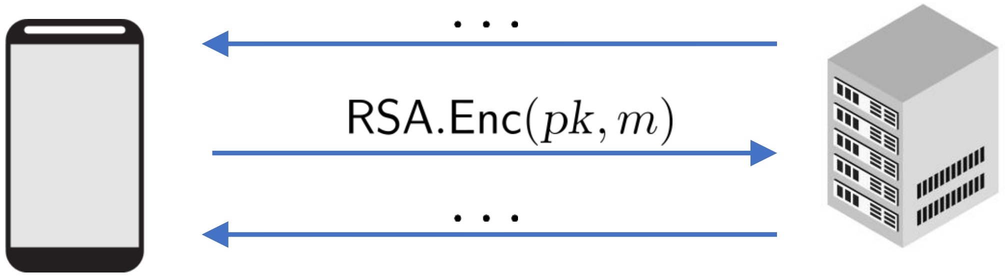


Telegram uses textbook **RSA** encryption.

$m :=$ SHA-1(data)$\|$data$\|$padding

Timing Side-Channel Attack Against Servers

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



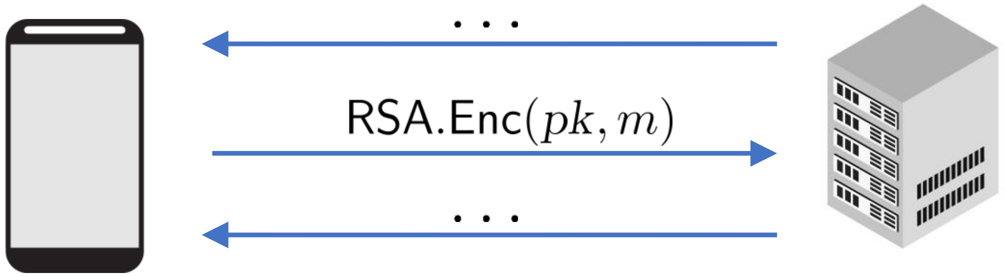
Telegram uses textbook **RSA** encryption.

$m :=$ SHA-1(data)||data||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.

$m :=$ SHA-1(data)||data||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.

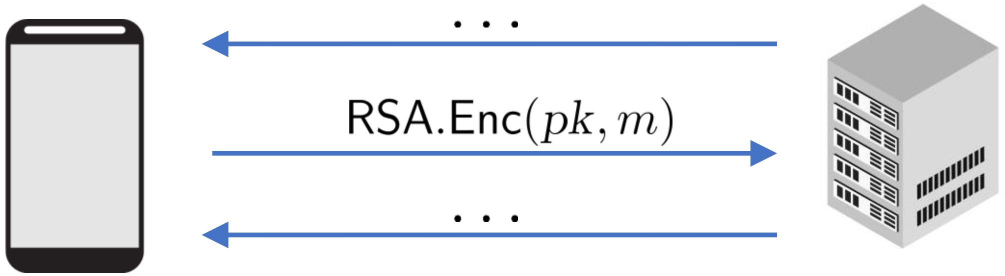
Can use data to recover server_salt and session_id.
Can use data to run a **MitM** attack against the (encrypted) **DH** exchange.

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.



Telegram uses textbook **RSA** encryption.

$m :=$ SHA-1(data)||data||padding

"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)

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.

Future Work

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.

Thanks!

More information at:
https://mtpsym.github.io/