Applied Cryptography Spring Semester 2023 Lecture 37

Kenny Paterson (@kennyog)

Applied Cryptography Group

https://appliedcrypto.ethz.ch/

Overview of this lecture

- Telegram

Thanks to Lenka Mareková, Igors Stepanovs and Theo von Arx for slides.

Telegram

Telegram

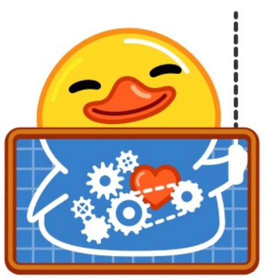
a new era of messaging

Telegram for **iPhone** / **iPad**

700 Million Users and Telegram Premium

Telegram now has over 700 million monthly active users. Today we're launching Telegram Premium – a subscription that lets you support Telegram's continued development and gives access…

Jun 21, 2022



Open

Telegram has an open API and source code free for everyone.

Private

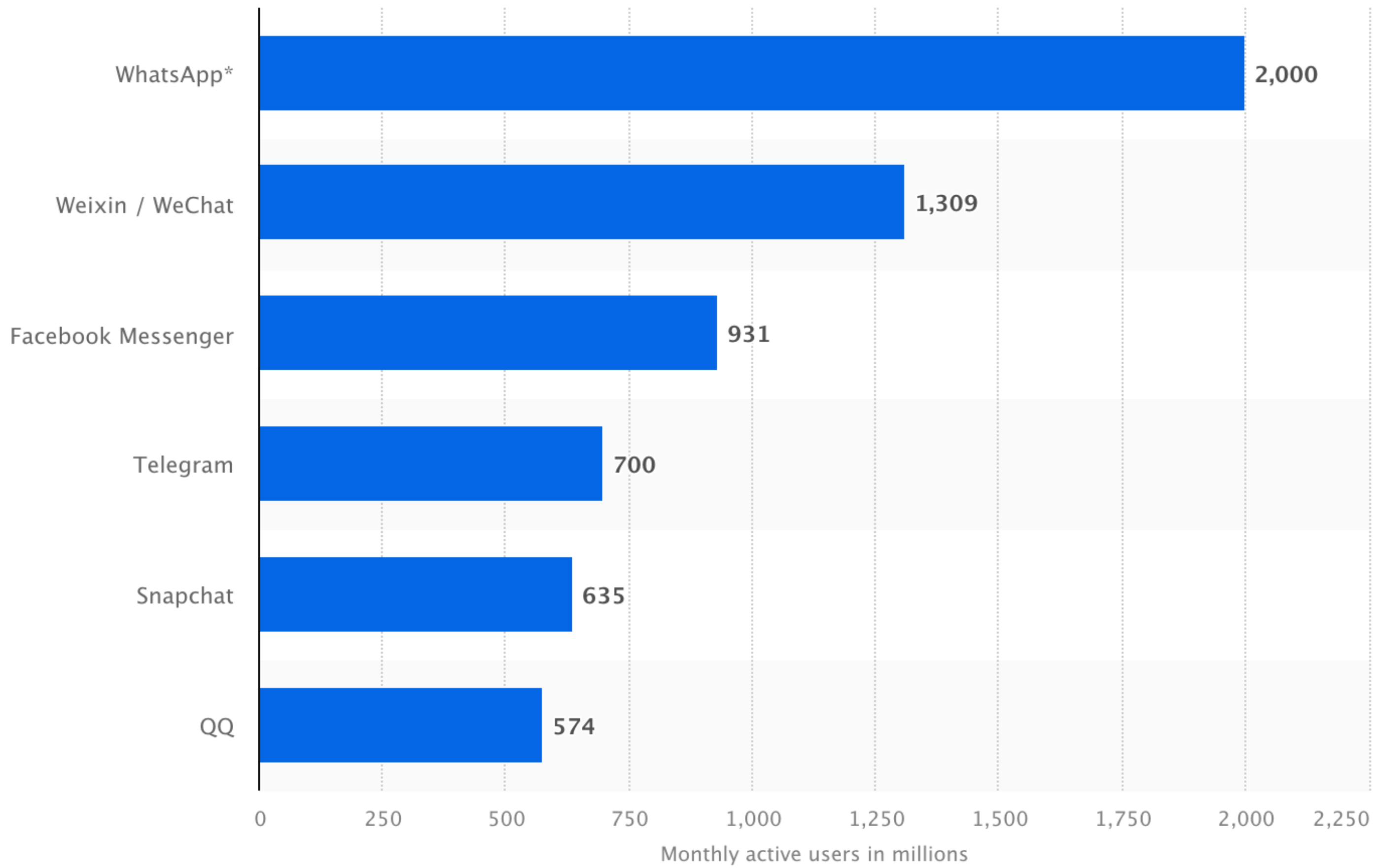
Telegram messages are heavily encrypted and can self-destruct.

Secure

Telegram keeps your messages safe from hacker attacks.

Source: https://telegram.org/

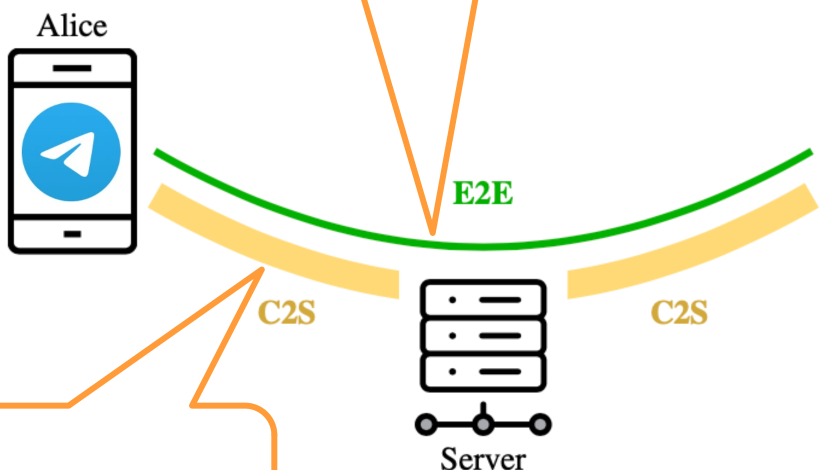
Telegram's Popularity (January 2023)



https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/

Telegram Architecture

E2E not enabled by default and not available at all for groups



MTProto 2.0 protocol

Telegram: MTProto 2.0 vs TLS

So why not just use TLS?

https://core.telegram.org/techfaq

Telegram FAQ

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.

The Design of MTProto 2.0: Main Key Exchange

MTP-KE

Client		**Server** (has (pk, sk))
$n \leftarrow\$ \{0,1\}^{128}$		
	$\xrightarrow{\quad n \quad}$	
		$s \leftarrow\$ \{0,1\}^{128}$ gen. p', q'
	$\xleftarrow{\quad n, s, p' \cdot q', \mathcal{F} \quad}$	
factor $p' \cdot q'$; pick $f \in \mathcal{F}$ for pk $n' \leftarrow\$ \{0,1\}^{256}$ $m_0 \leftarrow p' \cdot q', p', q', n, s, n', dc$ $c_0 \leftarrow \mathsf{TOAEP}^{+}.\mathsf{Enc}(pk, m_0)$		
	$\xrightarrow{\quad n, s, p', q', f, c_0 \quad}$	
		$a \leftarrow\$ \{0,1\}^{2048}$ $k, iv \leftarrow \mathsf{SHA\text{-}1\text{-}KDF}^{*}(n', s)$ $m_1 \leftarrow n, s, g, p, g^a \bmod p, servertime$ $c_1 \leftarrow \mathsf{IGE\text{-}MAC}^{*}.\mathsf{Eval}(k \parallel iv, m_1)$
	$\xleftarrow{\quad n, s, c_1 \quad}$	
$b \leftarrow\$ \{0,1\}^{2048}$ $k, iv \leftarrow \mathsf{SHA\text{-}1\text{-}KDF}^{*}(n', s)$ $m_2 \leftarrow n, s, rid, g^b \bmod p$ $c_2 \leftarrow \mathsf{IGE\text{-}MAC}^{*}.\mathsf{Eval}(k \parallel iv, m_2)$		
	$\xrightarrow{\quad n, s, c_2 \quad}$	
		$ak \leftarrow (g^b)^a \bmod p$ $aid \leftarrow \mathsf{SHA\text{-}1}(ak)[12:20]$ **if** aid is unique **then** $\quad h \leftarrow \mathsf{SHA\text{-}1\text{-}MAC}^{*}(ak, n' \parallel 01)$
	$\xleftarrow{\quad n, s, h \quad}$	
$ak \leftarrow (g^a)^b \bmod p$ $h \stackrel{?}{=} \mathsf{SHA\text{-}1\text{-}MAC}^{*}(ak, n' \parallel 01)$		

- MTProto 2.0 uses a bespoke key-exchange protocol.
- Diffie-Hellman in a mod *p* group.
- Authentication of server provided by ability to perform RSA decryption (cf. old TLS handshake).
- RSA uses an unanalysed variant of RSA-OAEP.
- Also relies on unanalysed MAC-like properties of IGE encryption.
- Many round trips – delivery time?
- Robustness, aka security?

Telegram: Prior MTProto 2.0 Analyses

- **2013** — **Telegram** launched with MTProto 1.0.
- **2016** — MTProto 1.0 is not CCA-secure [JO16].
- **2017** — Input validation bug (message replay) [SK17].
  Telegram released MTProto 2.0.
- **2018** — Input validation bug in key exchange [K18].
- **2020** — MTProto 2.0 secure in symbolic model [MV20].
  (assuming ideal building blocks)

Telegram

Four Attacks and a Proof for Telegram

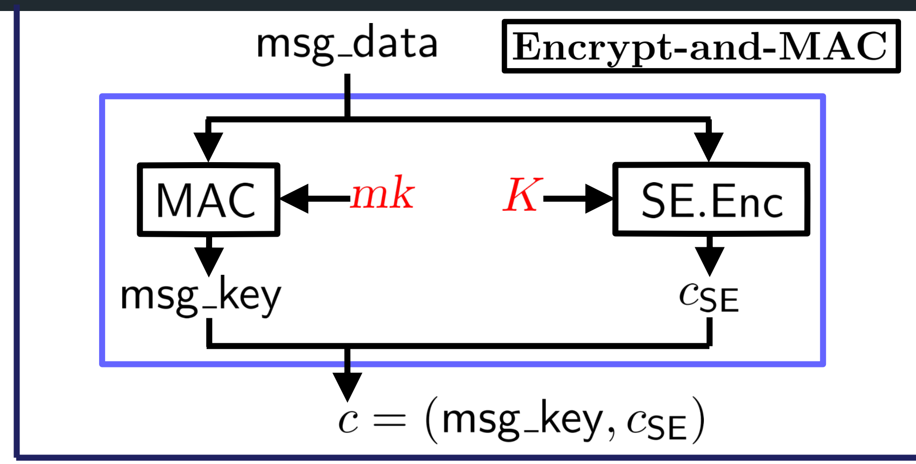
Martin R. Albrecht*, Lenka Mareková*, Kenneth G. Paterson† and Igors Stepanovs†
*Information Security Group, Royal Holloway, University of London, {martin.albrecht,lenka.marekova.2018}@rhul.ac.uk
†Applied Cryptography Group, ETH Zurich, {kenny.paterson,istepanovs}@inf.ethz.ch

IEEE Security and Privacy Symposium 2022

Distinguished paper award

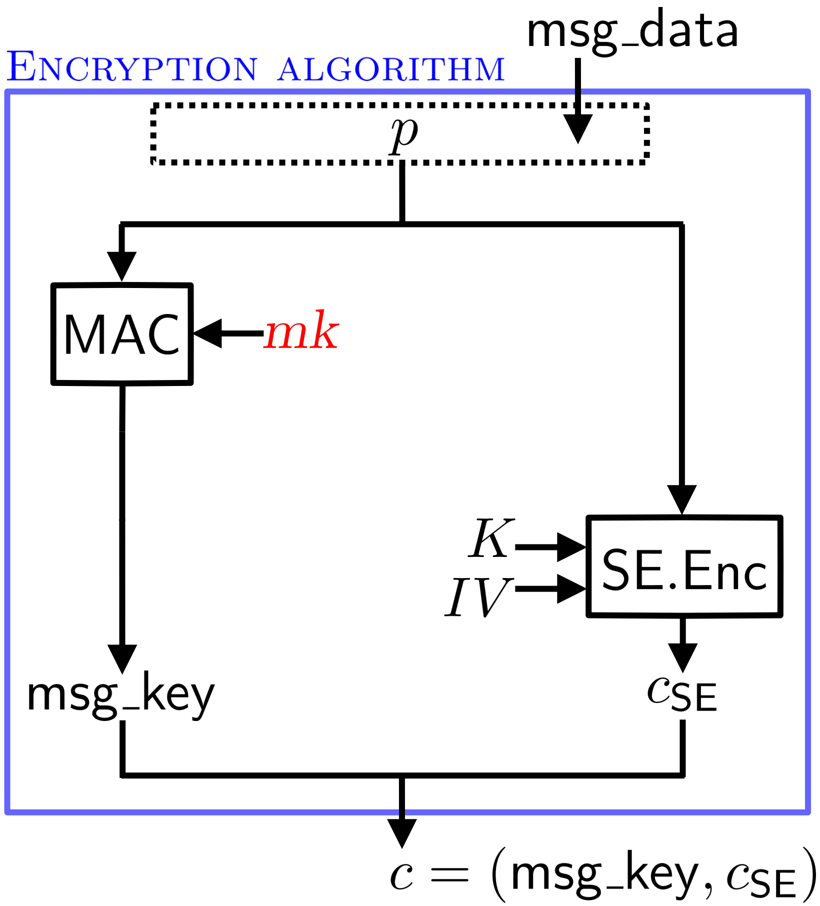
https://mtpsym.github.io/

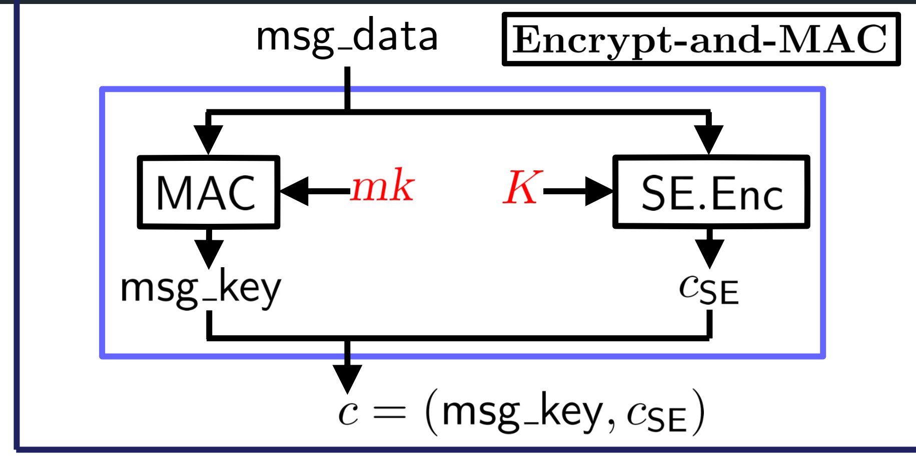
The Design of MTProto 2.0: Symmetric Crypto



MAC – Message Authentication Code
SE – Symmetric Encryption Scheme

The Design of MTProto 2.0: Symmetric Crypto



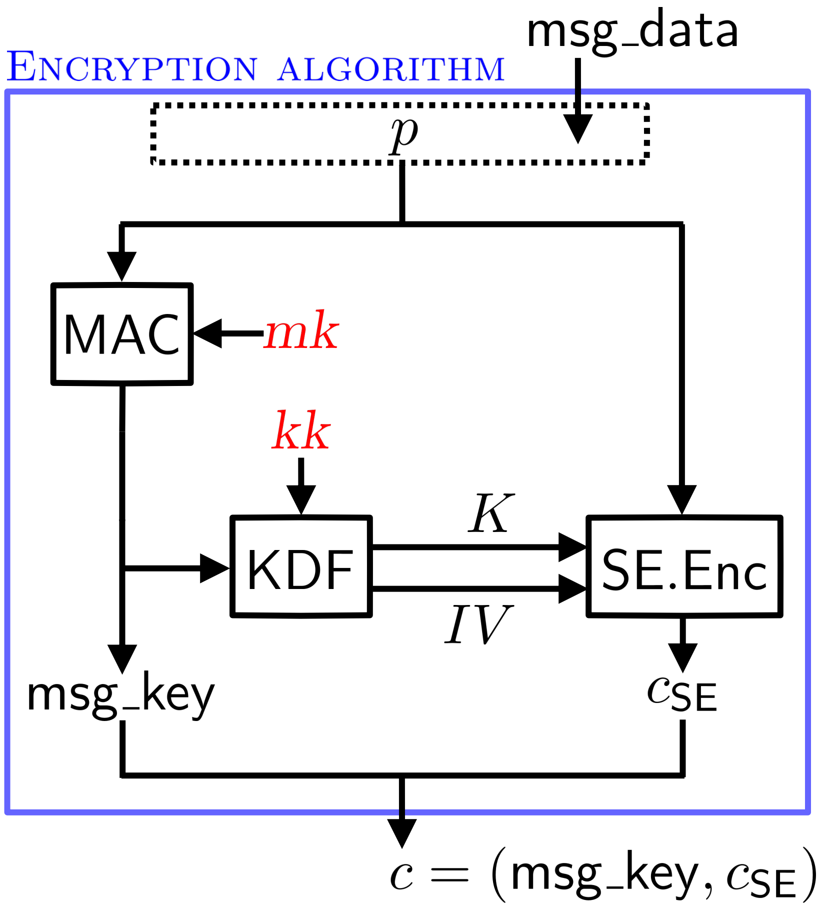


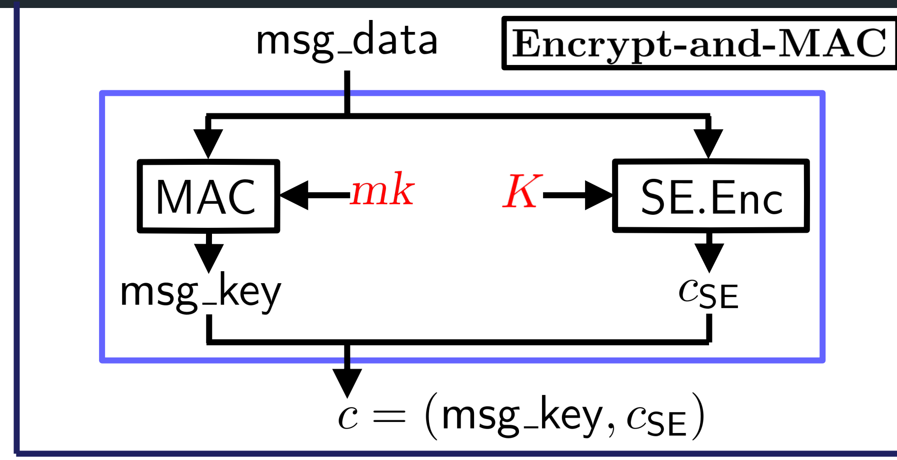
MAC – Message Authentication Code
SE – Symmetric Encryption Scheme

payload p

Random values:	server_salt	64 bits
	session_id	64 bits
Message sequence number:	msg_seq_no	96 bits
Message length:	msg_length	32 bits
Message body:	msg_data	$\leq$ 16 MB
Random padding:	padding	12-1024 bytes

The Design of MTProto 2.0: Symmetric Crypto



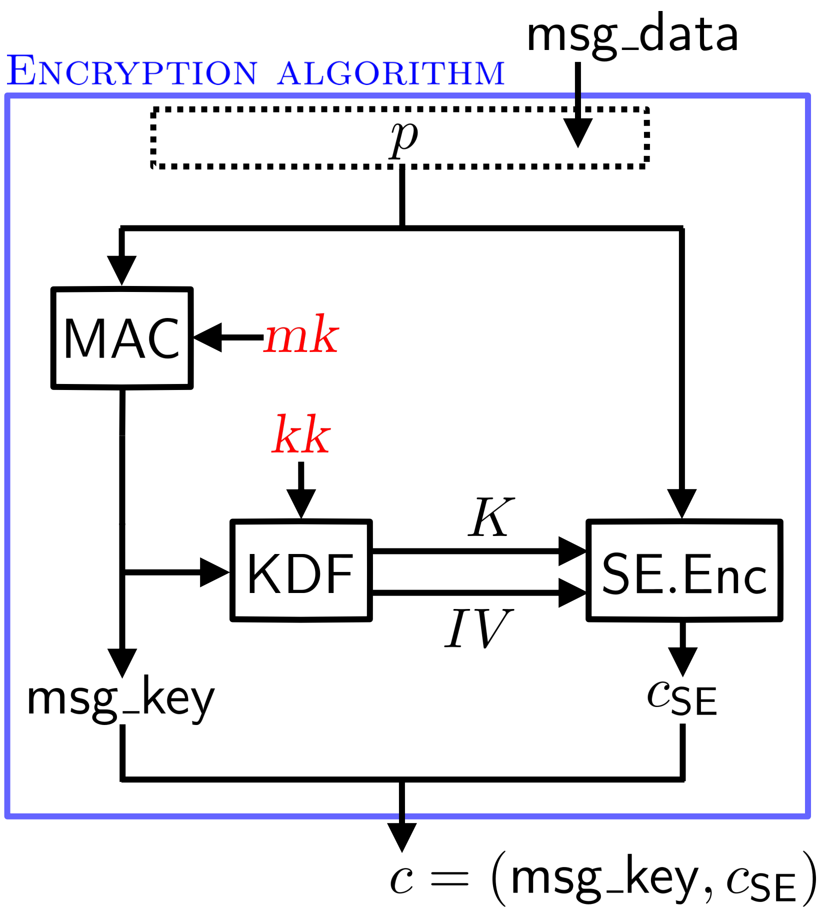


MAC – Message Authentication Code
SE – Symmetric Encryption Scheme
KDF – Key Derivation Function

payload p

Random values:	server_salt	64 bits
	session_id	64 bits
Message sequence number:	msg_seq_no	96 bits
Message length:	msg_length	32 bits
Message body:	msg_data	$\leq$ 16 MB
Random padding:	padding	12-1024 bytes

The Design of MTProto 2.0: Symmetric Crypto

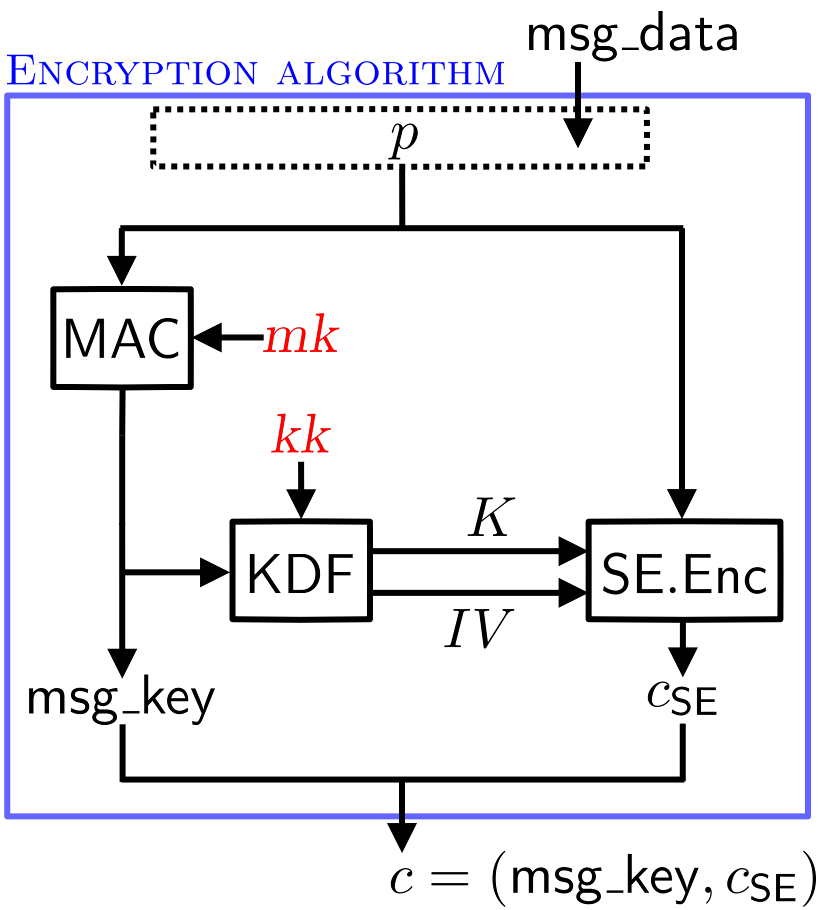


MTProto defines **ad-hoc** MAC and KDF schemes.

$\underline{\mathsf{MAC}(mk, p)}$
$\mathsf{msg\_key} \leftarrow \mathsf{SHA\text{-}256}(mk \| p)[64 : 192]$
Return $\mathsf{msg\_key}$

$\underline{\mathsf{KDF}(kk, \mathsf{msg\_key})}$
$(kk_0, kk_1) \leftarrow kk$
$K \leftarrow \mathsf{SHA\text{-}256}(\mathsf{msg\_key} \| kk_0)$
$IV \leftarrow \mathsf{SHA\text{-}256}(kk_1 \| \mathsf{msg\_key})$
Return K, IV

The Design of MTProto 2.0: Symmetric Crypto



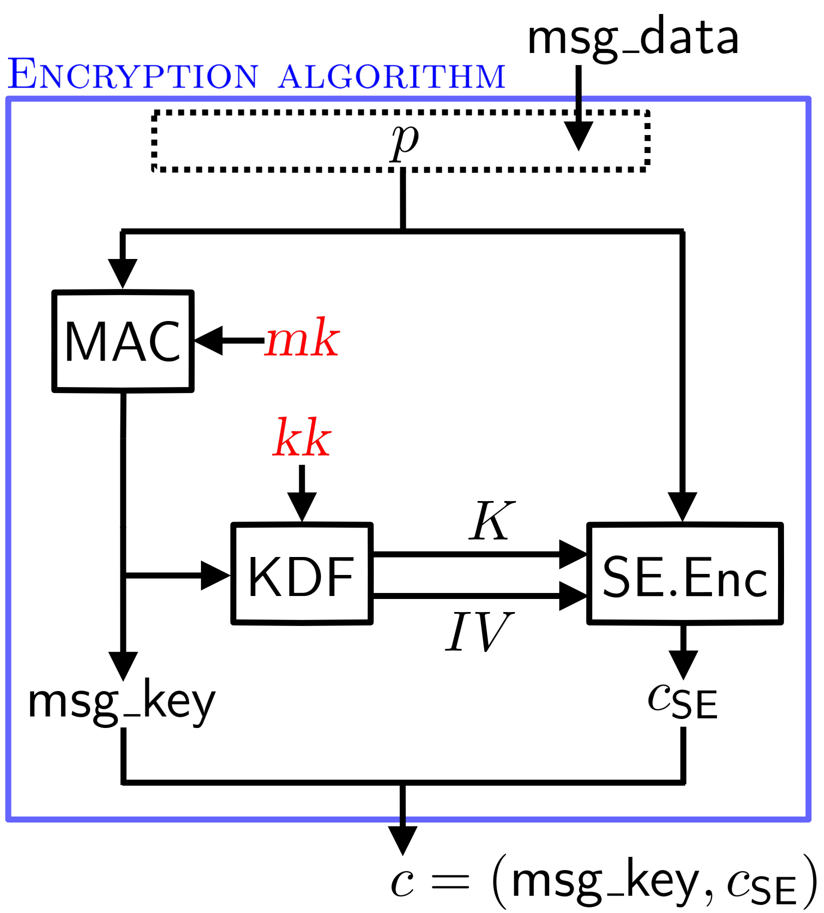
MTProto defines **ad-hoc** MAC and KDF schemes.

$\underline{\mathsf{MAC}(mk, p)}$
$\mathsf{msg\_key} \leftarrow \mathsf{SHA\text{-}256}(mk \| p)[64:192]$
Return $\mathsf{msg\_key}$

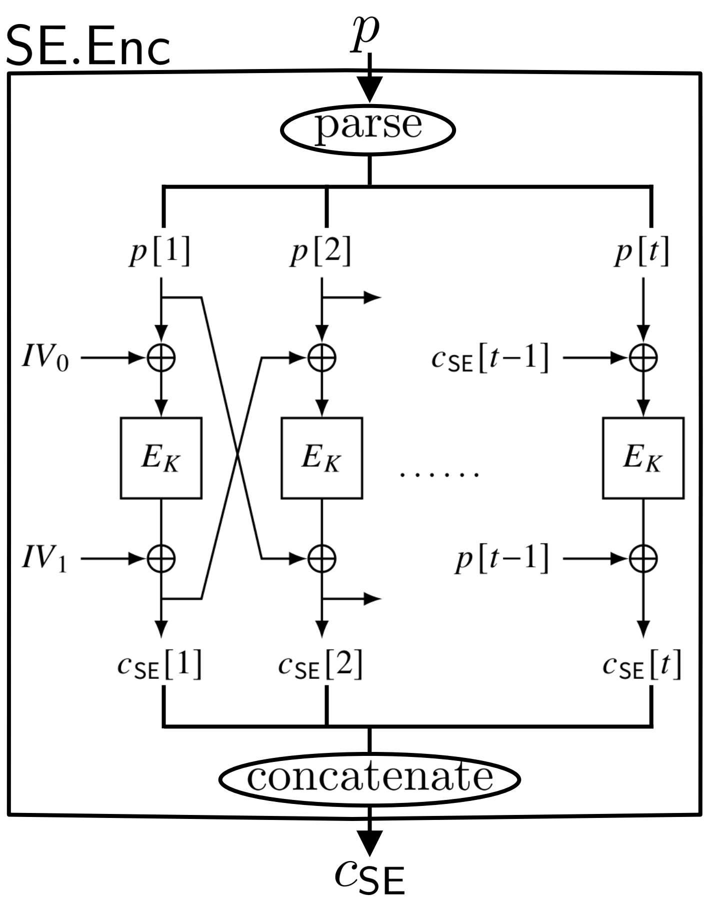
$\underline{\mathsf{KDF}(kk, \mathsf{msg\_key})}$
$(kk_0, kk_1) \leftarrow kk$
$K \leftarrow \mathsf{SHA\text{-}256}(\mathsf{msg\_key} \| kk_0)$
$IV \leftarrow \mathsf{SHA\text{-}256}(kk_1 \| \mathsf{msg\_key})$
Return K, IV

What about length extension attacks on the MAC?

The Design of MTProto 2.0: Symmetric Crypto

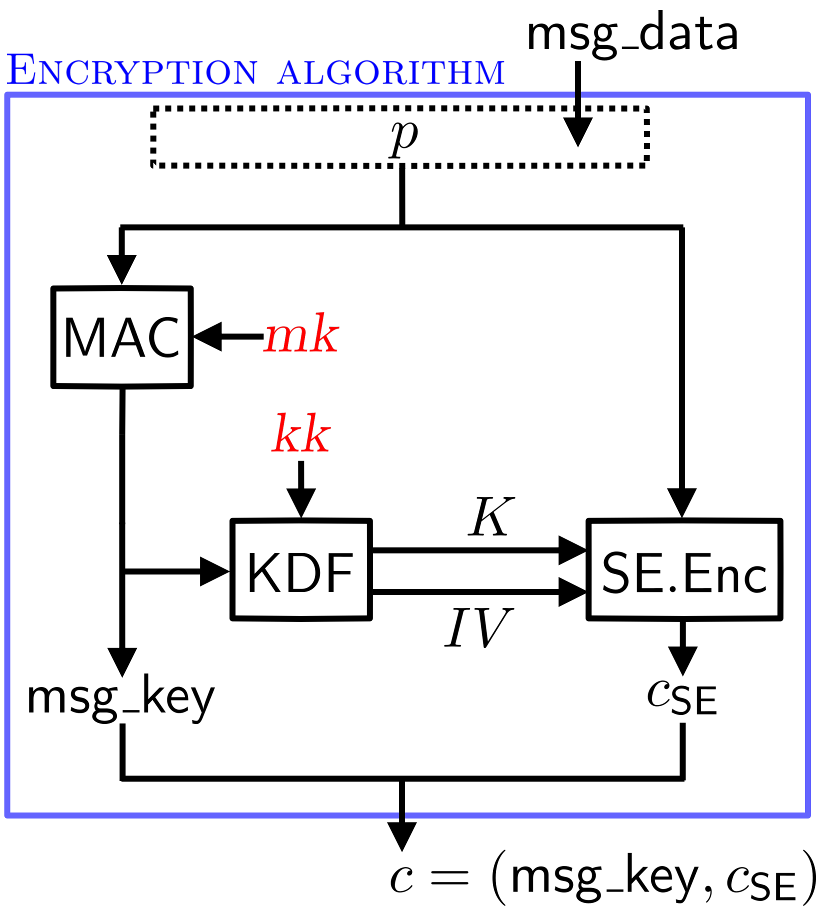


Infinite Garble Extension (IGE)
block cipher mode of operations



Not commonly used and not well studied.

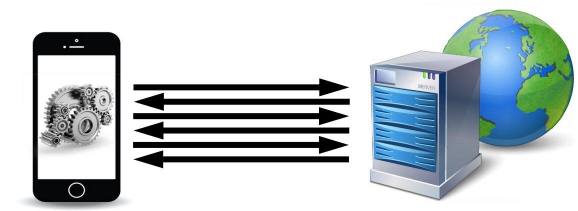
The Design of MTProto 2.0: Symmetric Crypto



How are mk and kk derived?

The Design of MTProto 2.0: Symmetric Crypto

MTProto uses Diffie-Hellman key exchange to agree on a raw shared secret g^{xy}.



Keys used for **client** $\rightarrow$ **server** encryption.

32 bits | 96 bits | 1088 bits

kk_0 (288 bits) | kk_1 (288 bits) | mk (256 bits)

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

kk_0 (288 bits) | kk_1 (288 bits) | mk (256 bits)

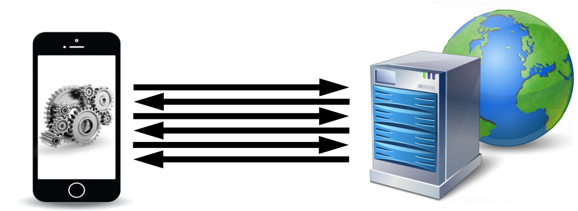
64 bits | 32 bits | 96 bits | 1024 bits

Keys used for **server** $\rightarrow$ **client** encryption.

Why overlap the key bits?

The Design of MTProto 2.0: Symmetric Crypto

MTProto uses Diffie-Hellman key exchange to agree on a raw shared secret g^{xy}.



Keys used for **client** $\rightarrow$ **server** encryption.

32 bits | 96 bits | 1088 bits

kk_0 (288 bits) | kk_1 (288 bits) | mk (256 bits)

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

kk_0 (288 bits) | kk_1 (288 bits) | mk (256 bits)

64 bits | 32 bits | 96 bits | 1024 bits

Keys used for **server** $\rightarrow$ **client** encryption.

Four Attacks Against MTProto2.0

April 16, 2021 — We reported 4 vulnerabilities to **Telegram**.

April 22, 2021 — **Telegram** confirmed the receipt of our e-mail.

June 08, 2021 — **Telegram** acknowledged the reported behaviours.

July 16, 2021 — Public disclosure (mutually agreed date).

2021 — **Telegram** awarded bug bounty for attacks and analysis.

All vulnerabilities fixed as of
7.8.1 for **Android**
7.8.3 for **iOS**
2.8.8 for **Desktop**

Telegram informed us that they

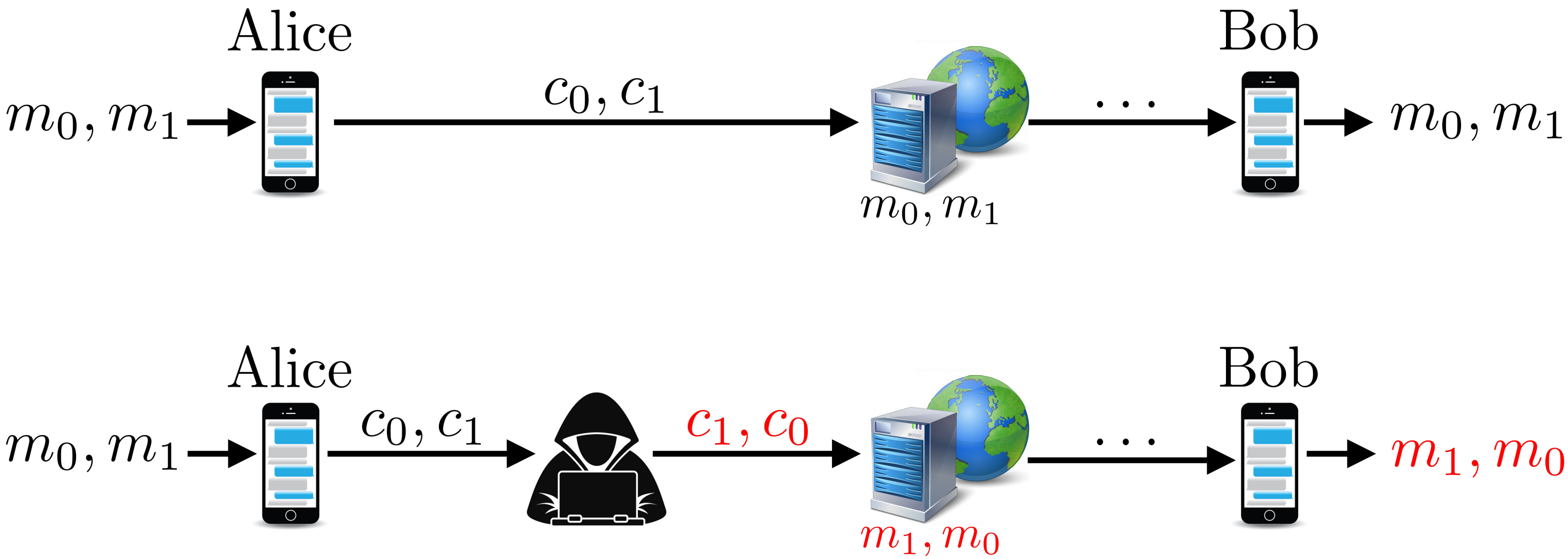
... do no security/bugfix releases except for post-release crash fixes.

(could not commit to release dates for specific fixes)

(fixes were rolled out as part of regular updates)

... did not wish to issue security advisories at the time of patching.

Message Reordering Attack



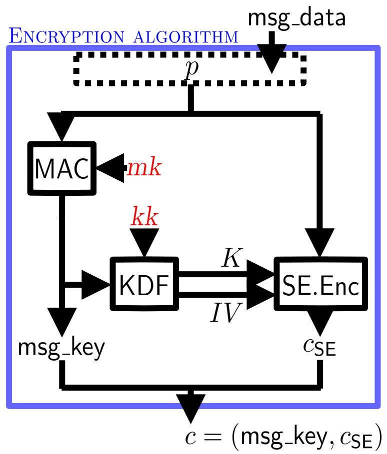
Technically trivial. Easy to exploit.

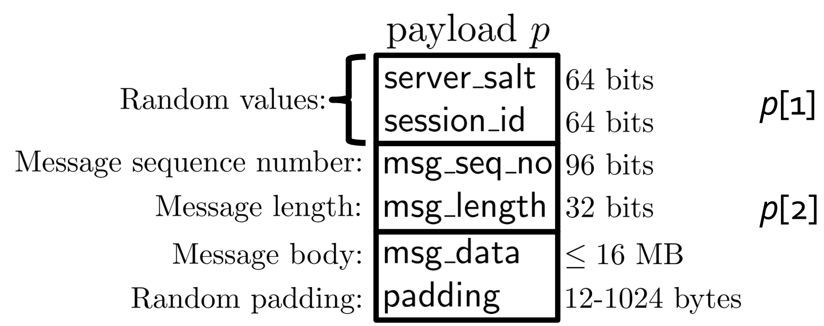
Further, more theoretical attacks

1. Plaintext recovering timing attack against E&M construction in MTProto 2.0 based on sanity-checking of length field.
2. Timing attack against MTProto 2.0 key exchange protocol allowing recovery of server_salt and message_id (needed for first attack) and breaking server authentication.
3. IND-CPA attack against MTProto 2.0's message retransmission feature.

All three attacks are more theoretical than practical but illustrate the fragility of the MTProto 2.0 design.

Details of Timing Attack on MTProto 2.0 E&M

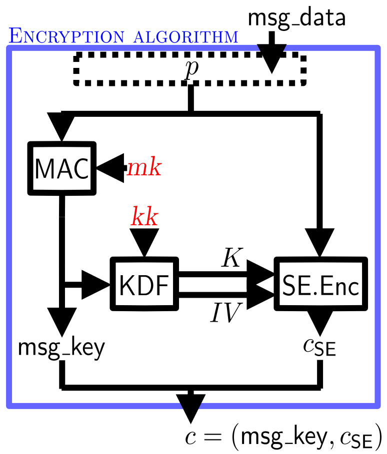


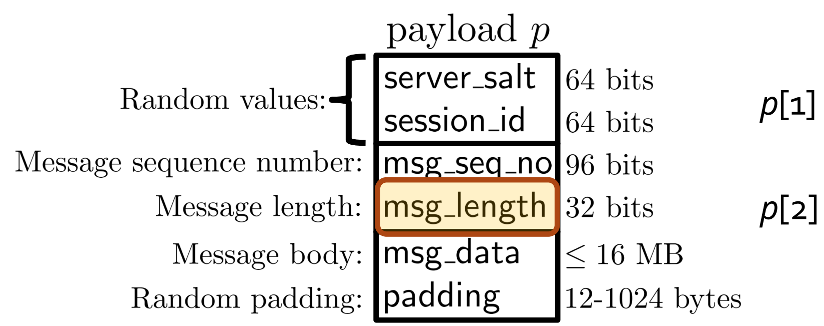


<u>Ideal decryption process:</u>

1. Parse *c* as (`msg_key`, c_{SE}).
2. Use `msg_key` to rederive IGE key *K* and *IV*.
3. Run IGE decryption on c_{SE} with (*K*,*IV*) to get payload *p* = *p*[1] *p*[2] … *p*[*t*].
4. Recompute MAC on *p* and compare to `msg_key`; accept if they match, otherwise reject.
5. Sanity check payload *p*, e.g. do message length checks, rejecting if invalid.

Details of Timing Attack on MTProto 2.0 E&M





<u>Real-world decryption process:</u>

1. Parse c as (`msg_key`, c_{SE})
2. Use `msg_key` to rederive IGE key and IV.
3. Run partial IGE decryption on c_{SE} to get partial payload $p = p[1]\, p[2]$.
4. Sanity check `msg_length` field in $p[2]$; reject if out of range.
5. Decrypt remaining blocks $p[3]\, p[4]$ …
6. Recompute MAC on p and compare to `msg_key`; accept if they match, otherwise reject.

Timing side channel based on content of *p*[2]!

Details of Timing Attack on MTProto 2.0 E&M

Ideal world: From Telegram security guidelines for client developers:

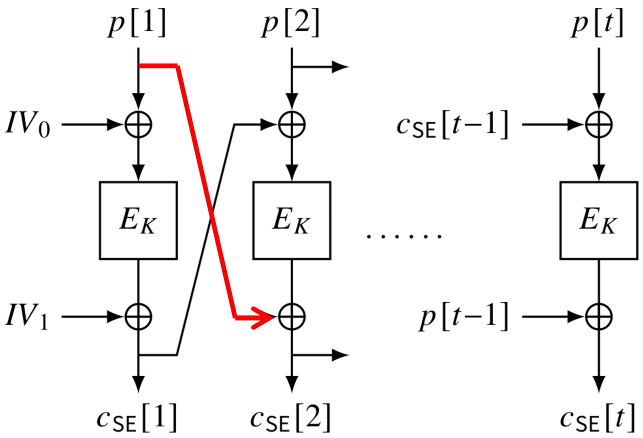
> "If an error is encountered before this check could be performed, the client must perform the `msg_key` check anyway before returning any result.
>
> "Note that the response to any error encountered before the `msg_key` check must be the same as the response to a failed `msg_key` check."

Real world: Telegram official desktop client, prior to `msg_key` check:

```
if (messageLength > kMaxMessageLength) {
        LOG(("TCP Error: bad messageLength %1").arg(
                messageLength));
        TCP_LOG(("TCP Error: bad message %1").arg(
                Logs::mb(ints,
                        intsCount * kIntSize).str()));

        return restart();
}
// ...
// MAC computation and check follow
```

IGE Malleability



IGE: $c[j] = E_K(p[j] \oplus c[j\text{-}1]) \oplus p[j\text{-}1]$

$p[j] = D_K(c[j] \oplus p[j\text{-}1]) \oplus c[j\text{-}1]$

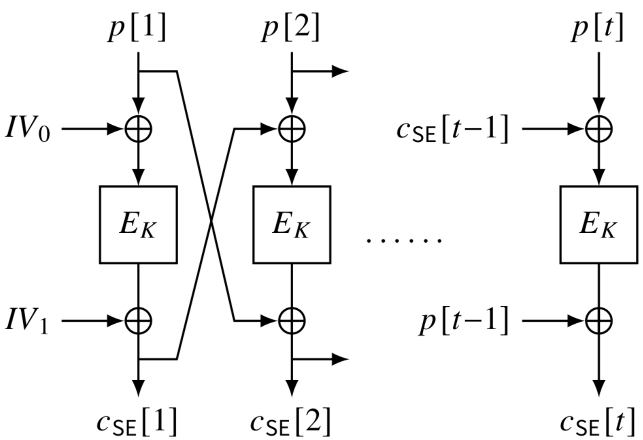
- Suppose $p[i\text{-}1]$ and $p[j\text{-}1]$ are both known, for two indices i, j.
- Consider setting:

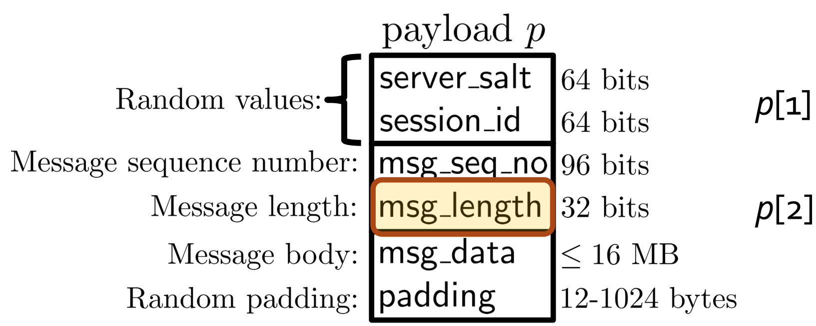
  $c[j]' = c[i] \oplus p[j\text{-}1] \oplus p[i\text{-}1]$.
- Then, after decryption:

$$\begin{aligned} p[j]' &= D_K(c[j]' \oplus p[j\text{-}1]) \oplus c[j\text{-}1] \\ &= D_K(c[i] \oplus p[j\text{-}1] \oplus p[i\text{-}1] \oplus p[j\text{-}1]) \oplus c[j\text{-}1] \\ &= D_K(c[i] \oplus p[i\text{-}1]) \oplus c[j\text{-}1] \\ &= D_K(c[i] \oplus p[\mathrm{i}\text{-}1]) \oplus c[i\text{-}1] \oplus c[i\text{-}1] \oplus c[j\text{-}1] \\ &= p[i] \oplus c[i\text{-}1] \oplus c[j\text{-}1] \end{aligned}$$

Plaintext in position j is now related to $p[i]$!

Details of Timing Attack on MTProto 2.0 E&M





<u>Recovering plaintext from a target block $c_{\mathsf{SE}}[i]$:</u>

- Suppose blocks $p[i\text{-}1]$ and $p[1]$ are known (case of j=2 from previous slide) and we want to recover information about target block $p[i]$.
- MITM attacker intercepts c and replaces block $c_{\mathsf{SE}}[2]$ with:

$$c_{\mathsf{SE}}[2]' = c_{\mathsf{SE}}[i] \oplus p[1] \oplus p[i\text{-}1].$$

- Then $c_{\mathsf{SE}}[2]'$ decrypts to $p[i] \oplus c_{\mathsf{SE}}[i\text{-}1] \oplus c_{\mathsf{SE}}[1]$.
- The length sanity check is now done on 32 bits of $p[i] \oplus c_{\mathsf{SE}}[i\text{-}1] \oplus c_{\mathsf{SE}}[1]$.
- Hence the timing side channel leaks some information on $p[i]$, the target block.

Details of Timing Attack on MTProto 2.0 E&M

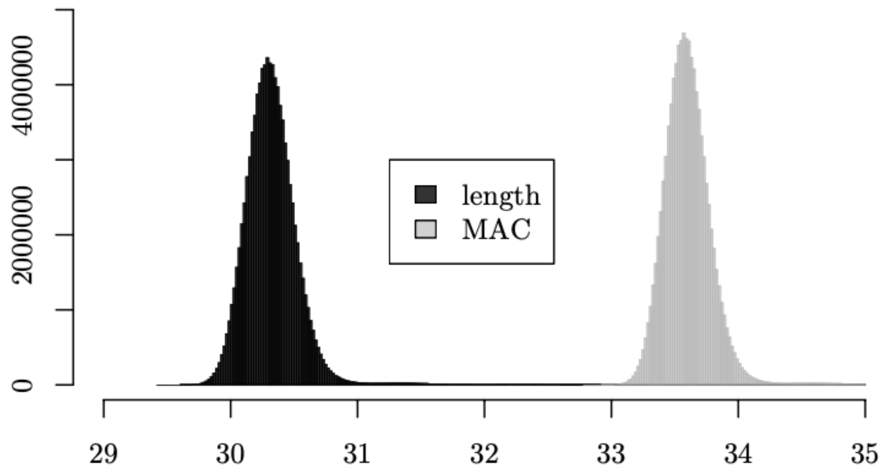
Telegram official desktop client, prior to msg_key check:

```
if (messageLength > kMaxMessageLength) {
        LOG(("TCP Error: bad messageLength %1").arg(
                messageLength));
        TCP_LOG(("TCP Error: bad message %1").arg(
                Logs::mb(ints,
                         intsCount * kIntSize).str()));

        return restart();
}
// ...
// MAC computation and check follow
```

- Value of `kMaxMessageLength`: 2^{24}.
- `messageLength`: set to least significant 32 bits of $p[i] \oplus c_{SE}[i\text{-}1] \oplus c_{SE}[1]$.
- This gives an attack that can recover 8 bits of $p[i]$.
- Because the "if" statement effectively checks if 8 MSBs of $p[i] \oplus c_{SE}[i\text{-}1] \oplus c_{SE}[1]$ are zero or not.
- Success probability 1/256 per attempt but can iterate if $p[i]$ is fixed across many messages.

Details of Timing Attack on MTProto 2.0 E&M



Timing difference in microseconds, length check failure (left) versus MAC failure (right), under ideal conditions, i.e. with hyper-threading, Turbo Boost etc. disabled, Linux desktop.

Four Attacks **and a Proof?**

- With small tweaks, we obtained a version of MTProto 2.0's secure channel construction that we could prove secure.
- Required new security model for *stateful bidirectional channels.*
- Standard assumptions:
  - Collision resistance of SHA-256 with truncated output
  - One-time IND-CPA security of CBC mode
  - One-time PRF security of SHACAL-1
  - One-time PRF security of the compression function of SHA-256
- New assumptions:
  - Related-key PRF security of SHACAL-2 with key leakage
  - Needed because of overlapping key bits.

Four Attacks and a Proof?

Theorem 1. *Let* ME, HASH, MAC, KDF, ϕ_{MAC}, ϕ_{KDF}, SE *be any primitives that meet the requirements stated in Definition 5 of channel* MTP-CH. *Let* CH = MTP-CH[ME, HASH, MAC, KDF, ϕ_{MAC}, ϕ_{KDF}, SE]. *Let* $\mathcal{D}_{\mathrm{IND}}$ *be any adversary against the* IND-*security of* CH, *making* q_{CH} *queries to its* CH *oracle. Then there exist adversaries* $\mathcal{D}_{\mathrm{OTWIND}}$, $\mathcal{D}_{\mathrm{RKPRF}}$, $\mathcal{D}_{\mathrm{ENCROB}}$, $\mathcal{F}_{\mathrm{UPREF}}$, $\mathcal{D}_{\mathrm{UPRKPRF}}$, $\mathcal{D}_{\mathrm{OTIND\$}}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{CH}}(\mathcal{D}_{\mathrm{IND}}) \le 2\cdot\Big(&\mathsf{Adv}^{\mathsf{otwind}}_{\mathsf{HASH}}(\mathcal{D}_{\mathrm{OTWIND}}) + \mathsf{Adv}^{\mathsf{rkprf}}_{\mathsf{KDF},\phi_{\mathsf{KDF}}}(\mathcal{D}_{\mathrm{RKPRF}})\\ &+ \mathsf{Adv}^{\mathsf{encrob}}_{\mathsf{ME}}(\mathcal{D}_{\mathrm{ENCROB}}) + \mathsf{Adv}^{\mathsf{upref}}_{\mathsf{ME}}(\mathcal{F}_{\mathrm{UPREF}})\\ &+ \mathsf{Adv}^{\mathsf{uprkprf}}_{\mathsf{MAC},\phi_{\mathsf{MAC}}}(\mathcal{D}_{\mathrm{UPRKPRF}}) + \frac{q_{\mathrm{CH}}\cdot(q_{\mathrm{CH}}-1)}{2\cdot 2^{\mathsf{MAC.ol}}}\\ &+ \mathsf{Adv}^{\mathsf{otind\$}}_{\mathsf{SE}}(\mathcal{D}_{\mathrm{OTIND\$}})\Big).\end{aligned}$$

Theorem 2. *Let* session_id $\in \{0,1\}^{64}$, pb $\in \mathbb{N}$, *and* bl $= 128$. *Let* ME = MTP-ME[session_id, pb, bl] *be the message encoding scheme as defined in Definition 6. Let* SE = MTP-SE *be the deterministic symmetric encryption scheme as defined in Definition 10. Let* HASH, MAC, KDF, ϕ_{MAC}, ϕ_{KDF} *be any primitives that, together with* ME *and* SE, *meet the requirements stated in Definition 5 of channel* MTP-CH. *Let* CH = MTP-CH[ME, HASH, MAC, KDF, ϕ_{MAC}, ϕ_{KDF}, SE]. *Let* supp = SUPP *be the support function as defined in Fig. 23. Let* $\mathcal{F}_{\mathrm{INT}}$ *be any adversary against the* INT-*security of* CH *with respect to* supp. *Then there exist adversaries* $\mathcal{D}_{\mathrm{OTWIND}}$, $\mathcal{D}_{\mathrm{RKPRF}}$, $\mathcal{F}_{\mathrm{UNPRED}}$, $\mathcal{F}_{\mathrm{RKCR}}$, $\mathcal{F}_{\mathrm{EINT}}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{int}}_{\mathsf{CH,supp}}(\mathcal{F}_{\mathrm{INT}}) \le\ &\mathsf{Adv}^{\mathsf{otwind}}_{\mathsf{HASH}}(\mathcal{D}_{\mathrm{OTWIND}}) + \mathsf{Adv}^{\mathsf{rkprf}}_{\mathsf{KDF},\phi_{\mathsf{KDF}}}(\mathcal{D}_{\mathrm{RKPRF}})\\ &+ \mathsf{Adv}^{\mathsf{unpred}}_{\mathsf{SE,ME}}(\mathcal{F}_{\mathrm{UNPRED}}) + \mathsf{Adv}^{\mathsf{rkcr}}_{\mathsf{MAC},\phi_{\mathsf{MAC}}}(\mathcal{F}_{\mathrm{RKCR}})\\ &+ \mathsf{Adv}^{\mathsf{eint}}_{\mathsf{ME,supp}}(\mathcal{F}_{\mathrm{EINT}}).\end{aligned}$$

- Computational security proofs for a pseudo-code description of Telegram's MTProto 2.0.
- Concrete reductions to standard and non-standard security properties.
- Proofs use standard game hopping techniques as seen in this course.
- Full details in the paper at: https://mtpsym.github.io/.

Studying the Wider Telegram Ecosystem

On the Cryptographic Fragility of the Telegram Ecosystem

Theo von Arx
ETH Zurich
theo.vonarx@inf.ethz.ch

Kenneth G. Paterson
ETH Zurich
kenny.paterson@inf.ethz.ch

eprint 2022/595
AsiaCCS 2023, to appear

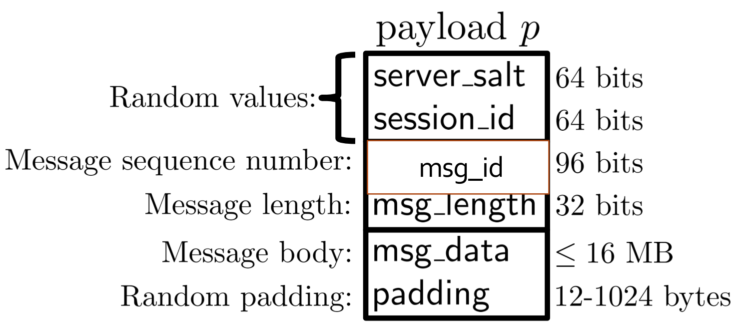
- Telegram encourages third party development of clients.
- So what can we say about their security?

The Telegram Ecosystem

Telegram security guidelines for client developers:

The client must check that `msg_id` has even parity for messages from client to server, and odd parity for messages from server to client.

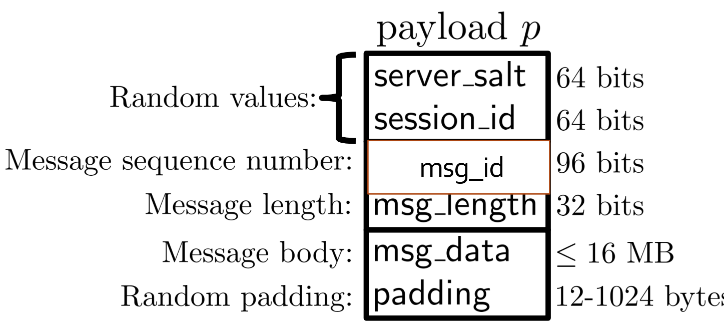
In addition, the identifiers (`msg_id` fields) of the last N messages received from the other side must be stored, and if a message comes in with an `msg_id` lower than all or equal to any of the stored values, that message is to be ignored. Otherwise, the new message `msg_id` is added to the set, and, if the number of stored `msg_id` values is greater than N, the oldest (i.e. the lowest) is discarded.



Telegram security guidelines for client developers, part 2:

In addition, `msg_id` values that belong over 30 seconds in the future or over 300 seconds in the past are to be ignored (recall that `msg_id` approximately equals `unixtime` * 232). This is especially important for the server. The client would also find this useful (to protect from a replay attack), but only if it is certain of its time (for example, if its time has been synchronized with that of the server).

Certain client-to-server service messages containing data sent by the client to the server (for example, `msg_id` of a recent client query) may, nonetheless, be processed on the client even if the time appears to be "incorrect".



The Telegram Ecosystem

In a more condensed form:

1. msg_id % 2==1
2. msg_id $\notin$ stored_msg_ids
3. msg_id > min(stored_msg_ids)
4. now() - 300s < msg_id < now() + 30s
5. stored_msg_ids $\leftarrow$ stored_msg_ids $\cup$ {msg_id}, |stored_msg_ids| $\leq$ *N*

NB Even if properly implemented, these checks do not prevent reordering attacks, as we already saw!

The Telegram Ecosystem

Name	★	Used by	Language
GramJS	379	580	JavaScript
MadelineProto	1.9K	167	PHP
Pyrogram	2.4K	39.2K	Python
TDLib*	4.5k	-	C++
Telethon	6.2K	25.9K	Python

Pyrogram

Listing 1: `mtproto.py`. Message processing in Pyrogram [30]. Modified for readability.

```
def unpack(b: BytesIO, session_id: bytes, auth_key: bytes
        , auth_key_id: bytes) -> Message:
    # [...]
    data = BytesIO(aes.ige256_decrypt(b.read(), aes_key,
            aes_iv))
    # [...]
    message = Message.read(data)
    # [...]
    # https://core.telegram.org/mtproto/
            security_guidelines#checking-msg-id
    assert message.msg_id % 2 != 0
    return message
```

Telethon

Listing 2: `mtprotostate.py`. Message processing in Telethon [31]. Modified for readability.

```python
def decrypt_message_data(self, body):
    # TODO Check salt, session_id and sequence_number
    # [...]
    body = AES.decrypt_ige(body[24:], aes_key, aes_iv)
    # [...]
    reader = BinaryReader(body)
    reader.read_long()  # remote_salt
    if reader.read_long() != self.id:
        raise SecurityError('Wrong session ID')
    remote_msg_id = reader.read_long()
    remote_sequence = reader.read_int()
    reader.read_int()  # msg_len
    obj = reader.tgread_object()
    return TLMessage(remote_msg_id, remote_sequence, obj)
```

GramJS

Listing 3: `MTProtoState.ts`. Message processing in GramJS [32]. Modified for readability.

```
async decryptMessageData(body: Buffer) {
    // [...]
    // TODO Check salt,sessionId, and sequenceNumber
    const keyId = helpers.readBigIntFromBuffer(body.slice
        (0, 8));
    // [...]
    body = new IGE(key, iv).decryptIge(body.slice(24));
    // [...]
    const reader = new BinaryReader(body);
    reader.readLong(); // removeSalt
    const serverId = reader.readLong();
    if (serverId !== this.id) {
        // throw new SecurityError('Wrong session ID');
    }
    const remoteMsgId = reader.readLong();
    const remoteSequence = reader.readInt();
    reader.readInt(); // msgLen
    // [...]
    const obj = reader.tgReadObject();
    return new TLMessage(remoteMsgId,remoteSequence,obj);
}
```

Telethon Replay and Reordering Attack

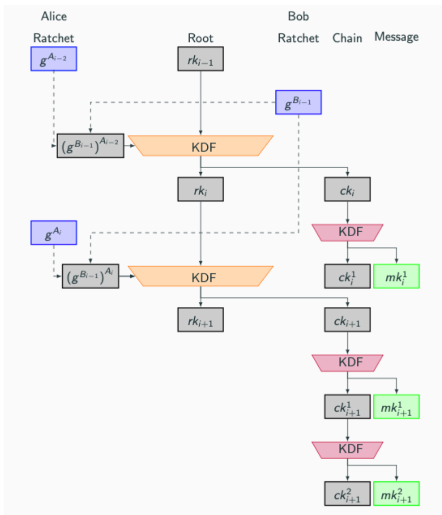
Telethon: Reorder Attack – Sample Run

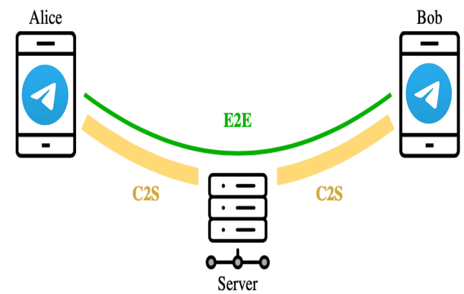
Sender	Receiver
Hi!	Hi!
I say yes	Hi!
to Pizza	Hi!
I say no	Hi!
to Crime	Hi!
That’s it!	I say yes
Please vote me for president!	to Crime
And nobody else!	I say no
See you!	to Pizza
Bye	Bye

The Telegram Ecosystem

- Telegram encourages third party development of clients.
- This attracts cryptographically inexperienced developers.
  - Pyrogram, Telethon, GramJS all had critical issues around message replay/reordering prevention.
  - MadelineProto was vulnerable to another timing attack on E&M.
- At the same time, the Telegram MTProto 2.0 protocol is cryptographically fragile and the developer guidance is complex and unclear.
- Telegram does not have a public communication channel to inform its developer community about security vulnerabilities.
  - So the earlier attacks on Telegram did not lead to fixes across the ecosystem.
- Telegram's TDLib helps, but only to a limited extent.
- Telegram should just replace MTProto 2.0 with TLS 1.3.

Telegram in Comparison to Signal





<u>Signal</u>

- E2EE by default
- Double-ratchet mechanism
- Fine-grained key derivation for FS and PCS
- Boring cryptography: EtM, HMAC, HKDF, etc.

<u>MTProto 2.0 in Telegram</u>

- **Not** E2EE by default
- DH mod p + RSA auth in handshake followed by E&M in MTProto 2.0
- Relatively static keys
- Exotic+old crypto: E&M, IGE, weird key derivation, mod p DH

Course Wrap Up

Course wrap up

- This course has only “scratched the surface” of applied cryptography.
- We focused on introducing the basic primitives in a semi-rigorous manner and on exploring a few case studies (password hashing, Signal, TLS, Telegram).
- We have glimpsed the research frontier at several points in the course.
- The principles you have seen should serve you well in building and analyzing systems using cryptography.

Course wrap up

Topics deserving of further study include:

- Randomness / PRNGs
- More extensive treatment of authentication and key exchange
- More advanced primitives.
- More applications (e.g. storage encryption, searchable encryption, crypto-currencies,...)

Course wrap up

A few key take-aways:

- You are not (yet) qualified to design your own cryptographic algorithms, but nor should you ever really need to!
- Use cryptographic libraries where you can, rather than "rolling your own". This saves time and will help you avoid many of the pitfalls.
- Designing higher-level cryptographic protocols is also fraught with dangers, so don't roll your own unless you really must.
- And if you must design your own protocol, then seek external review by experts.
- More advice in CyBoK paper: https://www.cybok.org/media/downloads/Applied_Cryptography_v1.0.0.pdf

Course wrap up

A few more key take-aways:

- Strong security goals can be achieved efficiently under reasonable assumptions to solve basic cryptographic problems like secure communication and secure storage.
- Mature standards exist to support these applications.
- This does not mean they are universally followed!
- We do not yet know how to achieve more advanced cryptographic goals as efficiently as we would like, but rapid progress is being made, in, e.g. ZK proofs, MPC, FHE, PIR,…
- We are now seeing lots of deployment of these more advanced primitives.
- It's a great time to become a cryptographer!

Course wrap up

Further courses on cryptography that you can take at ETH:

- Cryptographic protocols (Spring semester, Profs. Hirt and Maurer).
- Digital signatures (Spring semester, Prof. Hofheinz).
- Zero-Knowledge Proofs (Autumn semester, Dr. Jonathan Bootle, IBM Zurich).
- Advanced Encryption Schemes (Autumn semester, Dr. Romain Gay, IBM Zurich).

Seminars:

- Current topics in Information Security (seminar, Autumn semester, Profs. Basin, Capkun, Paterson, Perrig, Shinde).
- Current topics in cryptography (seminar, Spring semester, Profs. Hofheinz, Maurer, Paterson).

Projects/theses:

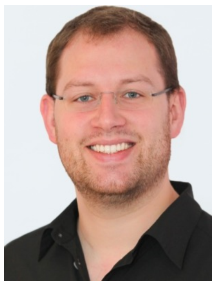
- The Applied Cryptography Group also welcomes approaches for semester projects and Master's theses.
- Details at: https://appliedcrypto.ethz.ch/education/student-projects.html

Course wrap up

Final exam info:

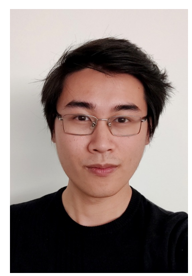
- 13.06.2023, 10-12.
- HG D 1.2 & F 1 & G 5.
- Expect 3 problems, 25 marks each, similar standard to mid-term exam and previous final exams.
- (Tentative) final exam inspection time: 28.06.2023, 10-12.

“Danke vielmal” from the whole team

Prof. Kenny Paterson
Dr. Felix Günther



<u>TAs:</u>
Mia Filić, MSc (Head TA)
Matilda Backendal, MSc
Matteo Scarlata, MSc
Kien Tuong Truong, MSc

<u>HAs:</u>
Daniele Coppola
Lucas Dodgson
Hongyi Ling
Andreas Tsouloupas