Analysis of the Telegram Key Exchange

Martin R. Albrecht[1] **Lenka Mareková**[2] Kenneth G. Paterson[2] Eyal Ronen[3] Igors Stepanovs[4]

[1]King's College London
[2]ETH Zurich
[3]Tel-Aviv University
[4]Amazon

8 May 2025, Madrid, Spain

Introduction

- Telegram is a messaging application with a billion users

The Atlantic:

The World's Most Important App (For Now)

Telegram played a central role in Yevgeny Prigozhin's revolt—and in so many other chaotic events that have defined recent history.

By Charlie Warzel

Introduction

- Telegram is a messaging application with a billion users
- It uses custom, highly non-standard cryptographic protocols (MTProto 2.0)

Introduction

- Telegram is a messaging application with a billion users
- It uses custom, highly non-standard cryptographic protocols (MTProto 2.0)

Caution: E2EE in Telegram is limited and largely unused
$\Longrightarrow$ **Focus on client-server connections**

Introduction

- Telegram is a messaging application with a billion users
- It uses custom, highly non-standard cryptographic protocols (MTProto 2.0)

Caution: E2EE in Telegram is limited and largely unused
$\Longrightarrow$ **Focus on client-server connections**

- In previous work [AMPS22] we

[AMPS22]: M. R. Albrecht, L. Mareková, K. G. Paterson and I. Stepanovs. *Four attacks and a proof for Telegram.* IEEE S&P, 2022.

Introduction

- Telegram is a messaging application with a billion users
- It uses custom, highly non-standard cryptographic protocols (MTProto 2.0)

Caution: E2EE in Telegram is limited and largely unused
$\Longrightarrow$ **Focus on client-server connections**

- In previous work [AMPS22] we
  - analysed and wrote proofs for the channel protocol

[AMPS22]: M. R. Albrecht, L. Mareková, K. G. Paterson and I. Stepanovs. *Four attacks and a proof for Telegram.* IEEE S&P, 2022.

Introduction

- Telegram is a messaging application with a billion users
- It uses custom, highly non-standard cryptographic protocols (MTProto 2.0)

Caution: E2EE in Telegram is limited and largely unused
$\Longrightarrow$ **Focus on client-server connections**

- In previous work [AMPS22] we
  - analysed and wrote proofs for the channel protocol
  - found an attack on the key exchange protocol

[AMPS22]: M. R. Albrecht, L. Mareková, K. G. Paterson and I. Stepanovs. *Four attacks and a proof for Telegram.* IEEE S&P, 2022.

Introduction

- Telegram is a messaging application with a billion users
- It uses custom, highly non-standard cryptographic protocols (MTProto 2.0)

Caution: E2EE in Telegram is limited and largely unused
$\Longrightarrow$ **Focus on client-server connections**

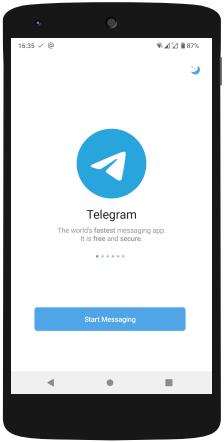
- In previous work [AMPS22] we
  - analysed and wrote proofs for the channel protocol
  - found an attack on the key exchange protocol
- Aim of this work: analyse the “fixed” key exchange protocol

[AMPS22]: M. R. Albrecht, L. Mareková, K. G. Paterson and I. Stepanovs. *Four attacks and a proof for Telegram.* IEEE S&P, 2022.

Overview

1. Introduction
2. Security model
3. Telegram protocols
4. Results
5. Takeaways

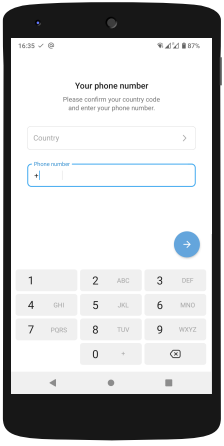
First use protocol flow



Client

Server

First use protocol flow

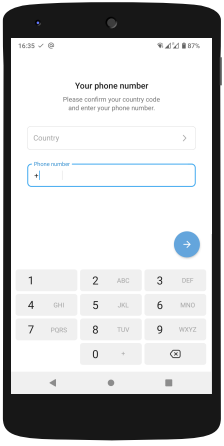


Client

establish long-term ak

Server

First use protocol flow



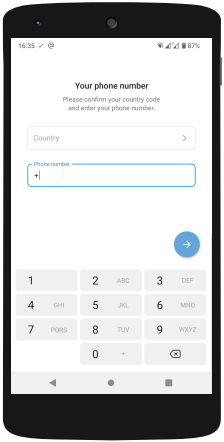
Client

establish long-term ak

establish temporary ak_{t}

Server

First use protocol flow



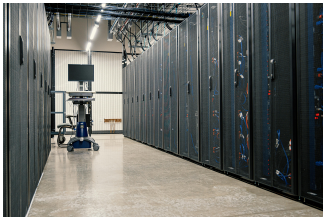
Client

establish long-term ak

establish temporary ak_t

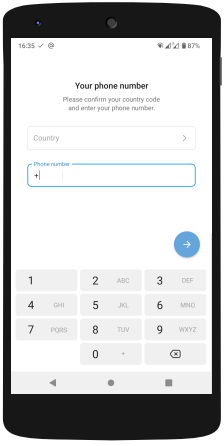
bind ak_t to ak

channel via ak_t



Server

First use protocol flow



Client

establish long-term ak

establish temporary ak_t

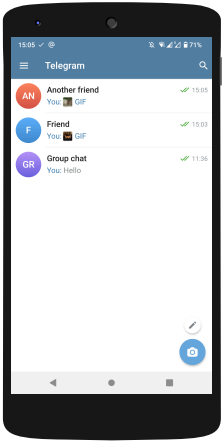
bind ak_t to ak

channel via ak_t

phone num. auth.

Server

Subsequent use protocol flow



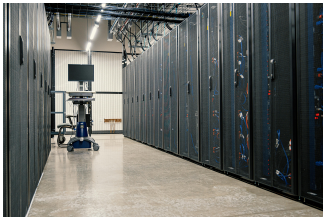
Client

establish temporary ak_t

bind ak_t to ak

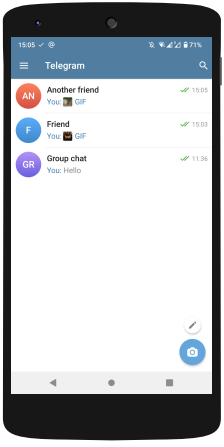
channel via ak_t

chat messages



Server

Scope of analysis



Client

key exchange

establish long-term ak

establish temporary ak_t

bind ak_t to ak

channel via ak_t

chat messages

channel

based on docs, client code & server interaction

Server

Overview of this talk

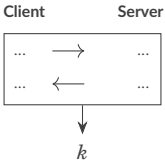
1. Introduction
2. Security model
3. Telegram protocols
4. Results
5. Takeaways

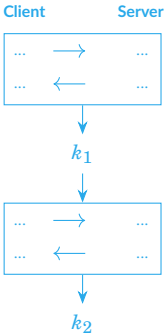
Security model

- New game-based authenticated key exchange model tailored to Telegram

Security model

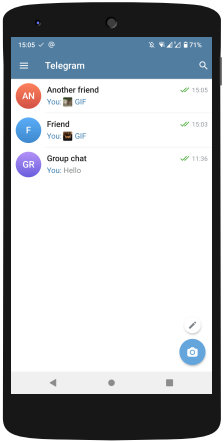
- New game-based authenticated key exchange model tailored to Telegram
- Building upon the **multi-stage** key exchange (MSKE) models [FG14, DFGS21]



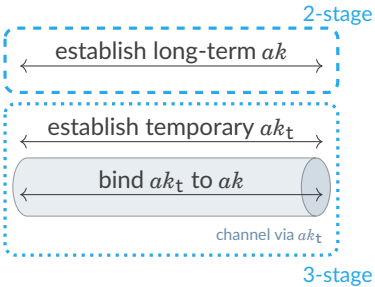


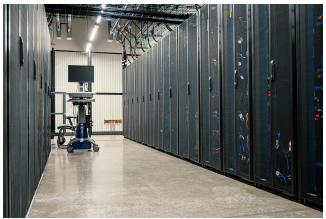
[FG14]: M. Fischlin and F. Günther. *Multi-stage key exchange and the case of Google's QUIC protocol.* CCS, 2014.
[DFGS21]: B. Dowling, M. Fischlin, F. Günther, and D. Stebila. *A cryptographic analysis of the TLS 1.3 handshake protocol.* JoC, 2021.

Telegram protocols as MSKE



Client





Server

Security model

- Standard setup (multi-user setting)

Security model

- Standard setup (multi-user setting)
- Standard key exchange adversary (Corrupt, Reveal, Test oracles)

Security model

- Standard setup (multi-user setting)
- Standard key exchange adversary (Corrupt, Reveal, Test oracles)
- Basic security goals

Security model

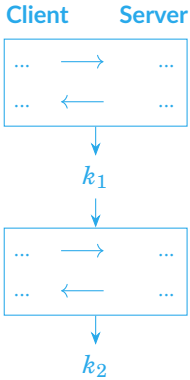
- Standard setup (multi-user setting)
- Standard key exchange adversary (Corrupt, Reveal, Test oracles)
- Basic security goals
  - indistinguishability of session keys of all stages, with forward secrecy

Security model

- Standard setup (multi-user setting)
- Standard key exchange adversary (Corrupt, Reveal, Test oracles)
- Basic security goals
  - indistinguishability of session keys of all stages, with forward secrecy
  - client and server authentication (without full key confirmation)

Security model

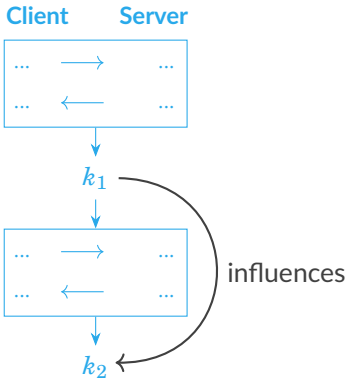
Weaker or non-standard properties:



Security model

Weaker or non-standard properties:

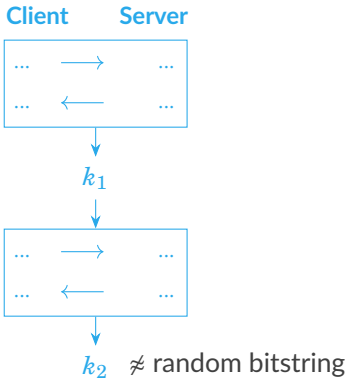
- Key dependence between stages



Security model

Weaker or non-standard properties:

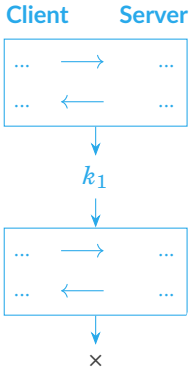
- Key dependence between stages
- Non-uniform session key distribution



Security model

Weaker or non-standard properties:

- Key dependence between stages
- Non-uniform session key distribution
- Testable and non-testable stages



auth. only

Overview of this talk

1. Introduction
2. Security model
3. Telegram protocols
4. Results
5. Takeaways

Establishing ak – Stage 1

Client (knows pk)

Server (has (pk, sk))

$n \leftarrow\$ \{0,1\}^{128}$

n →

$n_s \leftarrow\$ \{0,1\}^{128}$

$p', q' \leftarrow\$ \{\text{32-bit primes}\}$

$n_n \leftarrow\$ \{0,1\}^{256}$

← $n, n_s, prod', \mathcal{F}$

$prod' \leftarrow p' \cdot q'$

$c_0 \leftarrow\$$ TOAEP$^+$ .Enc$(pk, \ldots n_n \ldots)$

n, n_s, p', q', f, c_0 →

Establishing ak – Stage 1

Client (knows pk)

Server (has (pk, sk))

$n \leftarrow\$ \{0,1\}^{128}$

n →

$n_s \leftarrow\$ \{0,1\}^{128}$

$p', q' \leftarrow\$ \{\text{32-bit primes}\}$

$prod' \leftarrow p' \cdot q'$

$n_n \leftarrow\$ \{0,1\}^{256}$

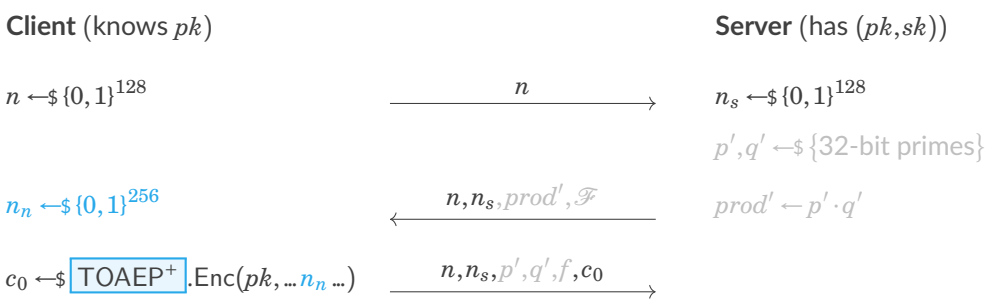
← $n, n_s, prod', \mathcal{F}$

$c_0 \leftarrow\$$ TOAEP$^+$ .Enc$(pk, \ldots n_n \ldots)$

n, n_s, p', q', f, c_0 →

stage 1 session key: n_n

Establishing ak – Stage 1



Telegram OAEP+ (custom version of OAEP+ [Sho02])

TOAEP$^+$.Enc(pk, m)

1: extract N from pk
2: $w \leftarrow N$
3: **while** $w \geq N$
4: $r \leftarrow\$ \{0,1\}^{256}$
5: $x \leftarrow\$ \mathsf{pack}(m)$
6: $c \leftarrow \text{SHA-256}(r \parallel x)$
7: $z \leftarrow \mathsf{reverse}(x) \parallel c$
8: $s \leftarrow \text{AES-256-IGE.Enc}(r, 0, z)$
9: $t \leftarrow \text{SHA-256}(s) \oplus r$
10: $w \leftarrow t \parallel s$
11: $c \leftarrow \text{RSA.Enc}(pk, w)$
12: **return** c

TOAEP$^+$.Dec(sk, c)

1: $w \leftarrow \text{RSA.Dec}(sk, c)$
2: $t \leftarrow w[0:256]$
3: $s \leftarrow w[256:2048]$
4: $r \leftarrow \text{SHA-256}(s) \oplus t$
5: $z \leftarrow \text{AES-256-IGE.Dec}(r, 0, s)$
6: $x \leftarrow \mathsf{reverse}(z[0:1536])$
7: $c \leftarrow z[1536:1792]$
8: **if** $c \neq \text{SHA-256}(r \parallel x)$ **then return** $\perp$
9: $m \leftarrow \mathsf{unpack}(x)$
10: **return** m

[Sho02]: Victor Shoup. *OAEP reconsidered.* Journal of Cryptology, 2002.

Telegram OAEP+ (custom version of OAEP+ [Sho02])

TOAEP$^+$.Enc(pk, m)

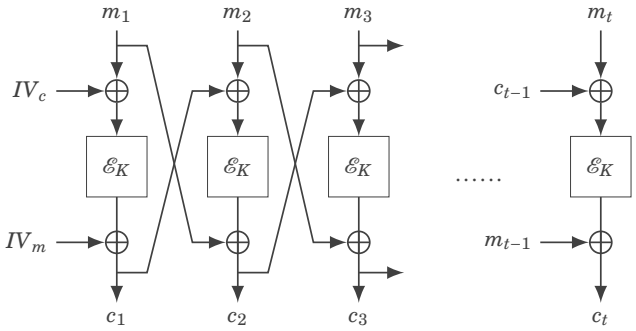
1: extract N from pk
2: $w \leftarrow N$
3: **while** $w \geq N$
4: $r \leftarrow\$ \{0,1\}^{256}$
5: $x \leftarrow\$$ pack(m)
6: $c \leftarrow$ SHA-256$(r \parallel x)$
7: $z \leftarrow$ reverse$(x) \parallel c$
8: $s \leftarrow$ AES-256-IGE.Enc$(r, 0, z)$
9: $t \leftarrow$ SHA-256$(s) \oplus r$
10: $w \leftarrow t \parallel s$
11: $c \leftarrow$ RSA.Enc(pk, w)
12: **return** c

TOAEP$^+$.Dec(sk, c)

1: $w \leftarrow$ RSA.Dec(sk, c)
2: $t \leftarrow w[0:256]$
3: $s \leftarrow w[256:2048]$
4: $r \leftarrow$ SHA-256$(s) \oplus t$
5: $z \leftarrow$ AES-256-IGE.Dec$(r, 0, s)$
6: $x \leftarrow$ reverse$(z[0:1536])$
7: $c \leftarrow z[1536:1792]$
8: **if** $c \neq$ SHA-256$(r \parallel x)$ **then return** $\perp$
9: $m \leftarrow$ unpack(x)
10: **return** m

[Sho02]: Victor Shoup. *OAEP reconsidered.* Journal of Cryptology, 2002.

IGE..?


m_1
m_2
m_3
m_t
IV_c
c_{t-1}
$\mathcal{E}_K$
$\mathcal{E}_K$
$\mathcal{E}_K$
.....
$\mathcal{E}_K$
IV_m
m_{t-1}
c_1
c_2
c_3
c_t

Establishing ak – Stage 2

Client (has n_n)		**Server** (has n_n)
		$a \leftarrow\$ \{0,1\}^{2048}$
		$k \parallel iv \leftarrow$ SKDF .Ev(n_n, n_s)
	$\xleftarrow{n, n_s, c_1}$	$c_1 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^a \bmod p$ …)
repeatable		
$b \leftarrow\$ \{0,1\}^{2048}$		
$c_2 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^b \bmod p$ …)	$\xrightarrow{n, n_s, c_2}$	$ak \leftarrow g^{ab} \bmod p$
		$ax \parallel \ldots \parallel aid \leftarrow$ SHA-1(ak)
		if $aid \in \mathscr{S}_{\text{aid}}$ **then** wait for retry
if $h_i = h_2$ **then** retry	$\xleftarrow{n, n_s, h_i}$	$h_i \leftarrow$ NH$_i$ .Ev(n_n, ax)

Establishing ak – Stage 2

Client (has n_n)

Server (has n_n)

$a \leftarrow\$ \{0,1\}^{2048}$

$k \parallel iv \leftarrow$ SKDF .Ev(n_n, n_s)

$n, n_s, c_1 \longleftarrow$

$c_1 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^a \bmod p$ …)

repeatable

$b \leftarrow\$ \{0,1\}^{2048}$

$c_2 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^b \bmod p$ …)

$\longrightarrow n, n_s, c_2$

$ak \leftarrow g^{ab} \bmod p$

$ax \parallel \ldots \parallel aid \leftarrow$ SHA-1(ak)

if $aid \in \mathscr{S}_{\text{aid}}$ **then** wait for retry

if $h_i = h_2$ **then** retry

$n, n_s, h_i \longleftarrow$

$h_i \leftarrow$ NH$_i$ .Ev(n_n, ax)

stage 2 session key: $ak[0:1024]$

Establishing ak – Stage 2

Client (has n_n)

Server (has n_n)

$a \leftarrow\$ \{0,1\}^{2048}$

$k \parallel iv \leftarrow$ SKDF .Ev(n_n, n_s)

n, n_s, c_1 ←

$c_1 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^a \bmod p$ …)

repeatable

$b \leftarrow\$ \{0,1\}^{2048}$

$c_2 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^b \bmod p$ …)

→ n, n_s, c_2

$ak \leftarrow g^{ab} \bmod p$

$ax \parallel \ldots \parallel aid \leftarrow$ SHA-1(ak)

if $aid \in \mathcal{S}_{\text{aid}}$ **then** wait for retry

if $h_i = h_2$ **then** retry

n, n_s, h_i ←

$h_i \leftarrow$ NH$_i$ .Ev(n_n, ax)

stage 2 session key: $ak[0:1024]$

Establishing ak – Stage 2

Client (has n_n)

Server (has n_n)

$a \leftarrow\$ \{0,1\}^{2048}$

$k \parallel iv \leftarrow$ SKDF .Ev(n_n, n_s)

n, n_s, c_1 ←

$c_1 \leftarrow\$$ HtE .Enc($k \parallel iv$, … $g^a \bmod p$ …)

repeatable

$b \leftarrow\$ \{0,1\}^{2048}$

$c_2 \leftarrow\$$ HtE .Enc($k \parallel iv$, … $g^b \bmod p$ …)

→ n, n_s, c_2

$ak \leftarrow g^{ab} \bmod p$

$ax \parallel \ldots \parallel aid \leftarrow$ SHA-1(ak)

if $aid \in \mathcal{S}_{\text{aid}}$ **then** wait for retry

if $h_i = h_2$ **then** retry

n, n_s, h_i ←

$h_i \leftarrow$ NH$_i$ .Ev(n_n, ax)

stage 2 session key: $ak[0:1024]$

Establishing ak – Stage 2

Client (has n_n)

Server (has n_n)

$a \leftarrow\$ \{0,1\}^{2048}$

$k \parallel iv \leftarrow$ SKDF .Ev(n_n, n_s)

n, n_s, c_1 ←

$c_1 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^a \bmod p$ …)

repeatable

$b \leftarrow\$ \{0,1\}^{2048}$

$c_2 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^b \bmod p$ …)

n, n_s, c_2 →

$ak \leftarrow g^{ab} \bmod p$

$ax \parallel \ldots \parallel aid \leftarrow$ SHA-1(ak)

if $aid \in \mathcal{S}_{\text{aid}}$ **then** wait for retry

if $h_i = h_2$ **then** retry

n, n_s, h_i ←

$h_i \leftarrow$ NH$_i$ .Ev(n_n, ax)

stage 2 session key: $ak[0:1024]$

Establishing ak – Stage 2

Client (has n_n)

Server (has n_n)

$a \leftarrow\$ \{0,1\}^{2048}$

$k \parallel iv \leftarrow$ SKDF .Ev(n_n, n_s)

n, n_s, c_1 ←

$c_1 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^a \bmod p$ …)

repeatable

$b \leftarrow\$ \{0,1\}^{2048}$

$c_2 \leftarrow\$$ HtE . Enc ($k \parallel iv$, … $g^b \bmod p$ …)

→ n, n_s, c_2

$ak \leftarrow g^{ab} \bmod p$

$ax \parallel \ldots \parallel aid \leftarrow$ SHA-1(ak)

if $aid \in \mathscr{S}_{\text{aid}}$ **then** wait for retry

if $h_i = h_2$ **then** retry

n, n_s, h_i ←

$h_i \leftarrow$ NH$_i$ .Ev(n_n, ax)

stage 2 session key: $ak[0:1024]$

Establishing ak – Stage 2

Client (has n_n)

Server (has n_n)

$a \leftarrow\$ \{0,1\}^{2048}$

$k \parallel iv \leftarrow$ SKDF.Ev(n_n, n_s)

n, n_s, c_1 ←

$c_1 \leftarrow\$$ HtE. Enc ($k \parallel iv$, … $g^a \bmod p$ …)

repeatable

$b \leftarrow\$ \{0,1\}^{2048}$

$c_2 \leftarrow\$$ HtE. Enc ($k \parallel iv$, … $g^b \bmod p$ …)

→ n, n_s, c_2

$ak \leftarrow g^{ab} \bmod p$

$ax \parallel \ldots \parallel aid \leftarrow$ SHA-1(ak)

if $aid \in \mathscr{S}_{\text{aid}}$ **then** wait for retry

if $h_i = h_2$ **then** retry

n, n_s, h_i ←

$h_i \leftarrow$ NH$_i$.Ev(n_n, ax)

stage 2 session key: $ak[0:1024]$

More custom primitives: SKDF, HtE, NH_i

- Custom key derivation function

$$\text{SKDF.Ev}(k,x) \rightarrow \text{SHA-1}\,(k \parallel x) \parallel \text{SHA-1}\,(x \parallel k) \parallel \text{SHA-1}\,(k \parallel k) \parallel k[0:32]$$

More custom primitives: SKDF, HtE, NH_i

- Custom key derivation function

$$\mathrm{SKDF.Ev}(k,x) \rightarrow \text{SHA-1}\,(k \parallel x) \parallel \text{SHA-1}\,(x \parallel k) \parallel \text{SHA-1}\,(k \parallel k) \parallel \boxed{k[0:32]}$$

More custom primitives: SKDF, HtE, NH_i

- Custom key derivation function

$$\mathrm{SKDF.Ev}(k,x) \rightarrow \mathrm{SHA\text{-}1}\,(k \parallel x) \parallel \mathrm{SHA\text{-}1}\,(x \parallel k) \parallel \mathrm{SHA\text{-}1}\,(k \parallel k) \parallel k[0:32]$$

- Custom *hash-then-encrypt* symmetric encryption scheme

$$\mathrm{HtE.Enc}(k \parallel iv, m) \rightarrow \mathrm{AES\text{-}256\text{-}IGE.Enc}(k, iv, \mathrm{SHA\text{-}1}\,(m) \parallel m \parallel \mathrm{pad})$$

More custom primitives: SKDF, HtE, NH_i

- Custom key derivation function

$$\mathrm{SKDF.Ev}(k,x) \rightarrow \mathrm{SHA\text{-}1}\ (k \parallel x) \parallel\ \mathrm{SHA\text{-}1}\ (x \parallel k) \parallel\ \mathrm{SHA\text{-}1}\ (k \parallel k) \parallel\ k[0:32]$$

- Custom *hash-then-encrypt* symmetric encryption scheme

$$\mathrm{HtE.Enc}(k \parallel iv, m) \rightarrow \mathrm{AES\text{-}256\text{-}IGE.Enc}(k, iv,\ \mathrm{SHA\text{-}1}\ (m) \parallel m \parallel \mathrm{pad})$$

- Custom stateful hash

$$\mathrm{NH}_i\mathrm{.Ev}(k,x) \rightarrow\ \mathrm{SHA\text{-}1}\ (k \parallel 0i \parallel x)[32:160]\ \text{ for } i \in \{1,2,3\}$$

More custom primitives: SKDF, HtE, NH_i

- Custom key derivation function

$$\mathrm{SKDF.Ev}(k,x) \rightarrow \boxed{\text{SHA-1}}(k \parallel x) \parallel \boxed{\text{SHA-1}}(x \parallel k) \parallel \boxed{\text{SHA-1}}(k \parallel k) \parallel\ k[0:32]$$

- Custom *hash-then-encrypt* symmetric encryption scheme

$$\mathrm{HtE.Enc}(k \parallel iv, m) \rightarrow \text{AES-256-IGE.Enc}(k, iv, \boxed{\text{SHA-1}}(m) \parallel m \parallel \text{pad})$$

- Custom stateful hash

$$\mathrm{NH}_i\text{.Ev}(k,x) \rightarrow \boxed{\text{SHA-1}}(k \parallel 0i \parallel x)[32:160] \quad \text{for } i \in \{1,2,3\}$$

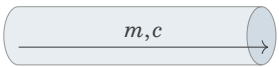
Binding a temporary ak_t to a long-term ak – Stage 3

Let $ak_{v1} = ak[0:1024]$, MTP-CH = channel using ak_t.

Client (has (ak_{v1}, aid), MTP-CH)

$m \leftarrow \ldots aid_t \parallel aid \ldots$

$c \leftarrow\!\$$ CHv1 .Enc(ak_{v1}, m)

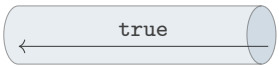


Server (has T, MTP-CH)

$ak_{v1} \leftarrow$ T$[aid]$

$m' \leftarrow$ CHv1 .Dec(ak_{v1}, c)

check $m' = m$



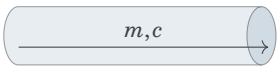
Binding a temporary ak_t to a long-term ak – Stage 3

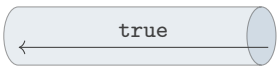
Let $ak_{v1} = ak[0:1024]$, MTP-CH = channel using ak_t.

Client (has (ak_{v1}, aid), MTP-CH)

$m \leftarrow \ldots aid_t \parallel aid \ldots$

$c \leftarrow\$$ CHv1.Enc(ak_{v1}, m)





Server (has T, MTP-CH)

$ak_{v1} \leftarrow T[aid]$

$m' \leftarrow$ CHv1.Dec(ak_{v1}, c)

check $m' = m$

Overview of this talk

1. Introduction
2. Security model
3. Telegram protocols
4. Results
5. Takeaways

Results

AKE security of 2-stage protocol (simplified)

Let $\mathscr{A}$ be an adversary against the 2-stage protocol with at most n sessions and s servers, using a fixed Diffie-Hellman group $\mathbb{G}$ of order q.

Then $\exists\, \mathscr{D}_{\text{IND-CCA}}, \mathscr{A}_{\text{INT-PTXT}}, \mathscr{D}_{\text{IND-KEY}}, \mathscr{D}_{\text{S-EXP}}, \mathscr{D}_{\text{DDH}}, \mathscr{D}_{\text{OTPRF}}$:

$$
\begin{aligned}
\text{Adv}_{\text{2-stage}}(\mathscr{A}) \leq\ & \frac{n^2}{2^{384}} + \frac{sn^2}{2^{497}} + \frac{16sn^3}{q} \\
& + 4sn(n+1) \cdot \Big(2 \cdot \text{Adv}_{\text{TOAEP}^+}(\mathscr{D}_{\text{IND-CCA}}) \\
& \qquad + \text{Adv}_{\text{HtE,SKDF}}(\mathscr{A}_{\text{INT-PTXT}}) \Big) \\
& + 4sn^2 \cdot \Big(\text{Adv}_{\text{SKDF,NH}}(\mathscr{D}_{\text{IND-KEY}}) + \text{Adv}_{\mathbb{G},q}(\mathscr{D}_{\text{S-EXP}}) \\
& \qquad + \text{Adv}_{\mathbb{G},q}(\mathscr{D}_{\text{DDH}}) + \text{Adv}_{\text{SHACAL-1}}(\mathscr{D}_{\text{OTPRF}}) \Big).
\end{aligned}
$$

Results

AKE security of 2-stage protocol (simplified)

Let $\mathscr{A}$ be an adversary against the 2-stage protocol with at most n sessions and s servers, using a fixed Diffie-Hellman group $\mathbb{G}$ of order q.

Then $\exists\, \mathscr{D}_{\text{IND-CCA}}, \mathscr{A}_{\text{INT-PTXT}}, \mathscr{D}_{\text{IND-KEY}}, \mathscr{D}_{\text{S-EXP}}, \mathscr{D}_{\text{DDH}}, \mathscr{D}_{\text{OTPRF}}$:

$$\begin{aligned}
\mathsf{Adv}_{\text{2-stage}}(\mathscr{A}) \leq\ & \frac{n^2}{2^{384}} + \frac{sn^2}{2^{497}} + \frac{16sn^3}{q} \\
& + 4sn(n+1) \cdot \Big(2 \cdot \mathsf{Adv}_{\text{TOAEP}^+}(\mathscr{D}_{\text{IND-CCA}}) \\
& \quad + \mathsf{Adv}_{\text{HtE,SKDF}}(\mathscr{A}_{\text{INT-PTXT}})\Big) \\
& + 4sn^2 \cdot \Big(\mathsf{Adv}_{\text{SKDF,NH}}(\mathscr{D}_{\text{IND-KEY}}) + \mathsf{Adv}_{\mathbb{G},q}(\mathscr{D}_{\text{S-EXP}}) \\
& \quad + \mathsf{Adv}_{\mathbb{G},q}(\mathscr{D}_{\text{DDH}}) + \mathsf{Adv}_{\text{SHACAL-1}}(\mathscr{D}_{\text{OTPRF}})\Big).
\end{aligned}$$

Results

IND-CCA of Telegram OAEP+ (simplified)

We model SHA-256 as a random oracle and AES-256 as an ideal cipher.
Let $\mathscr{D}_{\text{IND-CCA}}$ be an adversary against TOAEP$^+$, making q_e encryption and q_d decryption queries, q_H random oracle queries, and q_c ideal cipher queries.
Assume $q_e, q_d, q_H, q_c \leq 2^{126}$ so that $q_d \cdot q_H \leq 2^{134}$.
Then $\exists\, \mathscr{A}_{\text{OW}}$ against the one-wayness of RSA :

$$\text{Adv}_{\text{TOAEP}^+}(\mathscr{D}_{\text{IND-CCA}}) \leq 2 \cdot \text{Adv}_{\text{RSA}}(\mathscr{A}_{\text{OW}}) + 2^{-116}.$$

Results

IND-CCA of Telegram OAEP+ (simplified)

We model SHA-256 as a random oracle and AES-256 as an ideal cipher.
Let $\mathscr{D}_{\text{IND-CCA}}$ be an adversary against TOAEP$^+$, making q_e encryption and q_d decryption queries, q_H random oracle queries, and q_c ideal cipher queries.
Assume $q_e, q_d, q_H, q_c \leq 2^{126}$ so that $q_d \cdot q_H \leq 2^{134}$.
Then $\exists\, \mathscr{A}_{\text{OW}}$ against the one-wayness of RSA :

$$\text{Adv}_{\text{TOAEP}^+}(\mathscr{D}_{\text{IND-CCA}}) \leq 2 \cdot \text{Adv}_{\text{RSA}}(\mathscr{A}_{\text{OW}}) + 2^{-116}.$$

Results

INT-PTXT of Telegram's Hash-then-Encrypt scheme

We model AES-256 as an ideal cipher.
Let $\mathscr{A}_{\text{INT-PTXT}}$ be an adversary against HtE and SKDF, making encryption/decryption queries for t blocks, and making q_c queries to the ideal cipher.
Then $\exists\ \mathscr{A}_{\text{SPR}}, \mathscr{A}_{\text{UPREF}}, \mathscr{A}_{\text{USUFF}}$:

$$\begin{aligned}\text{Adv}_{\text{HtE,SKDF}}(\mathscr{A}_{\text{INT-PTXT}}) \leq \frac{t^2}{2^{128}} + \frac{q_c}{2^{256}} + \text{Adv}_{\text{SHA-1}}(\mathscr{A}_{\text{SPR}}) \\ + 2 \cdot \text{Adv}_{\text{SHACAL-1}}(\mathscr{D}_{\text{3TPRF}}) + \text{Adv}(\mathscr{A}_{\text{EM}})\ .\end{aligned}$$

Results

INT-PTXT of Telegram's Hash-then-Encrypt scheme

We model AES-256 as an ideal cipher.
Let $\mathscr{A}_{\text{INT-PTXT}}$ be an adversary against HtE and SKDF, making encryption/decryption queries for t blocks, and making q_c queries to the ideal cipher.
Then $\exists\ \mathscr{A}_{\text{SPR}}, \mathscr{A}_{\text{UPREF}}, \mathscr{A}_{\text{USUFF}}$:

$$\text{Adv}_{\text{HtE,SKDF}}(\mathscr{A}_{\text{INT-PTXT}}) \leq \frac{t^2}{2^{128}} + \frac{q_c}{2^{256}} + \text{Adv}_{\text{SHA-1}}(\mathscr{A}_{\text{SPR}}) + 2 \cdot \text{Adv}_{\text{SHACAL-1}}(\mathscr{D}_{\text{3TPRF}}) + \text{Adv}(\mathscr{A}_{\text{EM}}).$$

Results

AKE security of 3-stage protocol (simplified)

Let $\mathscr{A}$ be an adversary against the 3-stage protocol. Then $\exists\, \mathscr{A}_{\text{EUF-CMA}}, \mathscr{A}_{\text{KE}}$:

$$\text{Adv}_{\text{3-stage}}(\mathscr{A}) \leq \text{Adv}_{\text{2-stage}}(\mathscr{A}_{\text{KE}}) + 2 \cdot \text{Adv}_{\text{CHv1}}(\mathscr{A}_{\text{EUF-CMA}}) .$$

Results

AKE security of 3-stage protocol (simplified)

Let $\mathscr{A}$ be an adversary against the 3-stage protocol. Then $\exists\ \mathscr{A}_{\text{EUF-CMA}}, \mathscr{A}_{\text{KE}}$:

$$\text{Adv}_{\text{3-stage}}(\mathscr{A}) \leq \boxed{\text{Adv}_{\text{2-stage}}(\mathscr{A}_{\text{KE}})} + 2 \cdot \boxed{\text{Adv}_{\text{CHv1}}(\mathscr{A}_{\text{EUF-CMA}})}.$$

Results

EUF-CMA of MTProto 1.0 channel encryption

Let $\mathscr{A}$ be an adversary against CHv1 with at most s' long-term symmetric keys, making at most q_v verification queries. Then $\exists\, \mathscr{D}_{\text{4PRF}}, \mathscr{A}_{\text{UPCR}}$:

$$\text{Adv}_{\text{CHv1}}(\mathscr{A}) \leq s' \cdot \left(2 \cdot \text{Adv}_{\text{SHACAL-1}}(\mathscr{D}_{\text{4PRF}}) + \frac{q_v}{2^{128}} + \text{Adv}_{\text{SHA-1}}(\mathscr{A}_{\text{UPCR}})\right).$$

Results

EUF-CMA of MTProto 1.0 channel encryption

Let $\mathscr{A}$ be an adversary against CHv1 with at most s' long-term symmetric keys, making at most q_v verification queries. Then $\exists\, \mathscr{D}_{\text{4PRF}}, \mathscr{A}_{\text{UPCR}}$:

$$\text{Adv}_{\text{CHv1}}(\mathscr{A}) \leq s' \cdot \left(2 \cdot \text{Adv}_{\text{SHACAL-1}}(\mathscr{D}_{\text{4PRF}}) + \frac{q_v}{2^{128}} + \text{Adv}_{\text{SHA-1}}(\mathscr{A}_{\text{UPCR}})\right).$$

New assumptions

New assumptions

- 3TPRF, 4PRF: pseudorandomness of SHACAL-1 on fixed input with partial key leakage

New assumptions

- 3TPRF, 4PRF: pseudorandomness of SHACAL-1 on fixed input with partial key leakage
- SPR, UPCR: second-preimage resistance of SHA-1 with tweaks

New assumptions

- 3TPRF, 4PRF: pseudorandomness of SHACAL-1 on fixed input with partial key leakage
- SPR, UPCR: second-preimage resistance of SHA-1 with tweaks
- IND-KEY: indistinguishability of key reuse with SHA-1-based functions

Overview of this talk

1. Introduction
2. Security model
3. Telegram protocols
4. Results
5. Takeaways

Takeaways

- MTProto 2.0 is a brittle protocol

Takeaways

- MTProto 2.0 is a brittle protocol
  - SHA-1 is misused for many purposes where better solutions exist

Takeaways

- MTProto 2.0 is a brittle protocol
  - SHA-1 is misused for many purposes where better solutions exist
  - Plaintext checks prevent attacks, but place more burden on implementers

Takeaways

- MTProto 2.0 is a brittle protocol
  - SHA-1 is misused for many purposes where better solutions exist
  - Plaintext checks prevent attacks, but place more burden on implementers
  - Some protocol features such as retry loops only create complexity

Takeaways

- MTProto 2.0 is a brittle protocol
  - SHA-1 is misused for many purposes where better solutions exist
  - Plaintext checks prevent attacks, but place more burden on implementers
  - Some protocol features such as retry loops only create complexity
- Ad-hoc design leads to sub-par security guarantees

Takeaways

- MTProto 2.0 is a brittle protocol
  - SHA-1 is misused for many purposes where better solutions exist
  - Plaintext checks prevent attacks, but place more burden on implementers
  - Some protocol features such as retry loops only create complexity
- Ad-hoc design leads to sub-par security guarantees

- Implementation details such as encoding schemes *can* be modelled faithfully

Takeaways

- MTProto 2.0 is a brittle protocol
  - SHA-1 is misused for many purposes where better solutions exist
  - Plaintext checks prevent attacks, but place more burden on implementers
  - Some protocol features such as retry loops only create complexity
- Ad-hoc design leads to sub-par security guarantees

- Implementation details such as encoding schemes *can* be modelled faithfully
- Telegram is not as broken as you thought

Takeaways

- MTProto 2.0 is a brittle protocol
  - SHA-1 is misused for many purposes where better solutions exist
  - Plaintext checks prevent attacks, but place more burden on implementers
  - Some protocol features such as retry loops only create complexity
- Ad-hoc design leads to sub-par security guarantees

- Implementation details such as encoding schemes *can* be modelled faithfully
- Telegram is not as broken as you thought (unless you can break our assumptions?)

Thank you!

See the full version of the paper at ia.cr/2025/451

Any questions?