Analysis of the Threema Secure Messenger

Kenny Paterson, Matteo Scarlata, Kien Tuong Truong
What is Threema?

- An “end-to-end encrypted instant messaging application” for Android and iOS
- 11 million private users worldwide
Part I
Threema, the Protocol
Bird’s Eye View of the Threema Protocol

Two layers of encryption

(sk_A, pk_A) (sk_B, pk_B)
E2E Protocol

$$\text{(sk}_A, \text{pk}_A) \rightleftharpoons \text{DH}($$sk_A, pk_B$$) \rightleftharpoons \text{DH}($$sk_B, pk_A$$) \rightleftharpoons \text{(sk}_B, \text{pk}_B$$)

Encryption key

No Forward Secrecy ❌
No Post-Compromise Security ❌
C2S Protocol

Establishes a client-server session key through an **authenticated key exchange**
C2S: Client Authentication

\[(sk_A, pk_A) \xrightarrow{} (sk_S, pk_S)\]

\[(esk_A, epk_A) \xleftarrow{} KeyGen()\]

\[(esk_S, epk_S) \xleftarrow{} KeyGen()\]

\[vouch \xleftarrow{} epk_A\]

Handshake Sub-protocol

Transport Sub-protocol
Part II
Attacks on Threema
Attacks Found

- Attack: C2S Ephemeral Key Compromise
- Attack: Vouch Box Forgery

Compromised Threema Server

External/Network Attacker

- Attack: Message Reordering/Omission
- Attack: Message Replay/Reflection
- Attack: Kompromat

Physical Device Access (“Compelled Access”)

- Attack: Compression-Side Channel on Threema Safe
- Attack: Threema ID Export
Deja-vu?

\[(s_{kS}, \, pk_{S}) \quad (s_{kA}, \, pk_{A}) \quad (s_{kB}, \, pk_{B})\]
Deja-vu?

These two keys end up being the same!

Assume we managed to make $\text{C2S}$ and $\text{E2E}$ collide. What can we do now?

$$\text{vouch} = \text{epk}_A$$

$$(\text{sk}_S, \text{pk}_S)$$

$$(\text{sk}_A, \text{pk}_A)$$

$$(\text{N/A, pk}_S)$$

$$(\text{sk}_B, \text{pk}_B)$$
Key collision to Protocol Confusion

- **C2S** x **E2E** cross-protocol attack
- Sending a text message... compromises client authentication forever!
Two issues to still discuss

Find a suitable ephemeral key $epk^*$

**Task 1: Getting That Key**

Claim the server’s public key as ours

**Task 2: The Bamboozling**

(N/A, $pk_s$)  
($sk_B$, $pk_B$)
Task 1: Getting that Key

- **Problem:** getting a valid epk* turns out to be computationally intensive!
- Requires randomly sampling $2^{51}$ keys!

---

Matteo Scarlata 9:04 PM
Hi Kenny, we ran some quick estimates. 8192 cores for a week on AWS would cost ~180,000 USD.

Kenny Paterson 9:51 PM
Yikes.
Task 1: Getting that Key

kennyog
@kennyog

I’d like to borrow 8192 cores for a week. Anyone out there got some spare compute lying around to help out with a cool research project?

9:53 PM · Sep 27, 2022
Some optimizations and 8100 core-days later...

esk = 504ac13e00000000003000336d612d322d3232313231392d30332d303323000

epk = 0175396a36df93276a6ae0a496d4bb5edf8331d79b573a2dcc813bdca1524101

u9j6 ¶’jj� hijo 1□W:-؛مRA
Task 2: The Bamboozling

- Threema Gateway: paid API
- Can register accounts with arbitrary public keys
- Without proof of possession of the corresponding private key!
Part III

Conclusion
Mitigations

- Attack: C2S Ephemeral Key Compromise
- Attack: Vouch Box Forgery
- Change vouchbox derivation
  - Attack: Message Reordering/Omission
  - Attack: Message Replay/Reflection
  - Attack: Kompromat
- Metadata box mandatory
- Better key separation
- Attack: Compression-Side Channel on Threema Safe
- Attack: Threema ID Export
- Disable compression in backups
- Track ephemeral keys
Lessons Learnt: Rolling your Protocol

“[Threema has] a client-server protocol modelled after CurveCP, an end-to-end encryption protocol based on the NaCl library [...]

<table>
<thead>
<tr>
<th>Key pair</th>
<th>Nonce format</th>
</tr>
</thead>
<tbody>
<tr>
<td>The server’s long-term secret key $s$ and long-term public key $S$. The client knows $S$ before making a CurveCP connection.</td>
<td>The 8-byte ASCII string &quot;CurveCPK&quot; followed by a 16-byte compressed nonce.</td>
</tr>
<tr>
<td>The client’s long-term secret key $c$ and long-term public key $C$. Some servers differentiate between clients on the basis of known values of $C$.</td>
<td>The 8-byte ASCII string &quot;CurveCPV&quot; followed by a 16-byte compressed nonce.</td>
</tr>
<tr>
<td>The server’s short-term secret key $s'$ and short-term public key $S'$. These are specific to this connection and help provide forward secrecy.</td>
<td>The 16-byte ASCII string &quot;CurveCP-server-M&quot; followed by an 8-byte compressed nonce. The compressed nonce represents a 64-bit integer in little-endian form. These integers are generated in increasing order.</td>
</tr>
<tr>
<td>The client’s short-term secret key $c'$ and short-term public key $C'$. These are also specific to this connection.</td>
<td>A 16-byte ASCII string followed by an 8-byte compressed nonce. The string is &quot;CurveCP-client-H&quot; for a Hello packet, &quot;CurveCP-client-I&quot; for an Initiate packet, or &quot;CurveCP-client-M&quot; for a Message packet. The compressed nonce represents a 64-bit integer in little-endian form. These integers are generated in increasing order.</td>
</tr>
</tbody>
</table>
Lessons Learnt: Cross-Protocol Interactions

“Matrix’s encryption is based on the Double Ratchet Algorithm popularised by Signal”

Practically-exploitable Cryptographic Vulnerabilities in Matrix

Martin R. Albrecht*, Sofia Celi‡, Benjamin Dowling§ and Daniel Jones§
* King’s College London, martin.albrecht@kcl.ac.uk

Olm x Megolm

Confidentiality break!
Lessons Learnt: Proactive Security

PCS??
IBEX
E2E
C2S
Lessons Learnt

- Don’t roll your own crypto protocols

- But if you do:
  - Beware of cross-protocol interactions
  - You need provable and proactive security

Thank you for listening!
Questions?
kitruong@ethz.ch

https://breakingthe3ma.app