Analysis of the Threema Secure Messenger

Kenny Paterson, Matteo Scarlata, Kien Tuong Truong

Set to appear at USENIX Security 2023
What is Threema?
What is Threema?

- An “end-to-end encrypted instant messaging application” for Android and iOS
- Released in 2012
What is Threema?

- An “end-to-end encrypted instant messaging application” for Android and iOS
- Released in 2012

“Threema is **100% Swiss Made**, hosts its own servers in Switzerland, and, unlike US services (which are subject to the CLOUD Act, for example), it is fully GDPR-compliant.”
Who uses Threema?

- 11 million private users worldwide\(^1\)
- Various organizations and political entities:

\(^1\) [https://threema.ch/en/about](https://threema.ch/en/about) (Last checked 19 Mar 2023)
Bird’s Eye View of the Threema Protocol
Bird’s Eye View of the Threema Protocol
Bird’s Eye View of the Threema Protocol

Two layers of encryption
1. Attacking the End-to-End Protocol
2. Attacking the Client-to-Server Protocol
3. Attacking Backup Methods
4. Conclusions
E2E Protocol

E2E Protocol

Encrypted under

\[ K = DH(sk_A, pk_B) = DH(sk_B, pk_A) \]
E2E Protocol

(\text{sk}_A, \text{pk}_A) \quad \text{Encrypted under} \quad K = \text{DH}(\text{sk}_A, \text{pk}_B) = \text{DH}(\text{sk}_B, \text{pk}_A) \quad (\text{sk}_B, \text{pk}_B)

No Forward Secrecy!
E2E Protocol: Message Structure

E.g. 0x01 || Hello! || 0x02 0x02

K

msg type  message content  PKCS7 Padding

nonce

AE.Enc

metadata

metadata-box

nonce

encrypted msg
E2E Protocol: Message Structure
E2E Protocol: Message Structure

metadata

msg type | message content | PKCS7 Padding

nonce

K

KDF

AE.Enc

metadata

metadata-box

nonce

encrypted msg
1. Attacking the End-to-End Protocol
2. Attacking the Client-to-Server Protocol
3. Attacking Backup Methods
4. Conclusions
C2S Protocol

Establishes a client-server session key through an authenticated key exchange

\((sk_A, pk_A)\)

\((sk_B, pk_B)\)
The C2S Protocol

\[(\text{esk}_A, \text{epk}_A) \leftarrow \text{KeyGen}()\]

\[(\text{esk}_S, \text{epk}_S) \leftarrow \text{KeyGen}()\]

* Simplified, details omitted
The C2S Protocol

\( (sk_A, pk_A) \leftarrow \text{KeyGen}() \)

\( (sk_S, pk_S) \leftarrow \text{KeyGen}() \)

\( (esk_A, epk_A) \)

\( K_{\text{session}} \leftarrow \text{DH}(esk_A, epk_S) \)

\( K_{\text{vouch}} \leftarrow \text{DH}(sk_A, pk_S) \)

\( \text{vouch} \leftarrow \text{Enc}(K_{\text{vouch}}, epk_A) \)

\( \text{Enc}(K_{\text{session}}, ID_A || \text{vouch} || \ldots) \)

* Simplified, details omitted
The C2S Protocol

\[(esk_A, epk_A) \leftarrow \text{KeyGen()}\]

\[K_{\text{session}} \leftarrow \text{DH}(esk_A, epk_S)\]

\[K_{\text{vouch}} \leftarrow \text{DH}(sk_A, pk_S)\]

\[\text{vouch} \leftarrow \text{Enc}(K_{\text{vouch}}, epk_A)\]

\[\text{Enc}(K_{\text{session}}, ID_A \mid \mid \text{vouch} \mid \mid \ldots)\]

\[\ldots\]

\[\text{Enc}(K_{\text{session}}, m)\]

* Simplified, details omitted
The C2S Protocol: Vouch Box

\[ K_{vouch} \leftarrow \text{DH}(sk_A, pk_S) \]

\[ \text{vouch} \leftarrow \text{Enc}(K_{vouch}, epk_A) \]
The C2S Protocol: Vouch Box

\[ K_{\text{vouch}} \leftarrow \text{DH}(sk_A, pk_S) \]
\[ \text{vouch} \leftarrow \text{Enc}(K_{\text{vouch}}, epk_A) \]

DH(long-term, long-term)
The C2S Protocol: Vouch Box

$K_{vouch} \leftarrow \text{DH}(sk_A, pk_S)$  \hspace{1cm} \text{DH(long-term, long-term)}

$vouch \leftarrow \text{Enc}(K_{vouch}, epk_A)$  \hspace{1cm} \text{Enc(some value)}
The C2S Protocol: Vouch Box

What if we could find a special keypair \((esk, epk)\) such that:

\[
K_{\text{vouch}} \leftarrow \text{DH}(sk_A, pk_S) \quad \text{DH(long-term, long-term)}
\]

\[
vouch \leftarrow \text{Enc}(K_{\text{vouch}}, epk_A) \quad \text{Enc(some value)}
\]
The C2S Protocol: Vouch Box

\[ K_{\text{vouch}} \leftarrow \text{DH}(sk_A, pk_S) \quad \text{DH(long-term, long-term)} \]
\[ \text{vouch} \leftarrow \text{Enc}(K_{\text{vouch}}, epk_A) \quad \text{Enc(some value)} \]

What if we could find a special keypair \((esk, epk)\) such that:

\[ epk = 0x01 || \sigma || 0x01 \]
The C2S Protocol: Vouch Box

\[ K_{vouch} \leftarrow \text{DH}(sk_A, pk_S) \quad \text{DH(long-term, long-term)} \]

\[ \text{vouch} \leftarrow \text{Enc}(K_{vouch}, epk_A) \quad \text{Enc(some value)} \]

What if we could find a special keypair \((esk, epk)\) such that:

\[ epk = 0x01 \mid \sigma \mid 0x01 \]

UTF-8 valid string of 30B
Attacking the C2S Protocol

\[(sk_A, pk_A) \quad \text{E2E} \]
Attacking the C2S Protocol

My E2E public key is \( pks \)!
Attacking the C2S Protocol

My E2E public key is \( pk_s \)!

Can you send me \( \sigma \) as a text message?
Attacking the C2S Protocol

My E2E public key is pk_s!

Can you send me σ as a text message?

K←DH(sk_A, pk_s)
Attacking the C2S Protocol

(user, pk_A)  

My E2E public key is pk_S!

Can you send me $\sigma$ as a text message?

K ← DH(sk_A, pk_S)

Enc(K, 0x01 || $\sigma$ || 0x01)

msg type  content  PKCS7 Padding
Attacking the C2S Protocol

\[ K \leftarrow \text{DH}(sk_A, pk_S) \]

\[ = K_{\text{vouch}} \text{ in C2S} \]

\[ \text{Enc}(K, 0x01 \ || \ \sigma \ || \ 0x01) \]

\[ = \text{Enc}(K_{\text{vouch}}, epk) \]

A valid vouch box appears!

My E2E public key is \( pk_S \)!

Can you send me \( \sigma \) as a text message?
Vouch Box Forgery

- **C2S x E2E** cross-protocol attack:
- **C2S x E2E** cross-protocol attack:

- Sending a text message...

  compromises client authentication *forever!*
Vouch Box Forgery

- **C2S x E2E** cross-protocol attack:
- Sending a text message...
  compromises client authentication **forever**!
Vouch Box Forgery

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

- Find a suitable ephemeral key \( \text{epk} \) (AKA \textit{Getting That Key})
- Claim the server’s public key as ours (AKA \textit{The Bamboozling})
Part 1: Getting That Key

epk = 0x01 || σ || 0x01

UTF-8 valid string of 30B

Requires sampling $2^{51}$ keys!
Part 1: Getting That Key

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

Kenny Paterson 9:51 PM
Yikes.
Part 1: Getting That Key

Some optimizations and 8100 core-days later...

esk = 504ac13e00000000003000336d612d322d3232313231392d30332d303323000

epk = 0175396a36df93276a6ae0a496d4bb5edf8331d79b573a2dcc813bdca1524101

u9j6㎡jjख़्वचW:-;מRA
Part 2: The Bamboozling
Part 2: The Bamboozling

- Threema Gateway: paid API
- Can register accounts with arbitrary public keys
- Without proof of possession of the corresponding private key!
Part 2: The Bamboozling

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

=> *LYTAAAS
Contents

1. Attacking the End-to-End Protocol
2. Attacking the Client-to-Server Protocol
3. Attacking Backup Methods
4. Conclusions
Threema Safe

```
{
  "identity": "XXXXXXXX",
  "privatekey": <base64 encoded key>,
  "contacts": [
    {
      "id": "YYYYYYYY",
      "nickname": "...",
    },
    ...
  ]
}
```
Threema Safe

```json
{
    "identity": "XXXXXXXX",
    "privatekey": <base64 encoded key>,
    "contacts": [
        {
            "id": "YYYYYYYY",
            "nickname": "...",
        },
        ...
    ]
}
```

Password

KDF

Backup ID

Compress then Encrypt?
Threema Safe

```json
{
    "identity": "XXXXXXXX",
    "privatekey": "<base64 encoded key>",
    "contacts": [
        {
            "id": "YYYYYYYY",
            "nickname": "...",
        },
        ...
    ]
}
```

Attacker has partial control of backup content

Compress then Encrypt?

Attacker has partial control of backup content
Private Key Extraction: Feasibility
Private Key Extraction: Feasibility

Start Threema
Observe Backup Size
Drop Connection
Change Username
Kill Threema
Private Key Extraction: Feasibility

1. Start Threema
2. Observe Backup Size
3. Drop Connection
4. Change Username
5. Kill Threema

~16k
Private Key Extraction: Feasibility

Start Threema

Observe Backup Size

~16k

01 02 03 04 05

Kill Threema

Change Username

Drop Connection

Attack: Compression-Side Channel on Threema Safe
Attacks Found

- Attack: C2S Ephemeral Key Compromise
- Attack: Vouch Box Forgery
- Attack: Message Reordering/Omission
- Attack: Message Replay/Reflection
- Attack: Kompromat
- Attack: Compression-Side Channel on Threema Safe
- Attack: Threema ID Export
Attacks Found

Attack: C2S Ephemeral Key Compromise

Attack: Vouch Box Forgery

Change vouchbox derivation

Metadata box mandatory
Better key separation

Attack: Message Reordering/Omission

Attack: Message Replay/Reflection

Attack: Kompromat

Disable compression in backups
Track ephemeral keys

Attack: Compression-Side Channel on Threema Safe

Attack: Threema ID Export
Lessons Learnt
Lessons Learnt: Rolling your Protocol
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 [...]

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 [...]
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 [...]

![Diagram showing Threema E2E and C2S protocols with cryptographic operations]
Lessons Learnt: Rolling your Protocol
Lessons Learnt: Rolling your Protocol

Is the Bridgefy App safe to use?

Yes! We use the Signal Protocol, which is industry-leading encryption […]
Lessons Learnt: Rolling your Protocol

Is the Bridgefy App safe to use?
Yes! We use the **Signal Protocol**, which is industry-leading encryption [...]
Lessons Learnt: Cross-Protocol Interactions
Lessons Learnt: Cross-Protocol Interactions

Threema:

E2E x C2S

Permanent authentication break
Lessons Learnt: Cross-Protocol Interactions

Threema:

E2E $\times$ C2S

$\Downarrow$

Permanent authentication break

Threema:

E2E $\times$ Reg

$\Downarrow$

Kompromat
Lessons Learnt: Cross-Protocol Interactions
Lessons Learnt: Cross-Protocol Interactions

Matrix’s encryption is based on the Double Ratchet Algorithm popularised by Signal
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
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 × Megolm
Confidentiality break!
Lessons Learnt: Proactive Security
Lessons Learnt: Proactive Security
Lessons Learnt: Proactive Security

E2E

C2S

+FS
Lessons Learnt: Proactive Security

E2E

C2S

+FS
Lessons Learnt: Proactive Security

E2E

C2S

+FS
Lessons Learnt: Proactive Security

IBEX

E2E

C2S

+FS
Lessons Learnt: Proactive Security

PCS??

IBEX

E2E

C2S

+FS
Lessons Learnt
Lessons Learnt

- Don’t roll your own crypto protocols
Lessons Learnt

- Don’t roll your own crypto protocols

- But if you do:
  - Beware of cross-protocol interactions
  - You need provable security
Lessons Learnt

- Don’t roll your own crypto protocols

- But if you do: Who should?
  - Beware of cross-protocol interactions
  - You need provable security
Lessons Learnt

- Don’t roll your own crypto protocols

- **But if you do:** Who should?
  - Beware of cross-protocol interactions
  - You need provable security

{kitruong,scmatteo}@ethz.ch
https://breakingthe3ma.app/
Lessons Learnt

- Don’t roll your own crypto protocols

- But if you do: **Who should?**
  - Beware of cross-protocol interactions
  - You need provable security

Thank you for listening!

Questions?

{kitruong,scmatteo}@ethz.ch

https://breakingthe3ma.app/
Bonus Slides
Attacks Found

- Attack: C2S Ephemeral Key Compromise
- Attack: Vouch Box Forgery
- Attack: Message Reordering/Omission
- Attack: Message Replay/Reflection
- Attack: Kompromat
- Attack: Compression-Side Channel on Threema Safe
- Attack: Threema ID Export

External/Network Attacker

Compromised Threema Server

Physical Device Access ("Compelled Access")
The C2S Protocol

$$(\text{sk}_A, \text{pk}_A) \xrightarrow{\text{KeyGen}()} (sk_S, pk_S)$$

$$(\text{esk}_A, \text{epk}_A) \xleftarrow{\text{KeyGen}()}$$

$K_{session} \xleftarrow{\text{X25519}} (\text{esk}_A, \text{epk}_S)\quad K_{vouch} \xleftarrow{\text{X25519}} (\text{sk}_A, \text{pk}_S)\quad \text{vouch} \leftarrow \text{Enc}(K_{vouch}, \text{epk}_A)$$

$K_{vough} \leftarrow \text{X25519}(\text{sk}_A, \text{pk}_S)\quad (\text{esk}_S, \text{epk}_S) \xrightarrow{\text{KeyGen}()}$
Private Key Extraction: Comments

- We exploit the client’s **retry behaviour**: if backup fails, at the next app startup, the backup will be sent again
- Assume we have an unlocked device, but the app is protected by a PIN
- Force-stop the application, then restart
- On iOS: app exits after a notification is received
Vouch Box Forgery: Key Search

```python
In [5]: for i, key in enumerate(keys):
    ...:     print(i, bytes(key[1:-1]).decode('UTF-8'))
```

86
Final List of Vulnerabilities/Attacks

Assumes an External Attacker

1. **Ephemeral Key-Compromise Impersonation**: Revealing the ephemeral key allows an attacker to fully impersonate the victim.

2. **Vouch box Forging**: Attacker can claim the server public key as their own. If a specially crafted message is sent by the victim, the attacker can fully impersonate the victim.
Final List of Vulnerabilities/Attacks

Assumes a Compromised/Malicious Threema Server

1. **Message Reordering**: The server can re-order messages, overwriting the timestamps to avoid detection

2. **Replay/Reflection Attacks**: If the user re-installs the app/changes devices, the server can replay and reflect messages

3. **Social Graph Discovery**: Even though Threema claims to be anonymous, identifying information is sent to the server for contact matching

4. **Kompromat** (patched): The server can forge arbitrary E2E messages on behalf of a user.
Final List of Vulnerabilities/Attacks

Assumes access to the device:

1. **CRIME on Threema Safe**: Attacker can leak the private key from 16k backup attempts
2. **Export ID**: Attacker can easily clone the application
Lessons Learnt: Rolling your Protocol

Is the Bridgefy App safe to use?

Yes! We use the **Signal Protocol**, which is industry-leading encryption [...]

---

**Breaking Bridgefy, again:**
**Adopting libsignal is not enough**

Martin R. Albrecht  
*Information Security Group, Royal Holloway, University of London*

Raphael Eikenberg  
*Applied Cryptography Group, ETH Zurich*

Kenneth G. Paterson  
*Applied Cryptography Group, ETH Zurich*

---

Bridgefy TransactionManager

1. `sess_{AM}, uid = uid_{M}`
2. `store(uid_{M}, PKB_{MA})`
3. `sess_{AM}, uid = uid_{B}, send(uid_{B}, M_{1})`
4. `s = lookup(uid_{B}), s == sess_{AM}`
5. `sess_{AM}, uid = uid_{AM}, enc(s, uid_{M}, M_{1})`

---

**Mallory**

- `sess_{MA}, uid = uid_{A}, store(uid_{A}, PKB_{AM})`
- `uid_{A}, PKB_{MA}`
- `uid_{B}, PKB_{AM}`
- `decryption successful`