Analysis of the **Threema** Secure Messenger

Kenny Paterson, Matteo Scarlata, **Kien Tuong Truong**

ETH zürich

APPLIED CRYPTO GROUP

What is Threema?

- An “end-to-end encrypted instant messaging application” for Android and iOS
- 11 million private users worldwide

Threema.

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

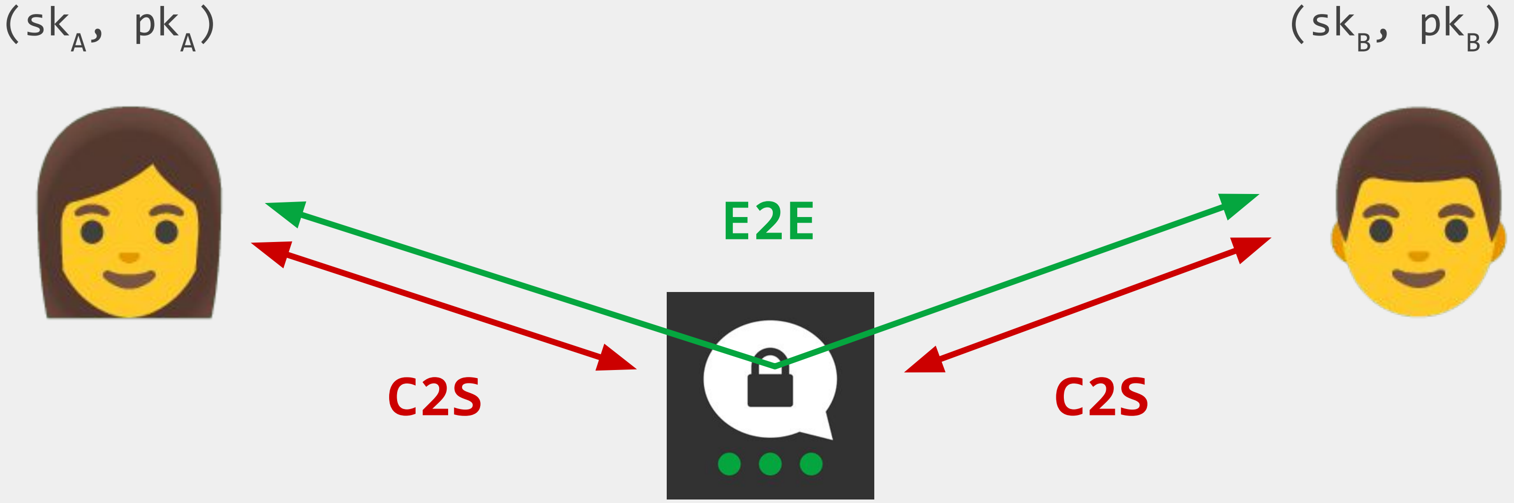
Mercedes-Benz

DEUTSCHE
BUNDESBANK
EUROSYSTEM

Emirates

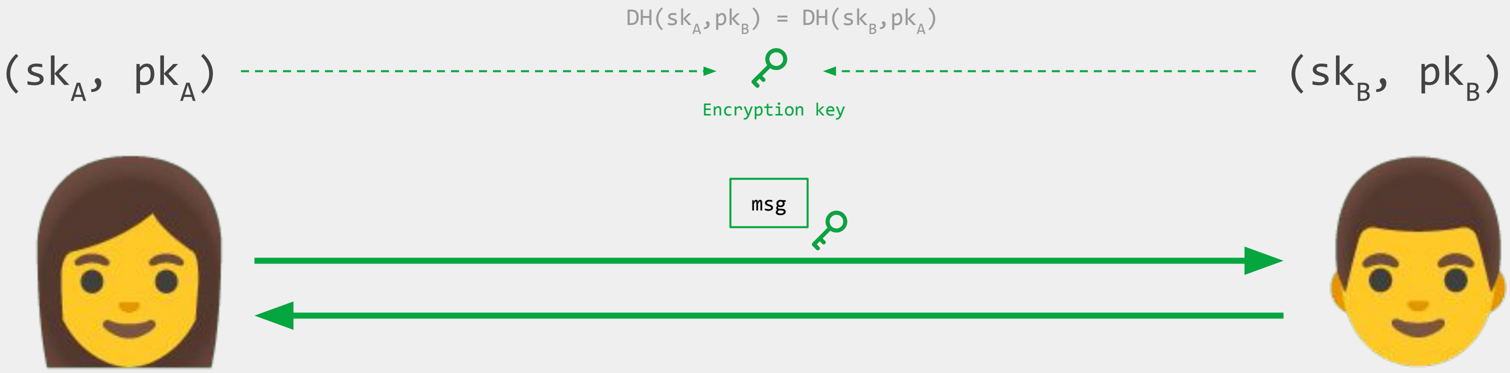
Part I
Threema, the Protocol

Bird's Eye View of the Threema Protocol



Two layers of encryption

E2E Protocol



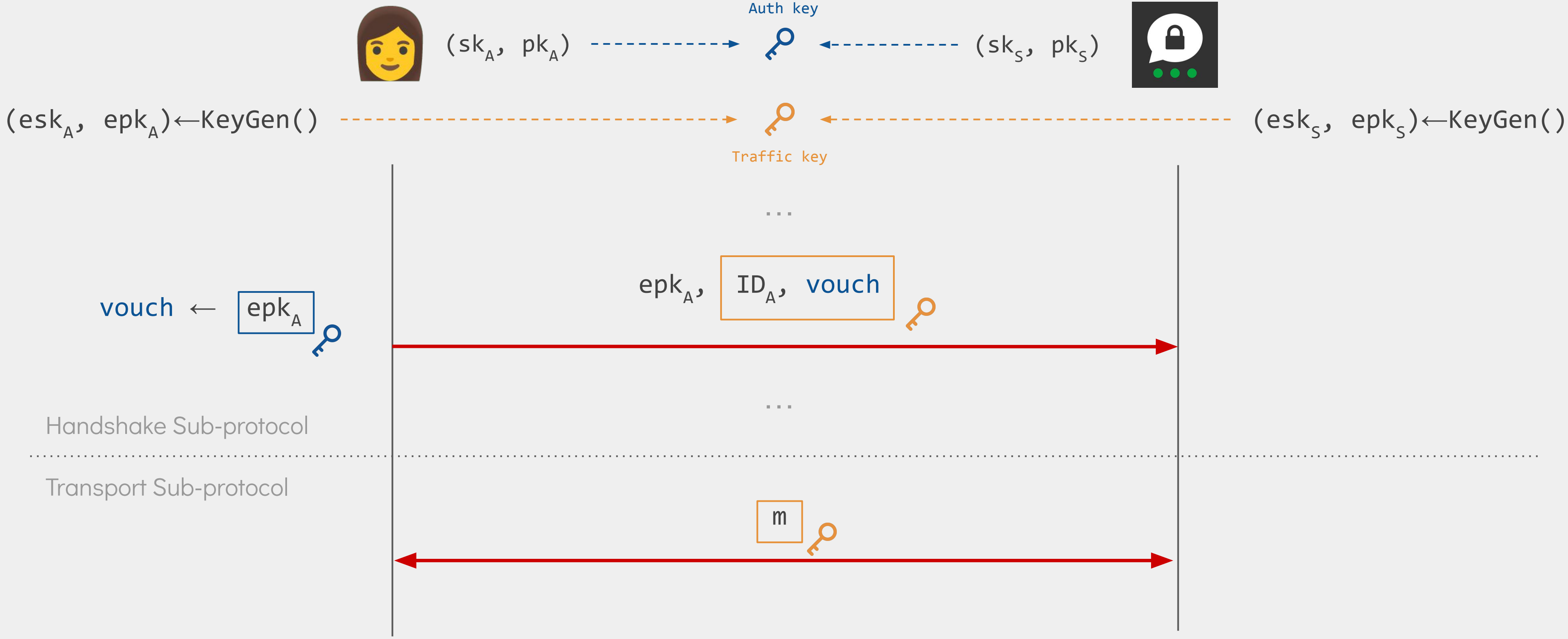
No Forward Secrecy ❌

No Post-Compromise Security ❌

C2S Protocol

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

C2S: Client Authentication



Part II

Attacks on Threema

Attacks Found

Attack: C2S Ephemeral Key Compromise

Attack: Vouch Box Forgery

External/Network Attacker

Attack: Message Reordering/Omission

Attack: Message Replay/Reflection

Attack: Kompromat

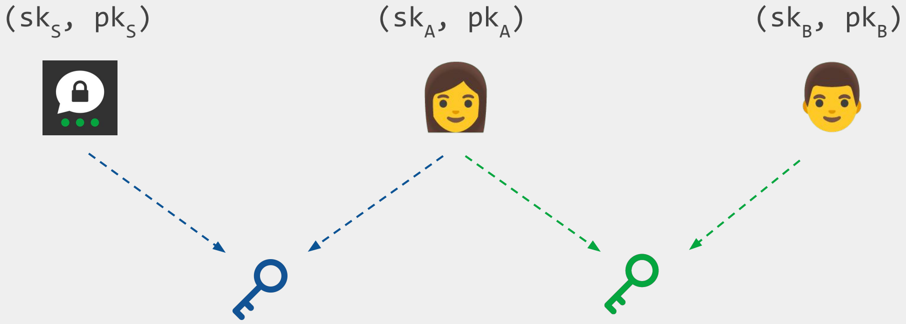
Compromised Threema Server

Attack: Compression-Side Channel on Threema Safe

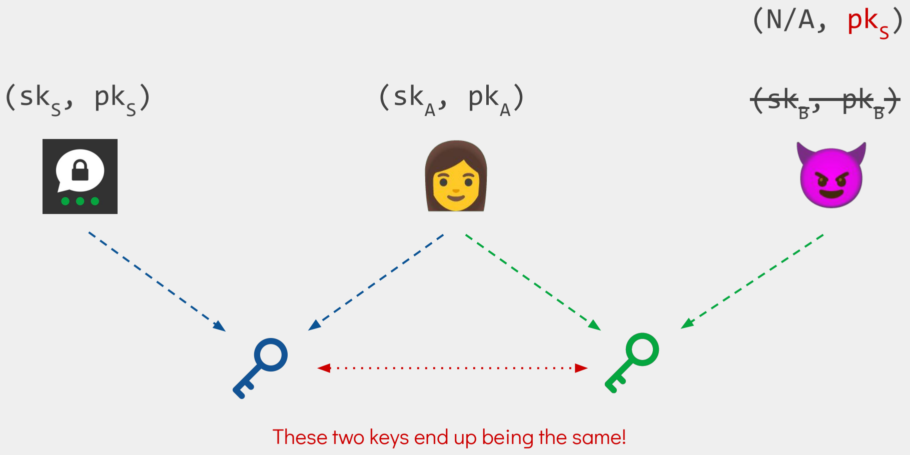
Attack: Threema ID Export

Physical Device Access
("Compelled Access")

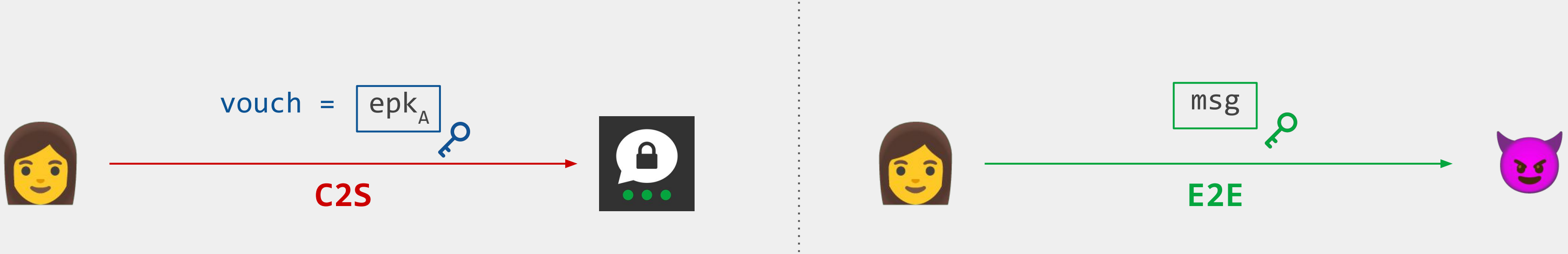
Deja-vu?



Deja-vu?



Assume we managed to make and collide. What can we do now?

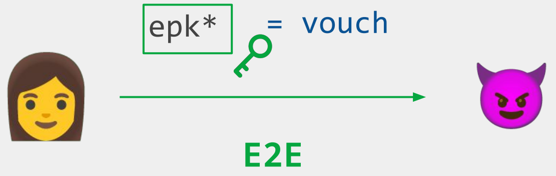


Key collision to Protocol Confusion

- **C2S** x **E2E** cross-protocol attack
- Sending a text message… compromises client authentication **forever**!

Loose Ends

Two issues to still discuss



Find a suitable ephemeral key epk*

Task 1: Getting That Key

$(\mathrm{N/A},\ pk_S)$

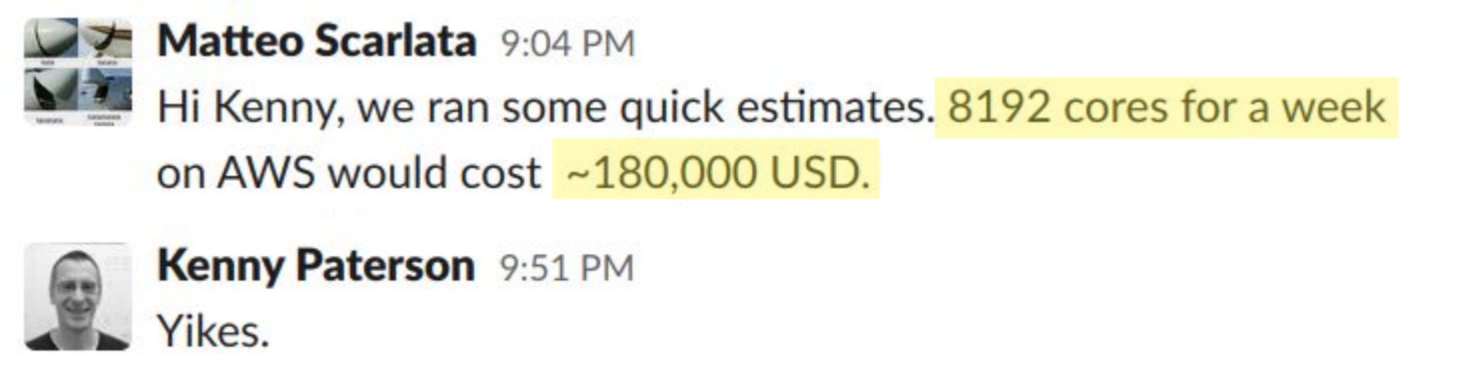
$\cancel{(sk_B,\ pk_B)}$

Claim the server's public key as ours

Task 2: The Bamboozling

Task 1: Getting that Key

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



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

Task 1: Getting that Key

Some optimizations and 8100 core-days later…

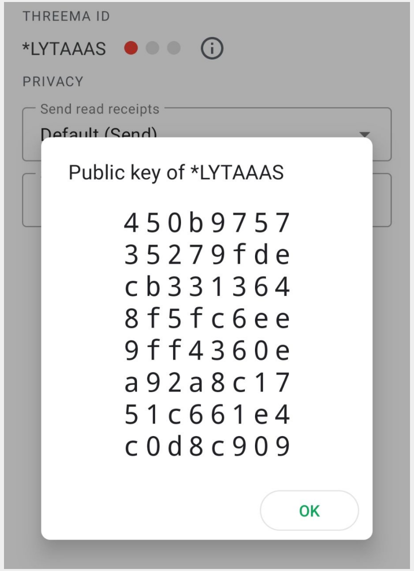
```
esk = 504ac13e000000000003000336d612d322d3232313231392d30332d3030323000

epk  = 0175396a36df93276a6ae0a496d4bb5edf8331d79b573a2dcc813bdca152410 1
```

u9j6□’jjखh^ɔ1□W:-;ƶRA

Task 2: The Bamboozling

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




```java
public static final byte[] SERVER_PUBKEY = new byte[] {
    (byte) 0x45, (byte) 0x0b, (byte) 0x97, (byte) 0x57,
    (byte) 0x35, (byte) 0x27, (byte) 0x9f, (byte) 0xde,
    (byte) 0xcb, (byte) 0x33, (byte) 0x13, (byte) 0x64,
    (byte) 0x8f, (byte) 0x5f, (byte) 0xc6, (byte) 0xee,
    (byte) 0x9f, (byte) 0xf4, (byte) 0x36, (byte) 0x0e,
    (byte) 0xa9, (byte) 0x2a, (byte) 0x8c, (byte) 0x17,
    (byte) 0x51, (byte) 0xc6, (byte) 0x61, (byte) 0xe4,
    (byte) 0xc0, (byte) 0xd8, (byte) 0xc9, (byte) 0x09
};
```

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*** *[...]”*

There are four different key pairs involved in a CurveCP connection, using four different types of nonces:

Key pair	Nonce format
The server's long-term secret key s and long-term public key S. The client knows S before making a CurveCP connection.	The 8-byte ASCII string "CurveCPK" followed by a 16-byte compressed nonce.
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.	The 8-byte ASCII string "CurveCPV" followed by a 16-byte compressed nonce.
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.	The 16-byte ASCII string "CurveCP-server-M" 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.
The client's short-term secret key c' and short-term public key C'. These are also specific to this connection.	A 16-byte ASCII string followed by an 8-byte compressed nonce. The string is "CurveCP-client-H" for a Hello packet, "CurveCP-client-I" for an Initiate packet, or "CurveCP-client-M" for a Message packet. The compressed nonce represents a 64-bit integer in little-endian form. These integers are generated in increasing order.

Lessons Learnt: Cross-Protocol Interactions

matrix

"Matrix's encryption is based on the Double Ratchet Algorithm popularised by Signal"

Practically-exploitable Cryptographic Vulnerabilities in Matrix

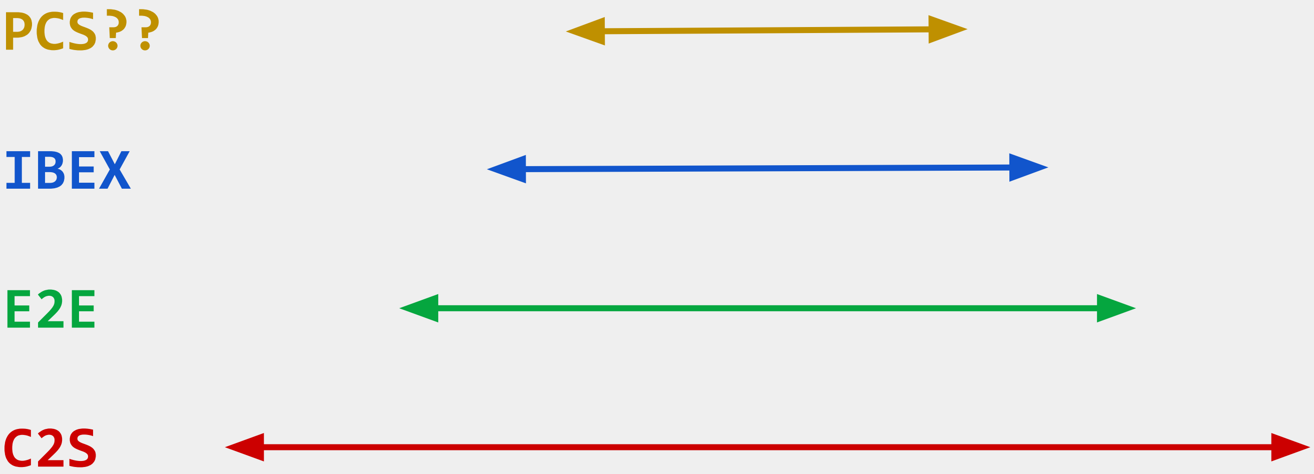
Martin R. Albrecht*, Sofía Celi†, Benjamin Dowling‡ and Daniel Jones§
* King's College London, martin.albrecht@kcl.ac.uk

Olm x Megolm

$\Downarrow$

Confidentiality break!

Lessons Learnt: Proactive Security



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