Analysis of the Threema Secure Messenger

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

What is Threema?

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.

*“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:

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Mercedes-Benz

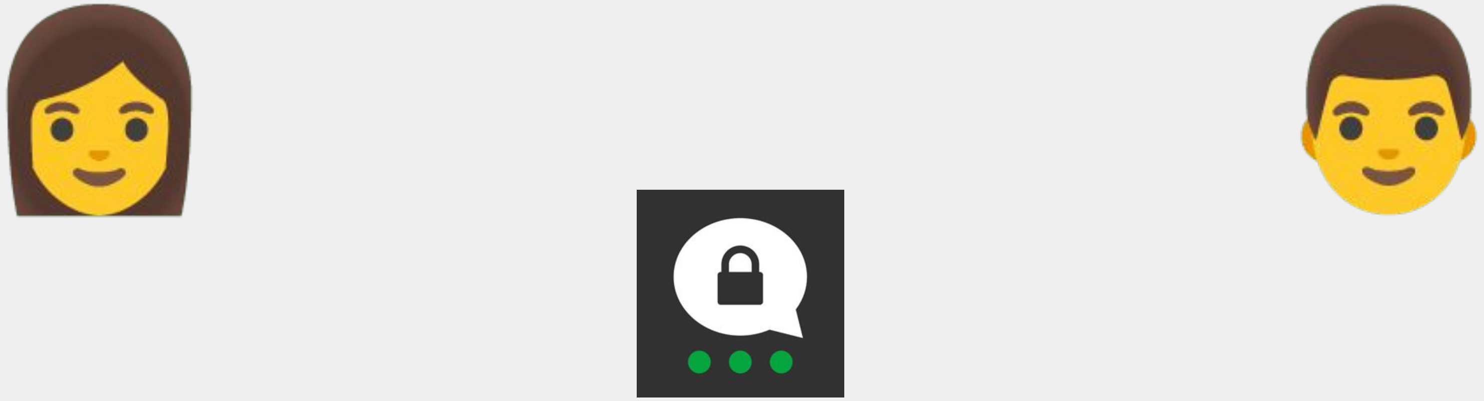
DEUTSCHE
BUNDESBANK
EUROSYSTEM

Emirates

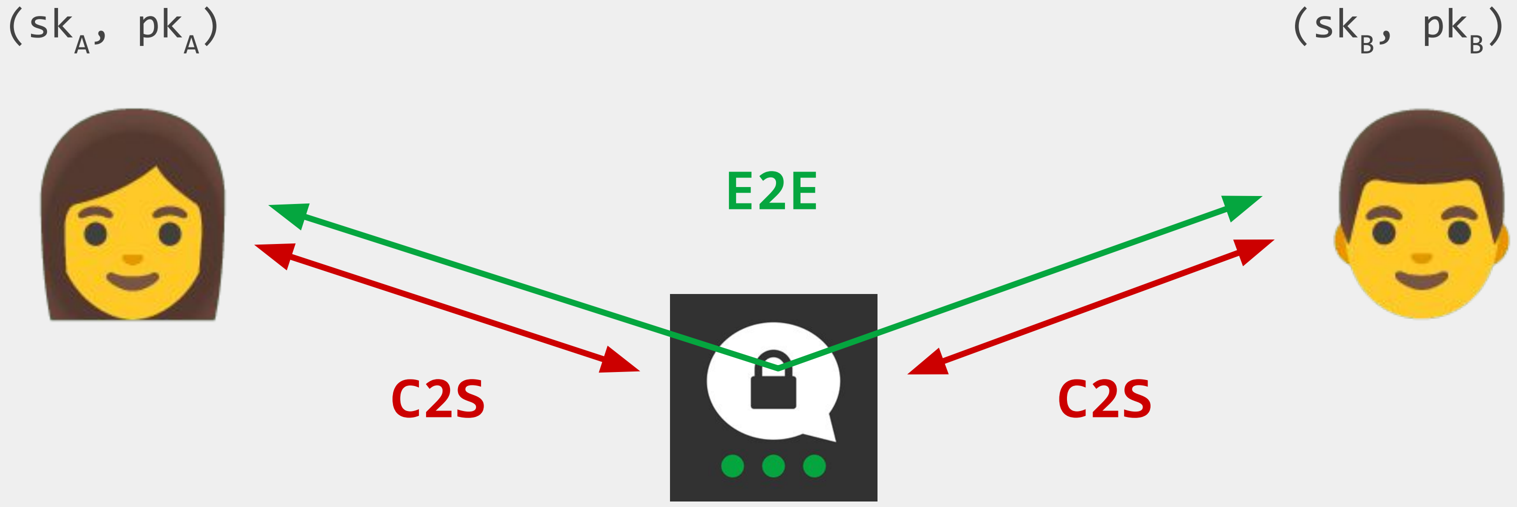
[1] https://threema.ch/en/about (Last checked 19 Mar 2023)

Bird’s Eye View of the Threema Protocol

Bird’s Eye View of the Threema Protocol



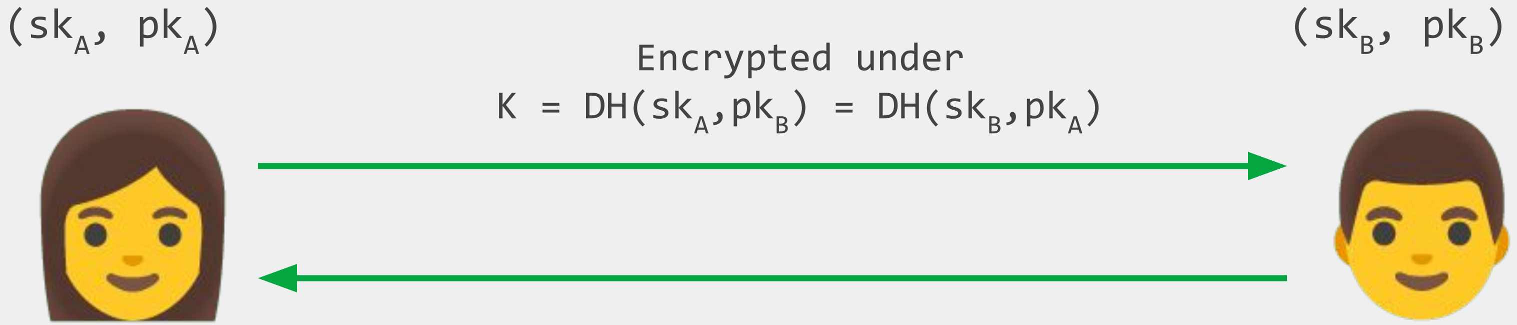
Bird's Eye View of the Threema Protocol



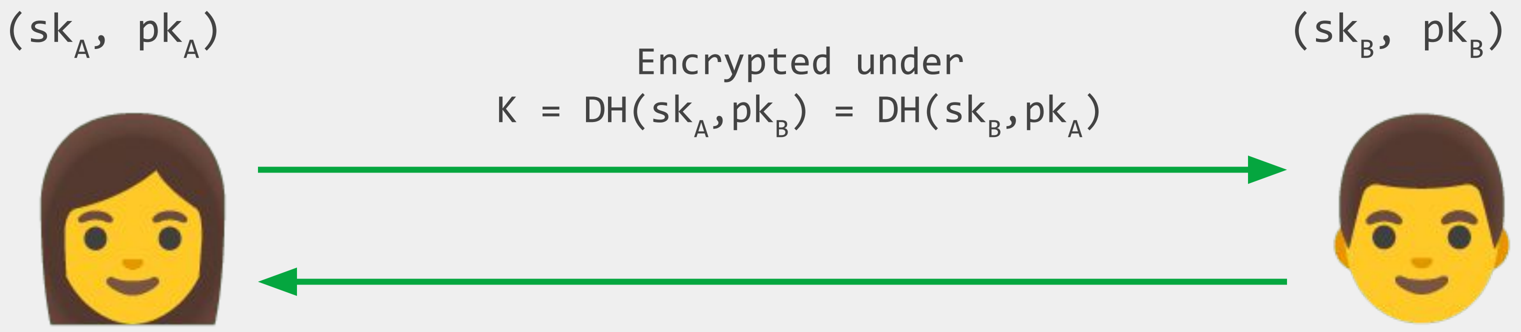
Two layers of encryption

Contents

E2E Protocol

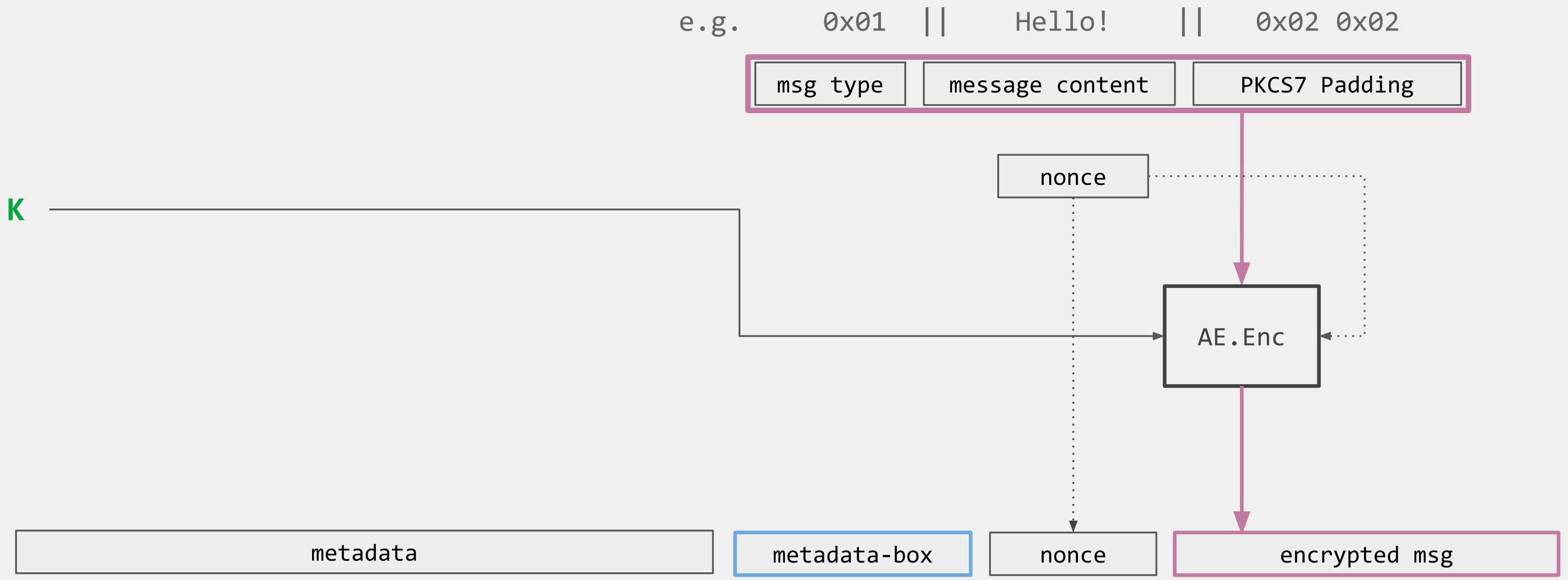


E2E Protocol

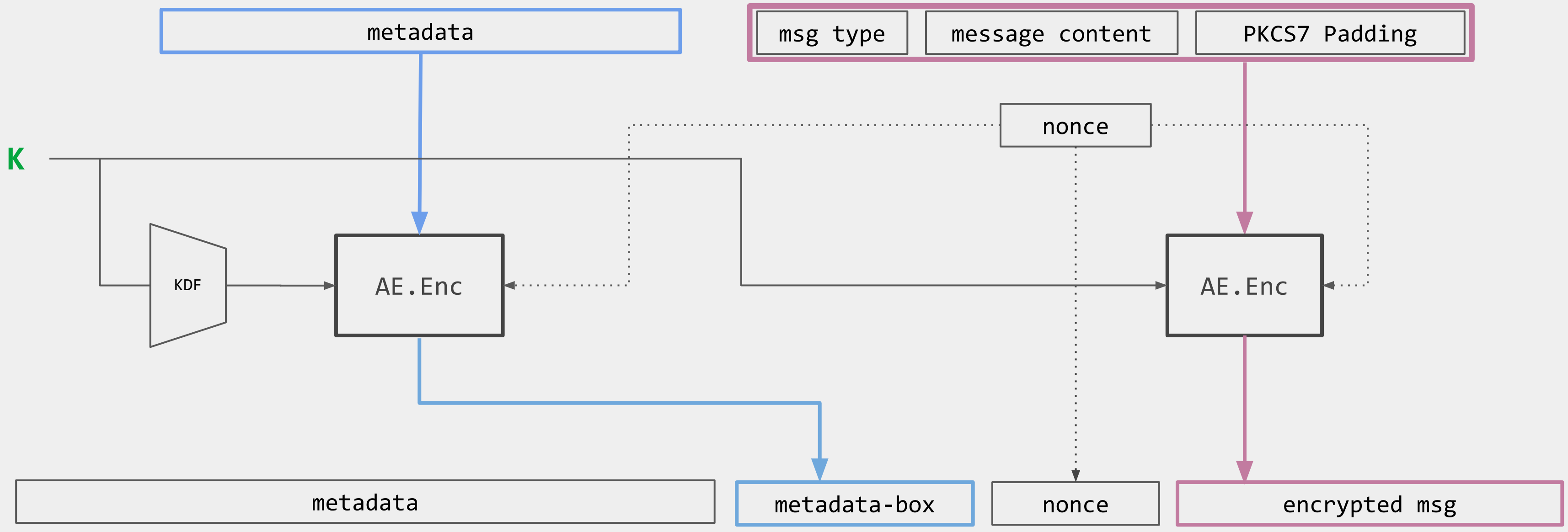


No Forward Secrecy!

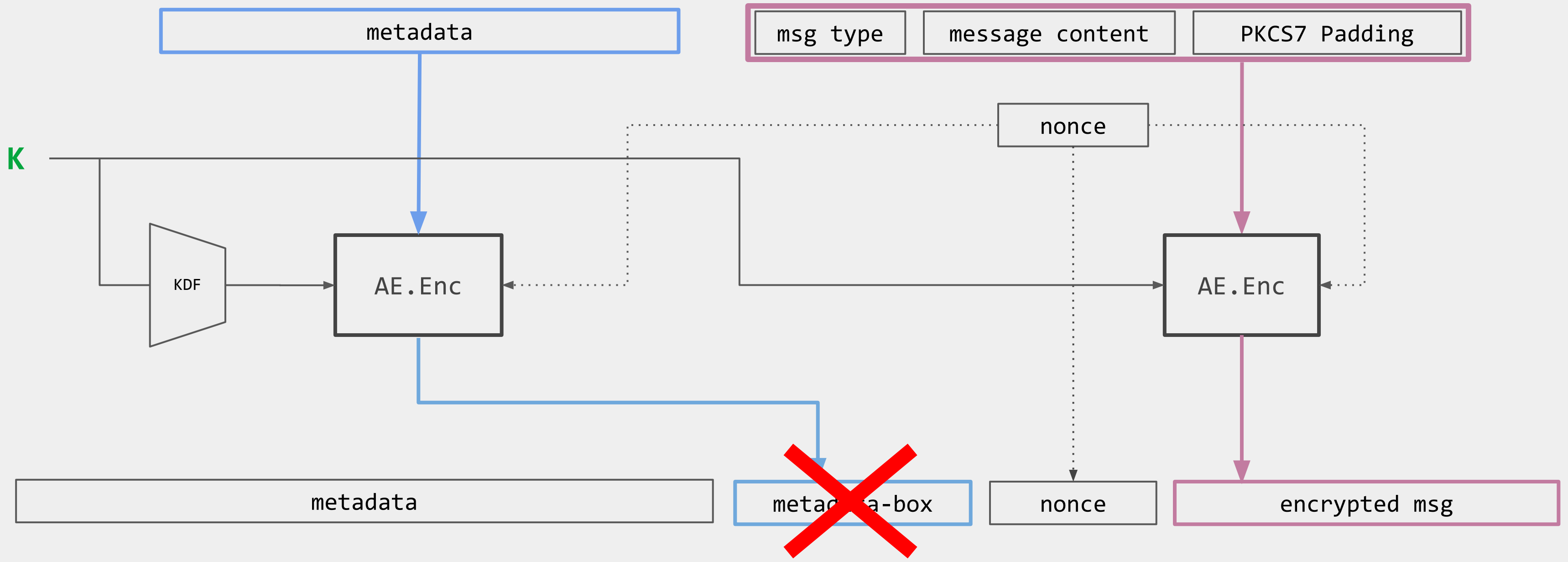
E2E Protocol: Message Structure



E2E Protocol: Message Structure

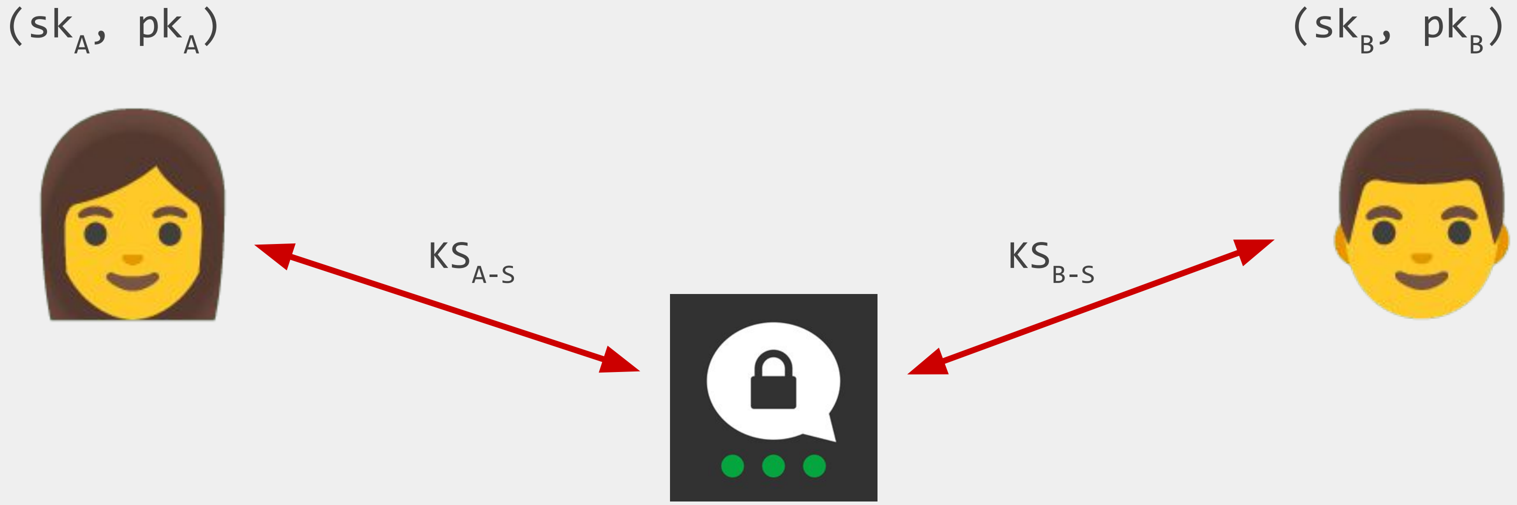


E2E Protocol: Message Structure



Contents

C2S Protocol



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

The C2S Protocol*

(sk_A, pk_A)

(sk_S, pk_S)

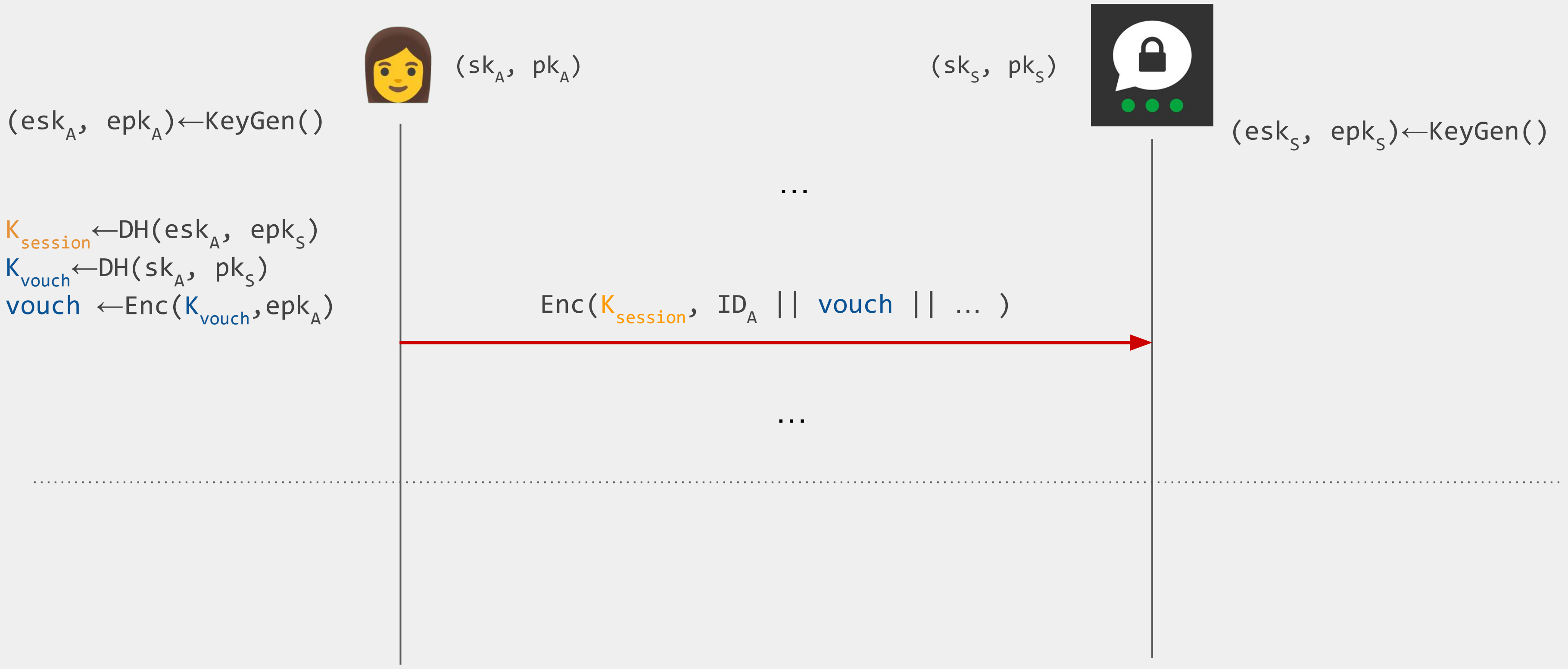
$(esk_A, epk_A) \leftarrow \text{KeyGen}()$

$(esk_S, epk_S) \leftarrow \text{KeyGen}()$

...

* Simplified, details omitted

The C2S Protocol*



$(sk_A,\ pk_A)$

$(sk_S,\ pk_S)$

$(esk_A,\ epk_A) \leftarrow \mathrm{KeyGen}()$

$(esk_S,\ epk_S) \leftarrow \mathrm{KeyGen}()$

...

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

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

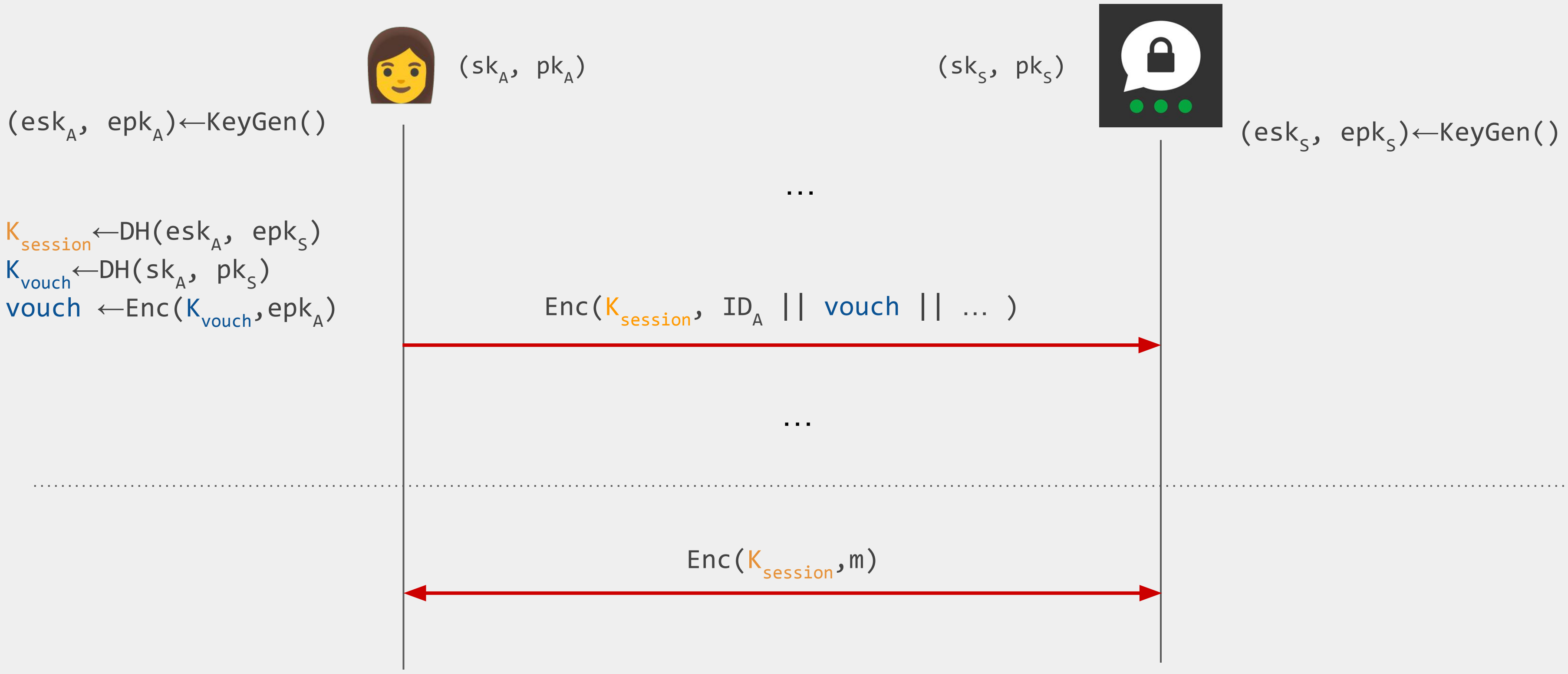
$vouch \leftarrow Enc(K_{vouch}, epk_A)$

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

...

* Simplified, details omitted

The C2S Protocol*


(sk_A, pk_A)
(sk_S, pk_S)
$(esk_A, epk_A) \leftarrow KeyGen()$
$(esk_S, epk_S) \leftarrow KeyGen()$
…
$K_{session} \leftarrow DH(esk_A, epk_S)$
$K_{vouch} \leftarrow DH(sk_A, pk_S)$
$vouch \leftarrow Enc(K_{vouch}, epk_A)$
$Enc(K_{session}, ID_A \mid\mid vouch \mid\mid \ldots)$
…
$Enc(K_{session}, m)$

* Simplified, details omitted

The C2S Protocol: Vouch Box

$$K_{vouch} \longleftarrow \mathrm{DH}(sk_A,\ pk_S)$$

$$vouch \longleftarrow \mathrm{Enc}(K_{vouch}, epk_A)$$

The C2S Protocol: Vouch Box

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

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

The C2S Protocol: Vouch Box

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

$vouch \leftarrow Enc(K_{vouch}, epk_A)$ Enc(some value)

The C2S Protocol: Vouch Box

$$K_{vouch} \leftarrow \mathrm{DH}(sk_A,\ pk_S)$$ DH(long-term, long-term)

$$vouch \leftarrow \mathrm{Enc}(K_{vouch}, epk_A)$$ Enc(some value)

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

The C2S Protocol: Vouch Box

$$K_{vouch} \leftarrow \mathrm{DH}(sk_A,\ pk_S)$$ DH(long-term, long-term)

$$vouch \leftarrow \mathrm{Enc}(K_{vouch}, epk_A)$$ Enc(some value)

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

```
epk = 0x01 || σ || 0x01
```

The C2S Protocol: Vouch Box

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

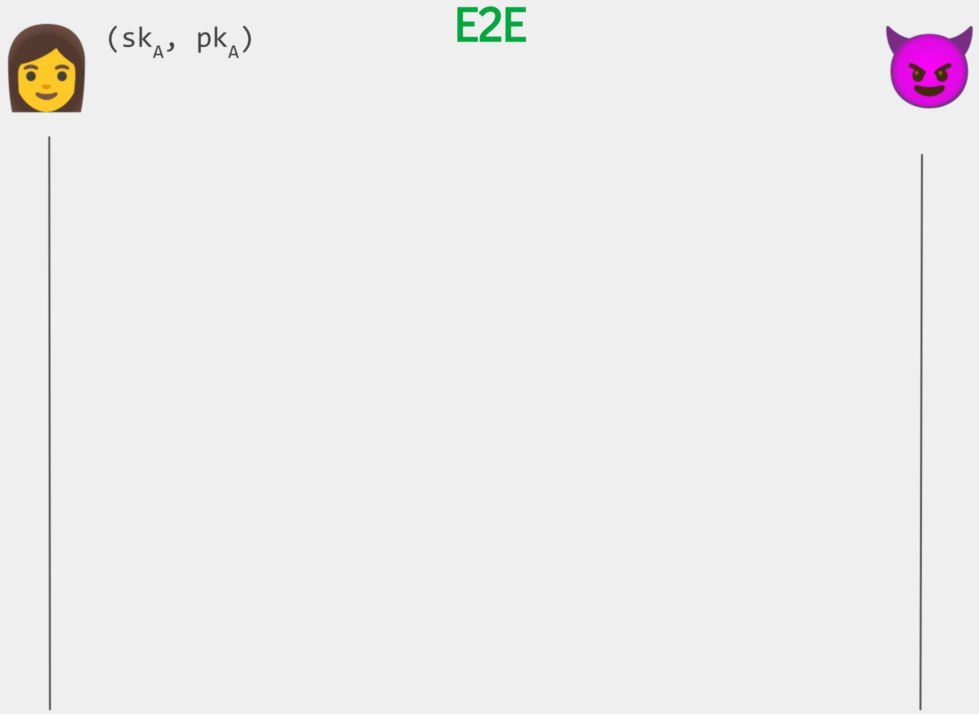
$$vouch \leftarrow Enc(K_{vouch}, epk_A)$$ Enc(some value)

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

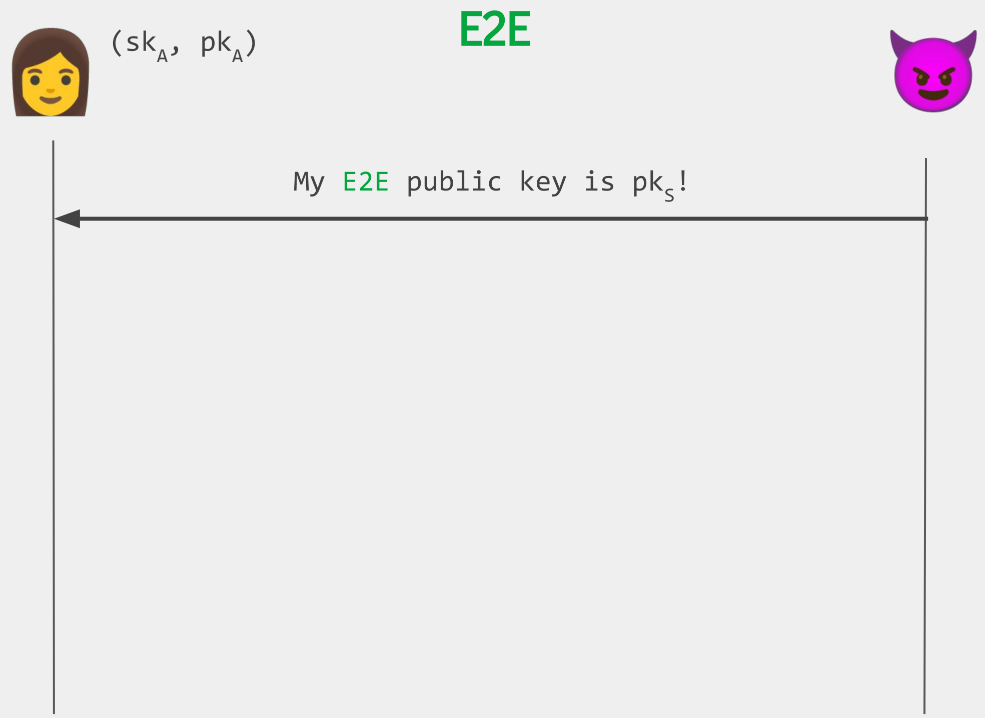
$$epk = 0x01 \mathbin{||} \boxed{\sigma} \mathbin{||} 0x01$$

UTF-8 valid string of 30B

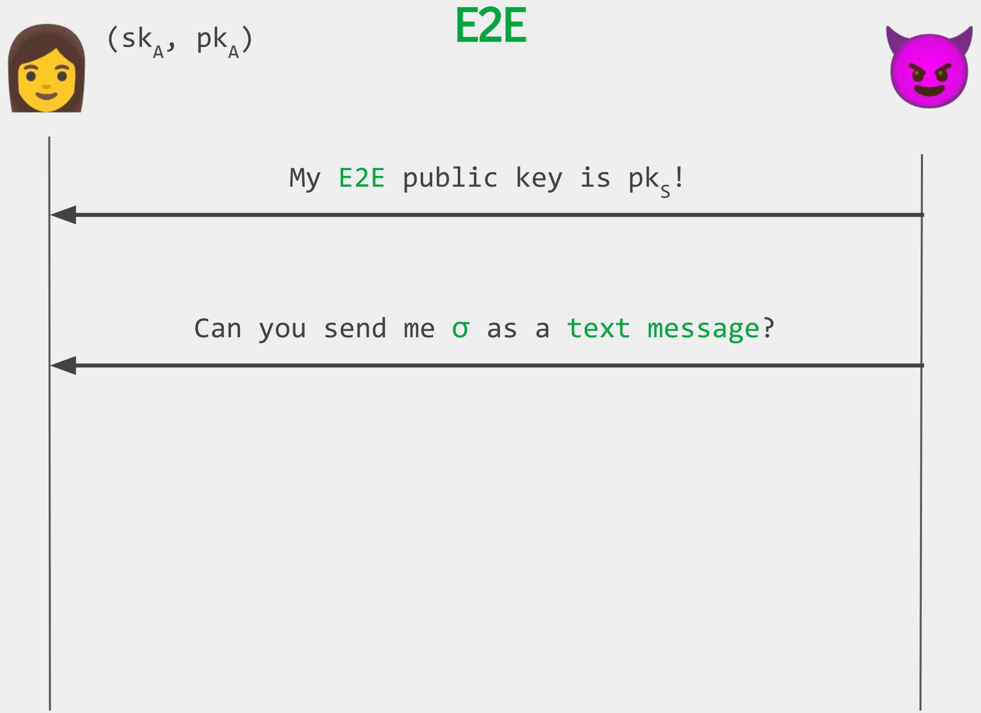
Attacking the C2S Protocol



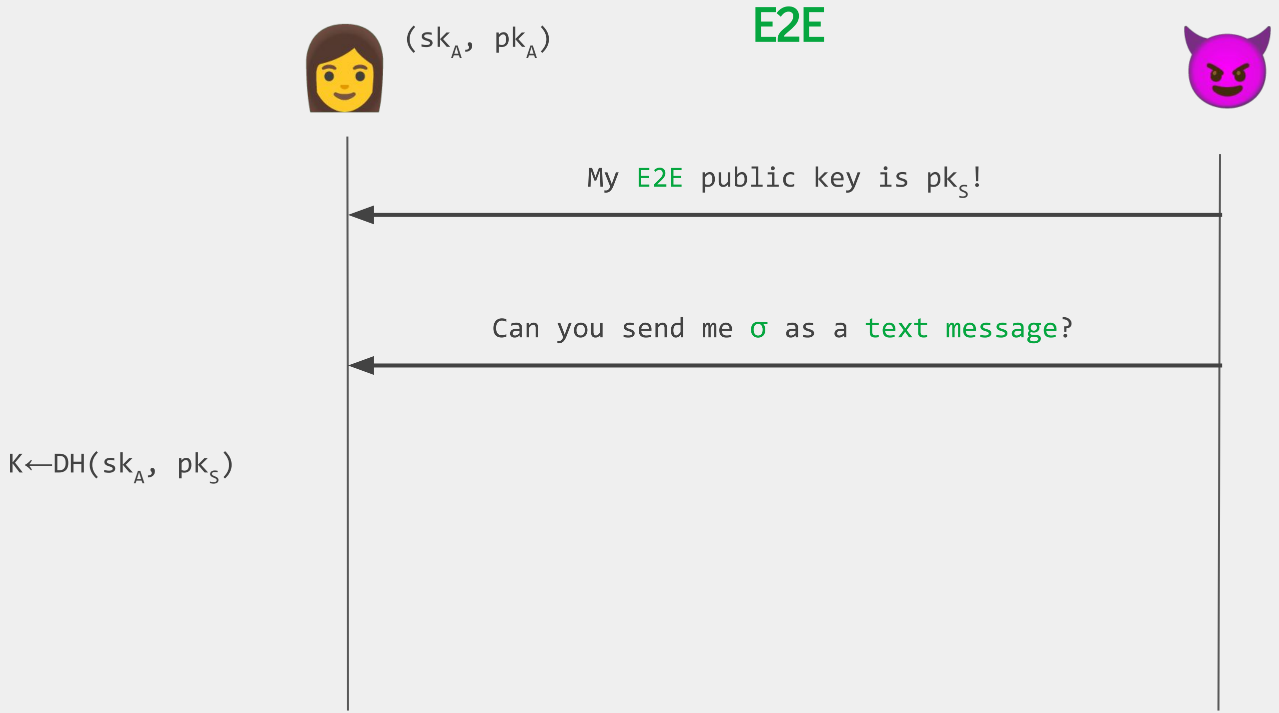
Attacking the C2S Protocol



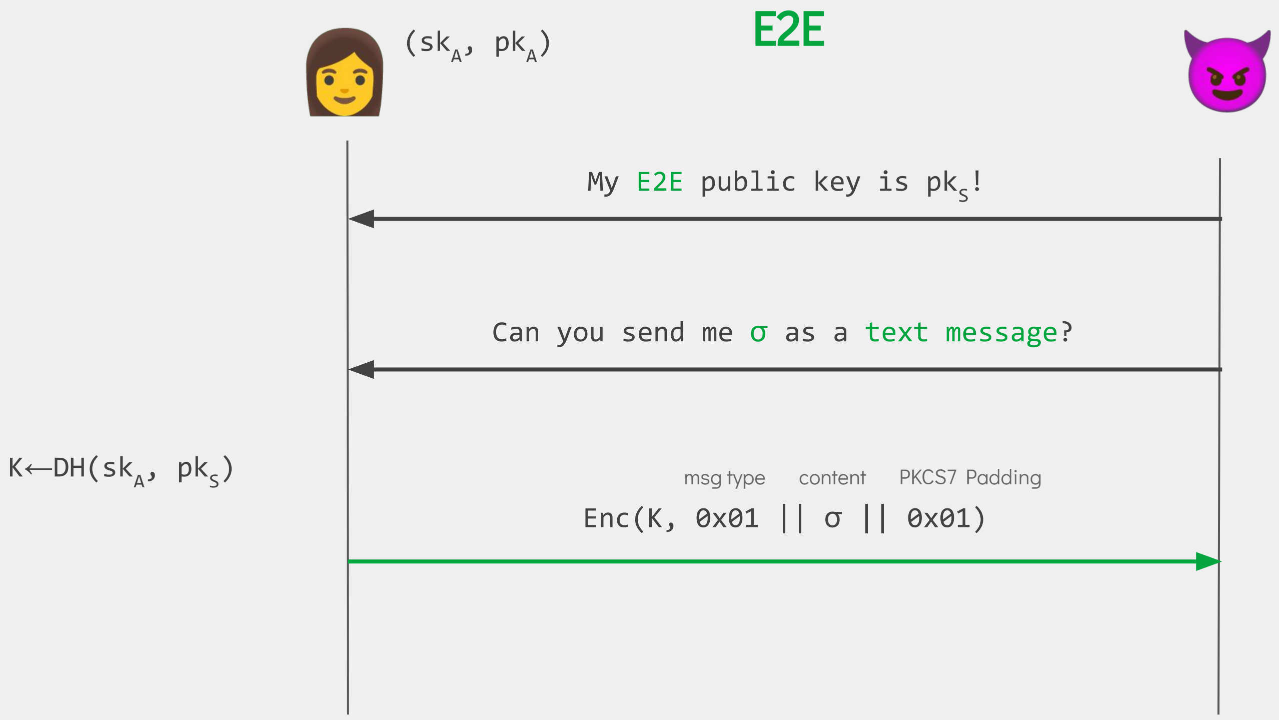
Attacking the C2S Protocol


(sk_A, pk_A)
E2E
My E2E public key is pk_S!
Can you send me σ as a text message?

Attacking the C2S Protocol

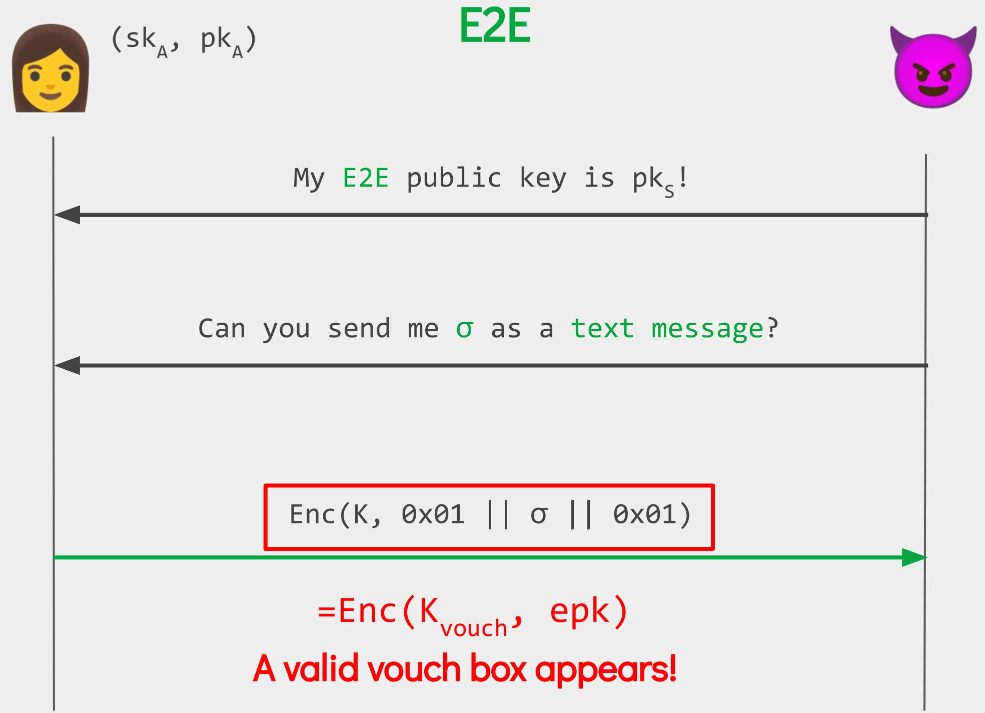

(sk_A, pk_A)
E2E
My E2E public key is pk_S!
Can you send me σ as a text message?
$K \leftarrow DH(sk_A, pk_S)$

Attacking the C2S Protocol



(sk_A, pk_A)
E2E
My E2E public key is pk_S!
Can you send me σ as a text message?
$K \leftarrow DH(sk_A, pk_S)$
msg type
content
PKCS7 Padding
Enc(K, 0x01 || σ || 0x01)

Attacking the C2S Protocol


(sk_A, pk_A)
E2E
My E2E public key is pk_S!
Can you send me σ as a text message?
$K \leftarrow DH(sk_A, pk_S)$
$=K_{vouch}$ in C2S
$Enc(K, 0x01 || σ || 0x01)$
$=Enc(K_{vouch}, epk)$
A valid vouch box appears!

Vouch Box Forgery

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

Vouch Box Forgery

- **C2S x E2E** cross-protocol attack:
- Sending a text message…

  compromises client

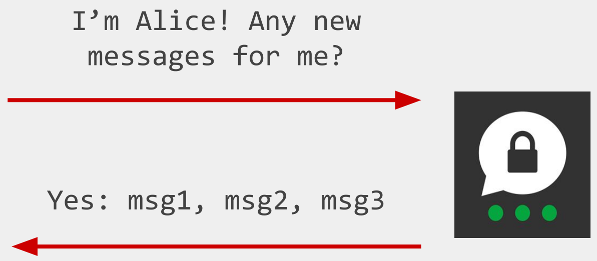
  authentication **forever**!

Vouch Box Forgery

- **C2S x E2E** cross-protocol attack:
- Sending a text message…

  compromises client

  authentication **forever**!

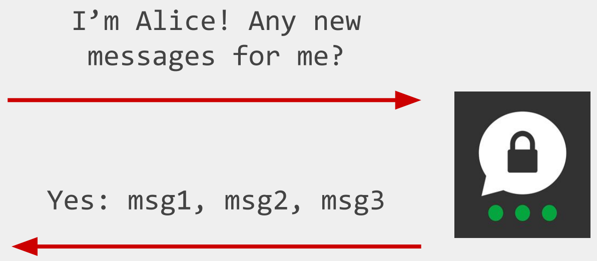


Vouch Box Forgery

- **C2S x E2E** cross-protocol attack:
- Sending a text message...

  compromises client

  authentication **forever**!



Attack: Vouch Box Forgery

Loose Ends

Two issues to still discuss:

- Find a suitable ephemeral key epk (AKA **Getting That Key**)
- Claim the server’s public key as ours (AKA **The Bamboozling**)

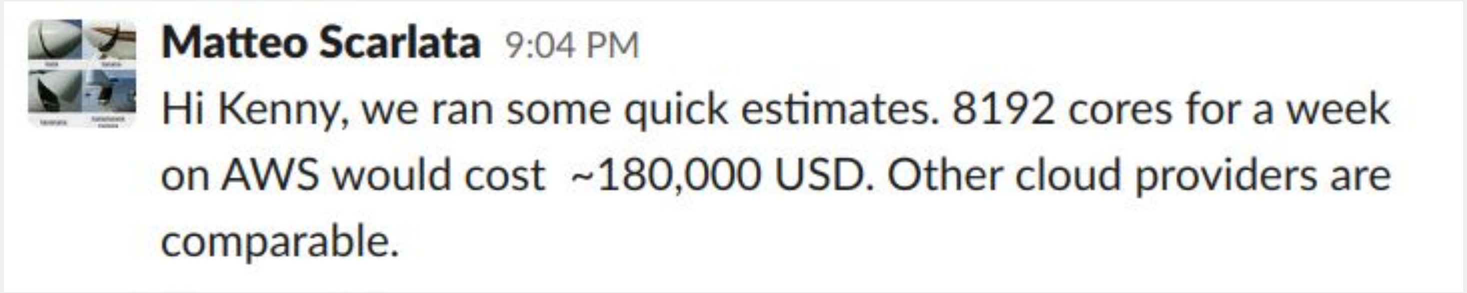
Part 1: Getting That Key

$$\texttt{epk} = \texttt{0x01} \,||\, \boxed{\sigma} \,||\, \texttt{0x01}$$

UTF-8 valid string of 30B

Requires sampling 2^{51} keys!

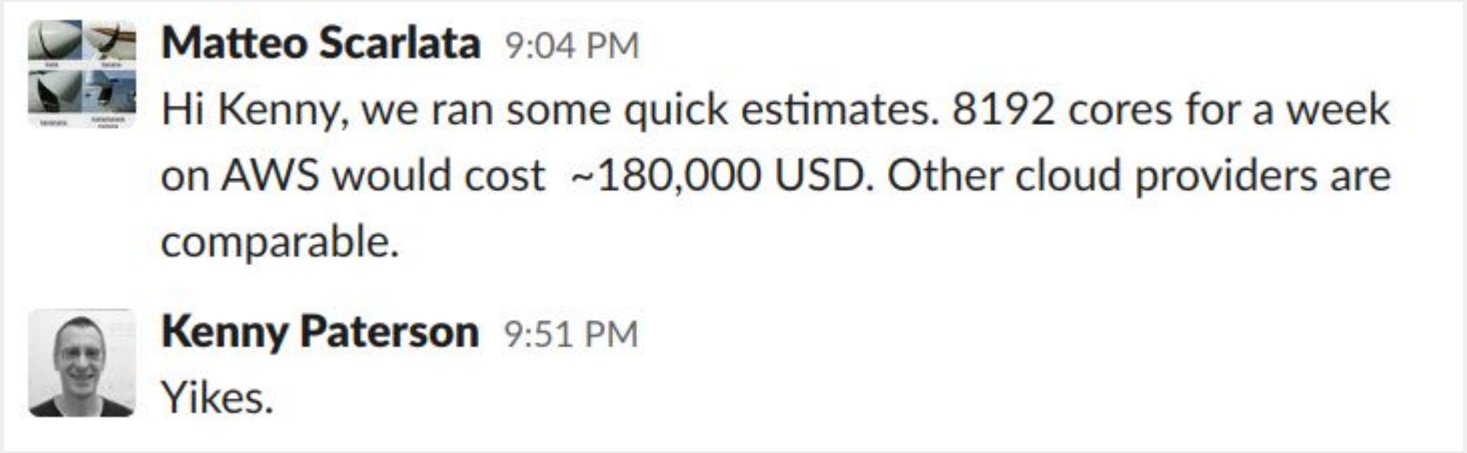
Part 1: Getting That Key



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.

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.

Kenny Paterson 9:51 PM

Yikes.

Part 1: Getting That Key

Some optimizations and 8100 core-days later…

```
esk = 504ac13e000000000003000336d612d322d3232313231392d30332d3030323000

epk  = 0175396a36df93276a6ae0a496d4bb5edf8331d79b573a2dcc813bdca152410 1
```

u9j6□’jjखh^ɔ1□W:-;ƶRA

Part 2: The Bamboozling

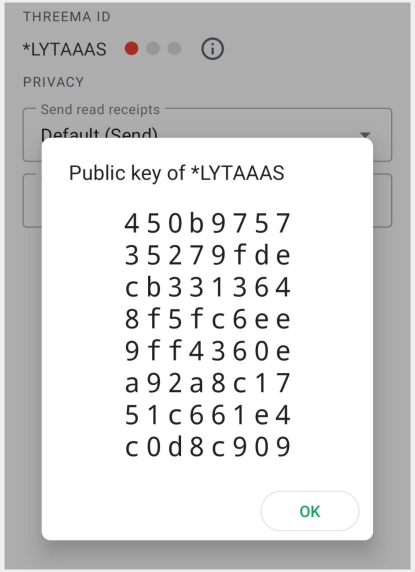
Part 2: The Bamboozling

- 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**


THREEMA ID

*LYTAAAS

PRIVACY

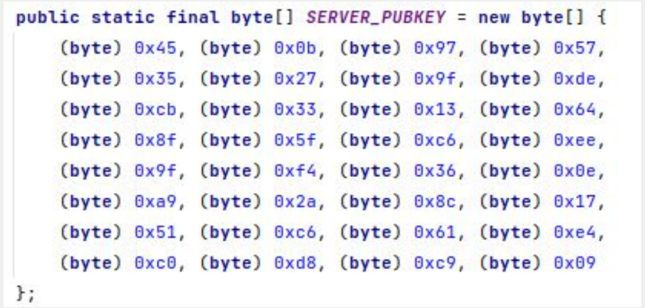
Send read receipts

Default (Send)

Public key of *LYTAAAS

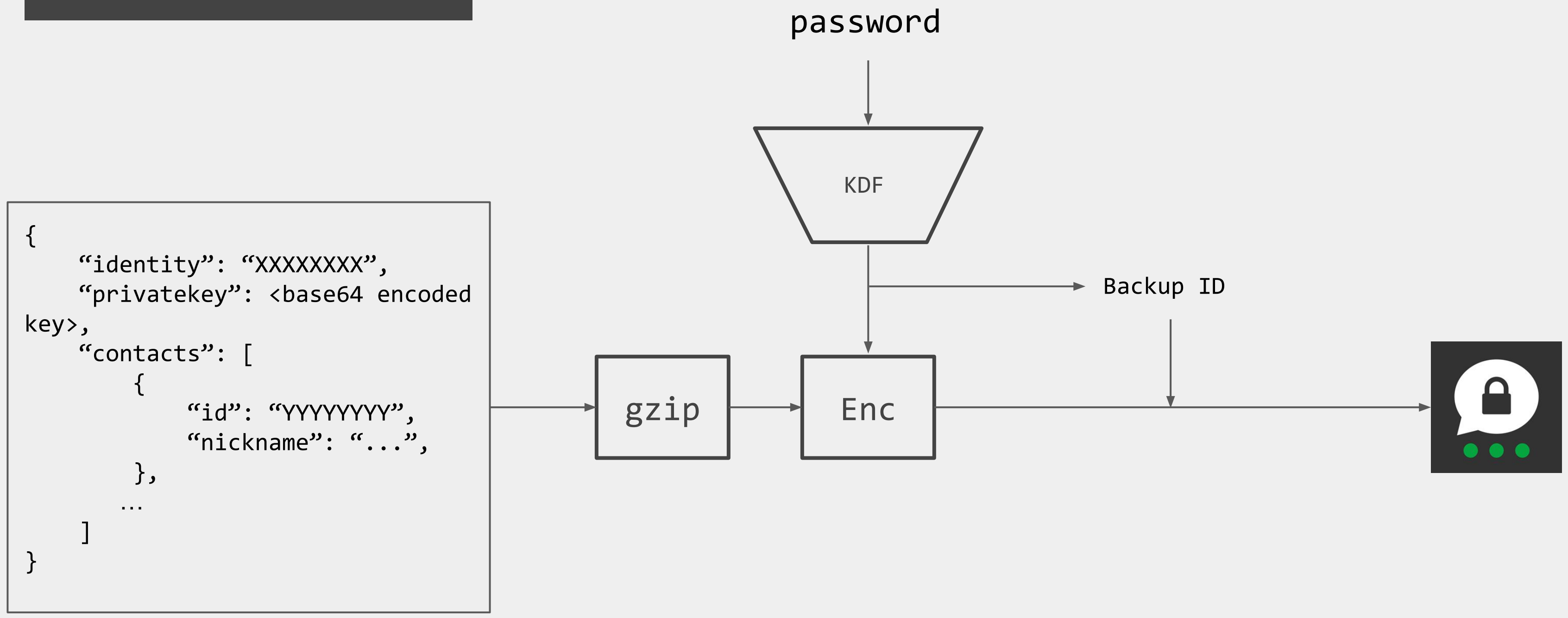
```
450b9757
35279fde
cb331364
8f5fc6ee
9ff4360e
a92a8c17
51c661e4
c0d8c909
```

OK


```
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
};
```

Contents

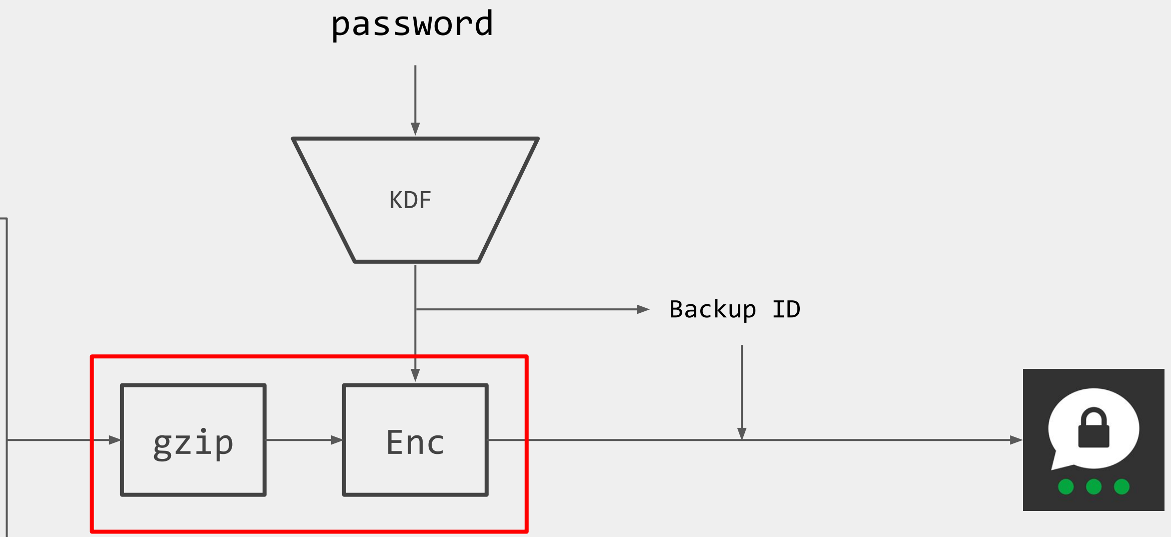
Threema Safe



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

Threema Safe

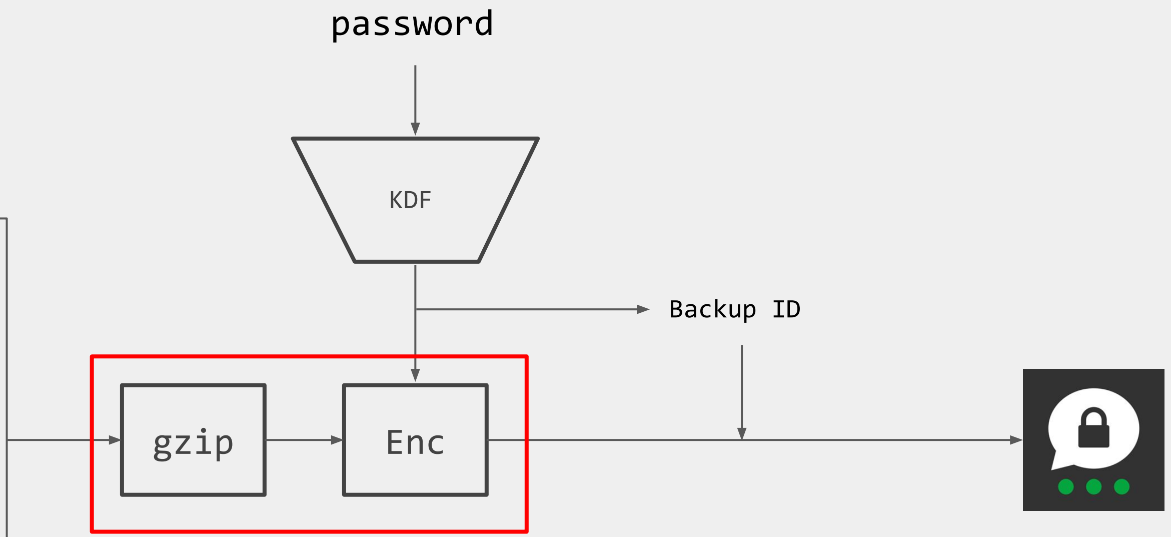
```
{
    "identity": "XXXXXXXX",
    "privatekey": <base64 encoded
key>,
    "contacts": [
        {
            "id": "YYYYYYYY",
            "nickname": "...",
        },
        ...
    ]
}
```



Compress then Encrypt?

Threema Safe

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

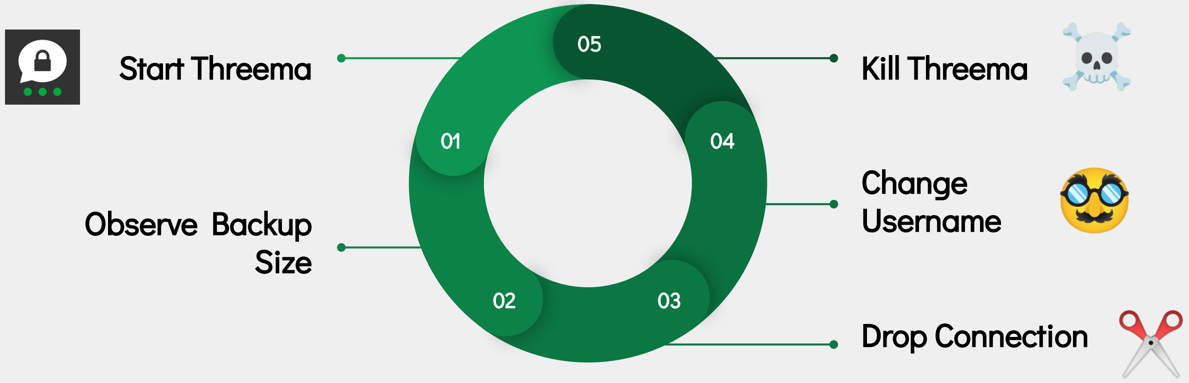


Compress then Encrypt?

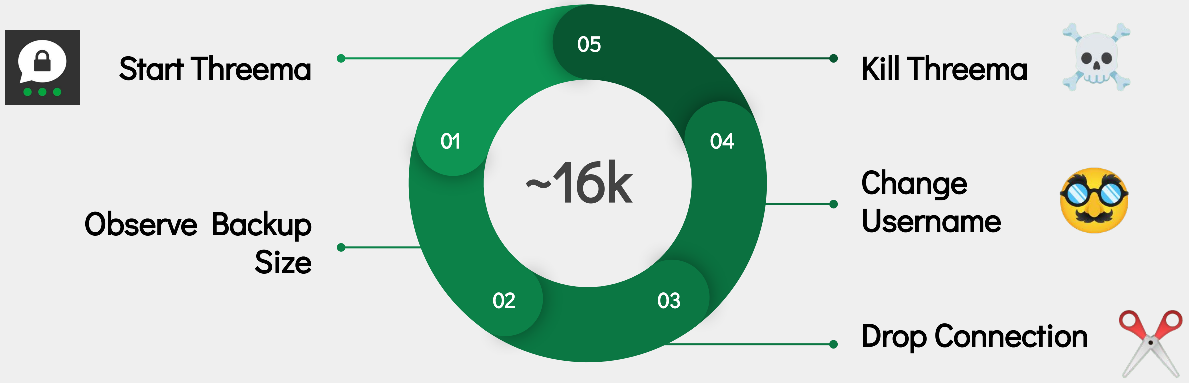
Attacker has partial control of backup content

Private Key Extraction: Feasibility

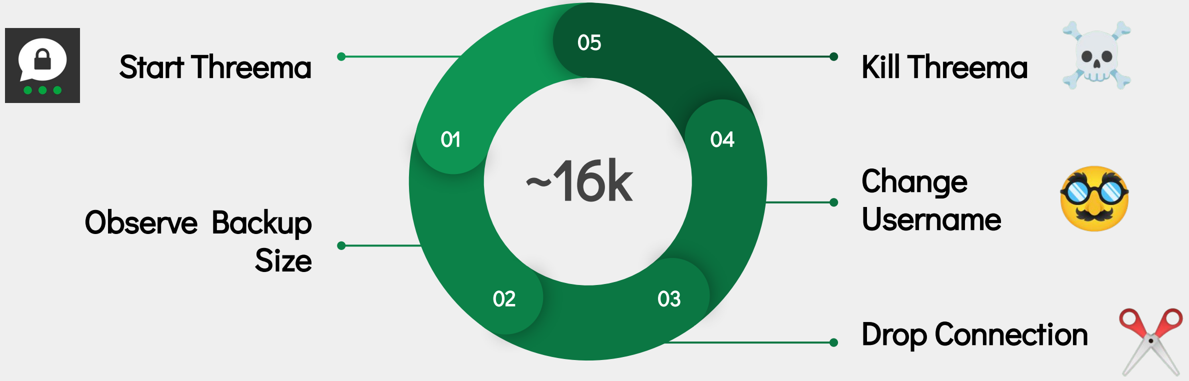
Private Key Extraction: Feasibility



Private Key Extraction: Feasibility



Private Key Extraction: Feasibility



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

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

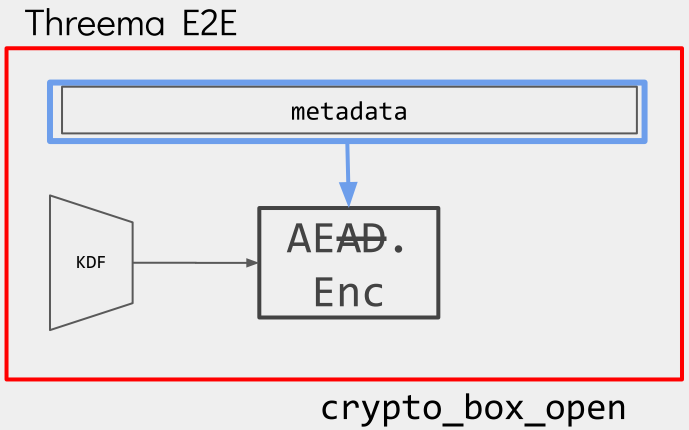
Lessons Learnt: Rolling your Protocol

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

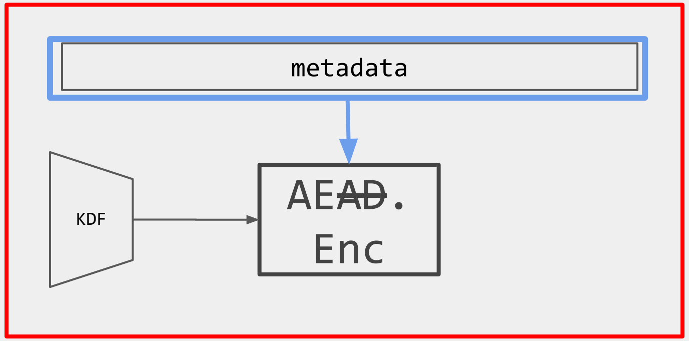
*[Threema has] a client-server protocol modelled after CurveCP, an end-to-end encryption protocol based on the **NaCl library** [...]*



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** [...]*

Threema E2E



crypto_box_open

Threema C2S

$K_1 \leftarrow \texttt{X25519}(x, Y)$

$K_2 \leftarrow \texttt{X25519}(a, S)$

$\texttt{vouch} \leftarrow \texttt{AE.Enc}(K_2, X)$

crypto_box_open

Lessons Learnt: Rolling your Protocol

Lessons Learnt: Rolling your Protocol

Is the Bridgefy App safe to use?

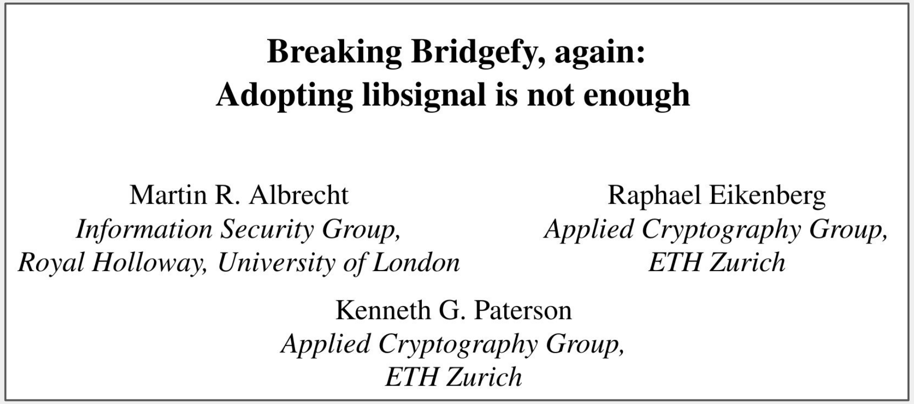
bridgefy

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 [...]*



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

Lessons Learnt: Cross-Protocol Interactions

Lessons Learnt: Cross-Protocol Interactions

Threema:

E2E x C2S

$\Downarrow$

Permanent
authentication break

Lessons Learnt: Cross-Protocol Interactions

Threema:

E2E x C2S

$\Downarrow$

Permanent authentication break

Threema:

E2E x Reg

$\Downarrow$

Kompromat

Lessons Learnt: Cross-Protocol Interactions

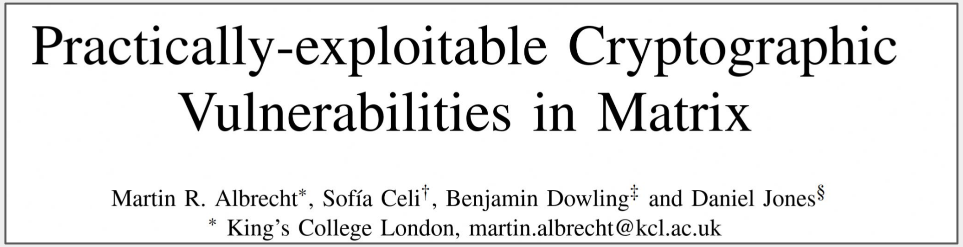
Lessons Learnt: Cross-Protocol Interactions

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

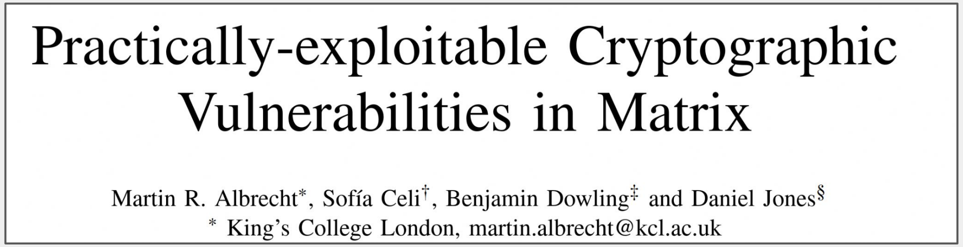
Martin R. Albrecht*, Sofía Celi†, Benjamin Dowling‡ and Daniel Jones§
* King's College London, martin.albrecht@kcl.ac.uk

Lessons Learnt: Cross-Protocol Interactions

Matrix's encryption is based on the

Double Ratchet Algorithm popularised by Signal

matrix


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

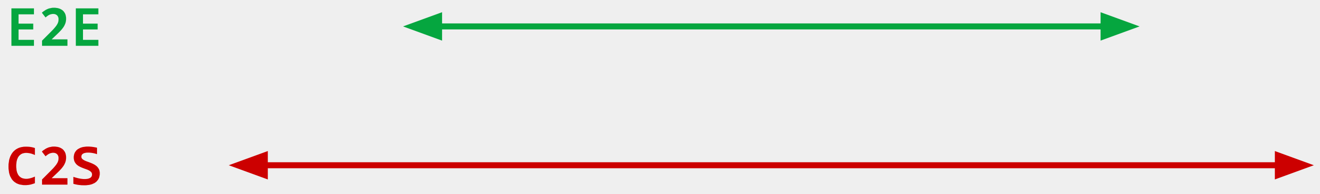
Olm x Megolm

$\Downarrow$

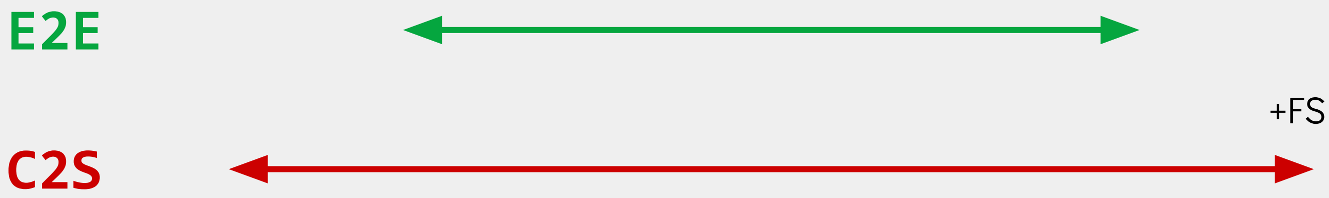
Confidentiality break!

Lessons Learnt: Proactive Security

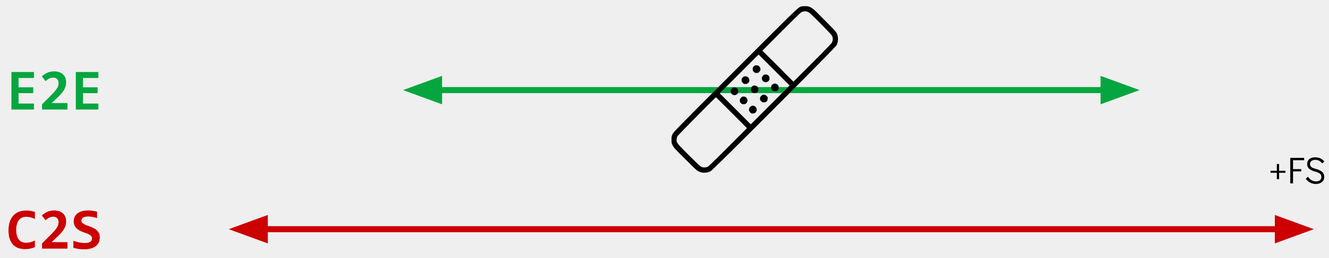
Lessons Learnt: Proactive Security



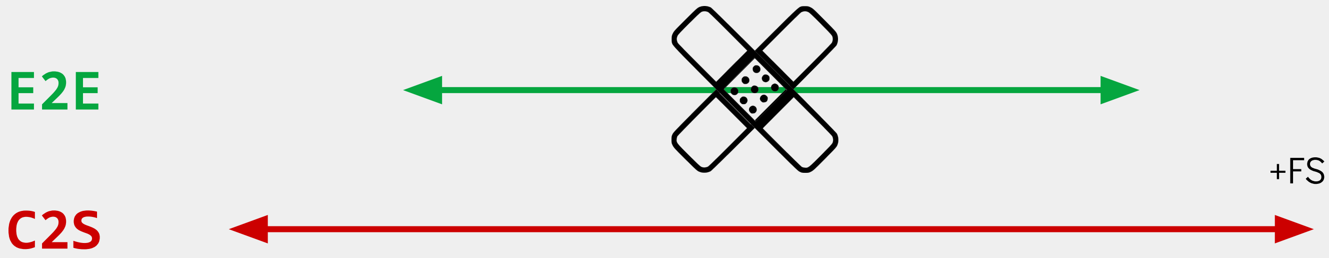
Lessons Learnt: Proactive Security



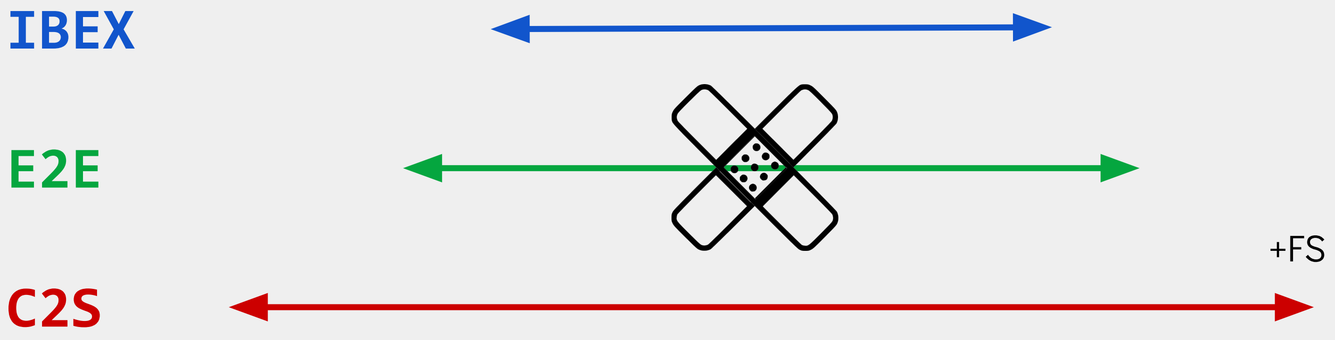
Lessons Learnt: Proactive Security



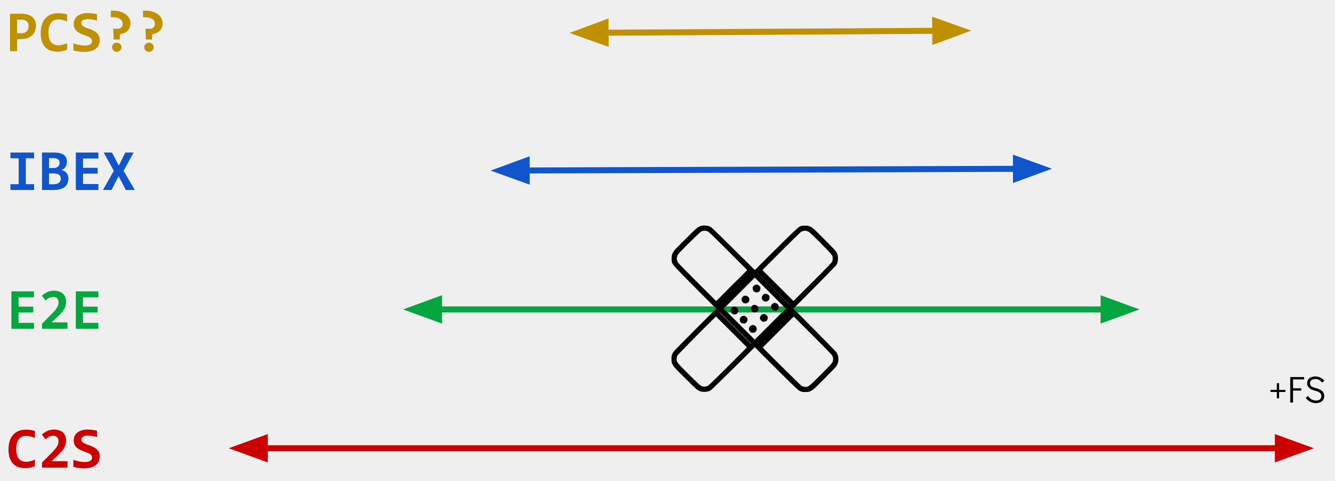
Lessons Learnt: Proactive Security



Lessons Learnt: Proactive Security



Lessons Learnt: Proactive Security



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
- $\boxed{\text{But if you do:}}$ 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
- $\boxed{\text{But if you do:}}$ 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

External/Network Attacker

Compromised Threema Server

Attack: Message Reordering/Omission

Attack: Message Replay/Reflection

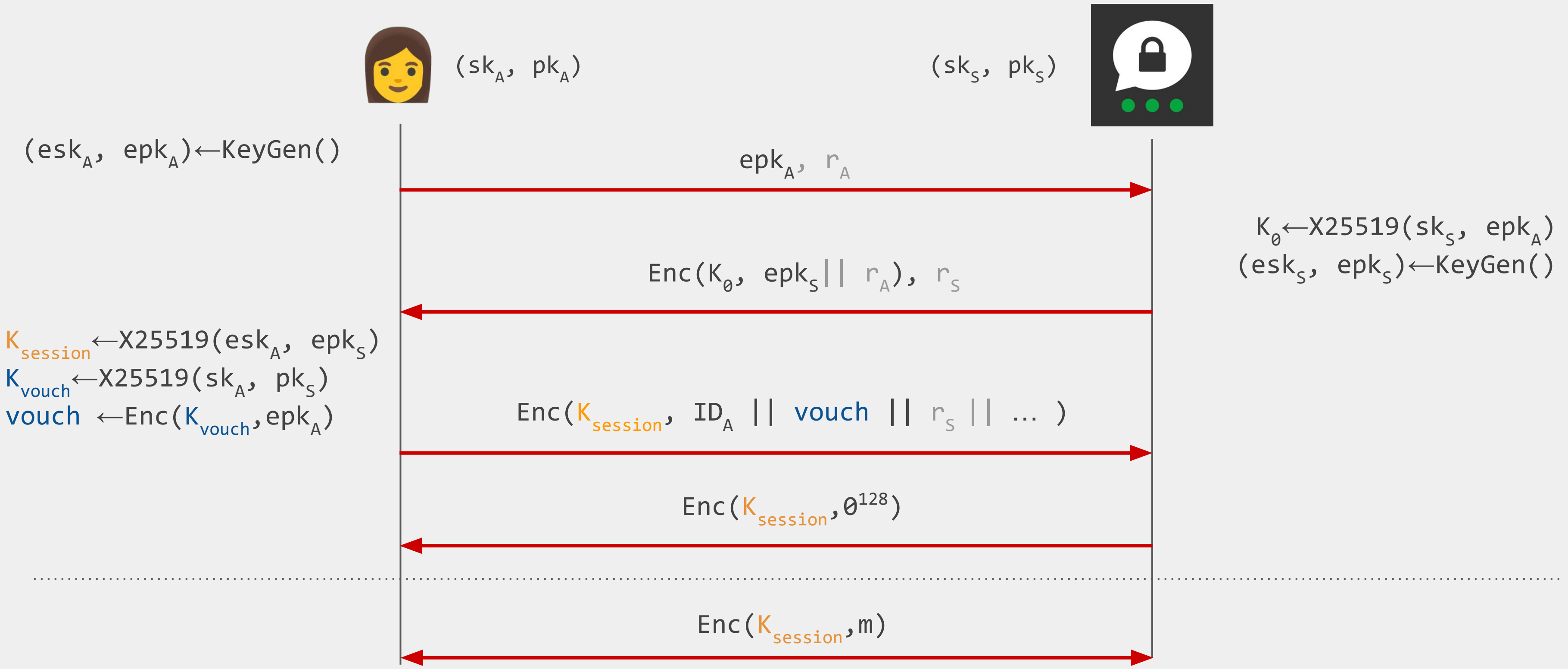
Attack: Kompromat

Attack: Compression-Side Channel on Threema Safe

Attack: Threema ID Export

Physical Device Access
("Compelled Access")

The C2S Protocol*

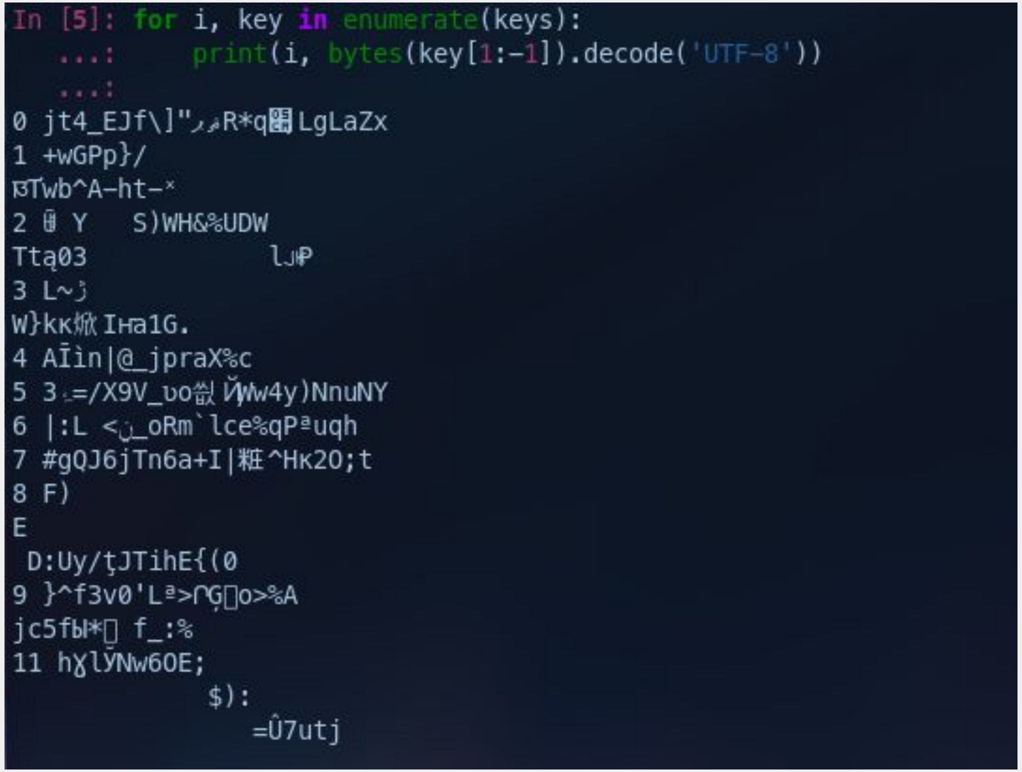

(sk_A, pk_A)
(sk_S, pk_S)
$(esk_A, epk_A) \leftarrow \mathrm{KeyGen}()$
epk_A, r_A
$K_0 \leftarrow \mathrm{X25519}(sk_S, epk_A)$
$(esk_S, epk_S) \leftarrow \mathrm{KeyGen}()$
$\mathrm{Enc}(K_0, epk_S \| r_A), r_S$
$K_{session} \leftarrow \mathrm{X25519}(esk_A, epk_S)$
$K_{vouch} \leftarrow \mathrm{X25519}(sk_A, pk_S)$
$vouch \leftarrow \mathrm{Enc}(K_{vouch}, epk_A)$
$\mathrm{Enc}(K_{session}, ID_A \| vouch \| r_S \| \ldots)$
$\mathrm{Enc}(K_{session}, 0^{128})$
$\mathrm{Enc}(K_{session}, m)$

* Simplified, details omitted

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



```
In [5]: for i, key in enumerate(keys):
   ...:     print(i, bytes(key[1:-1]).decode('UTF-8'))
   ...:
0 jt4_EJf\]"ݲݳR*q🆘LgLaZx
1 +wGPp}/
ꞵTwb^A-ht-ˣ
2 Ꙫ Y   S)WH&%UDW
Ttą03            ľ⅃₽
3 L~ڗ
W}kκ焮Iнa1G.
4 AĪìn|@_jpraX%c
5 3ؠ=/X9V_ʋo㽈ЙꙀw4y)NnuNY
6 |:L <ڽ_oRm`lce%qPªuqh
7 #gQJ6jTn6a+I|粧^Hκ2O;t
8 F)
E
 D:Uy/ţJTihE{(0
9 }^f3v0'Lª>ՐĢ▯o>%A
jc5fЫ*▯ f_:%
11 hɣlȲNw6OE;
             $):
                =Û7utj
```

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

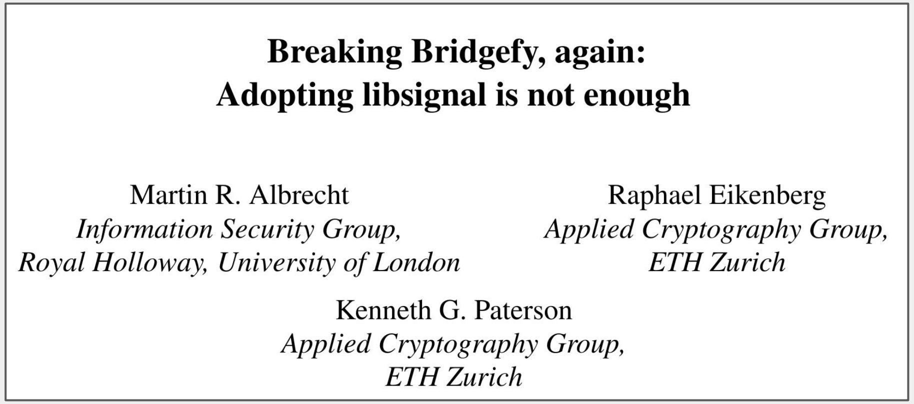
Assumes access to the device:

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 [...]*



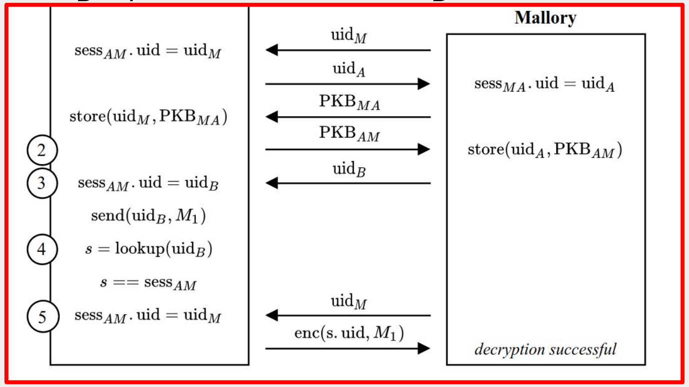
Breaking Bridgefy, again:
Adopting libsignal is not enough

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



Mallory
$\text{sess}_{AM}.\text{uid} = \text{uid}_M$
uid_M
uid_A
$\text{sess}_{MA}.\text{uid} = \text{uid}_A$
$\text{store}(\text{uid}_M, \text{PKB}_{MA})$
PKB_{MA}
PKB_{AM}
$\text{store}(\text{uid}_A, \text{PKB}_{AM})$
2
3
$\text{sess}_{AM}.\text{uid} = \text{uid}_B$
uid_B
$\text{send}(\text{uid}_B, M_1)$
4
$s = \text{lookup}(\text{uid}_B)$
$s == \text{sess}_{AM}$
5
$\text{sess}_{AM}.\text{uid} = \text{uid}_M$
uid_M
$\text{enc}(s.\text{uid}, M_1)$
decryption successful

`SessionCipher.message_decrypt`