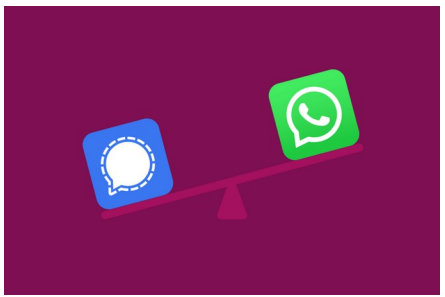
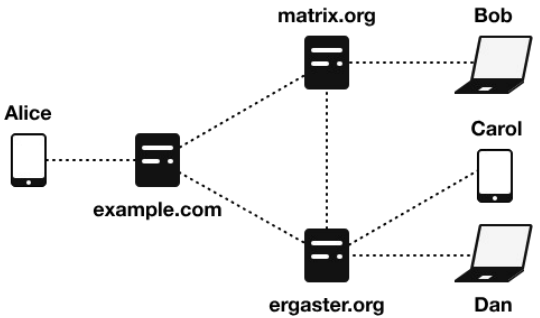
Hagen Echzell <hagene@uio.no>, 2025-11-26, TEK4500

End-to-end encryption in group messaging protocols



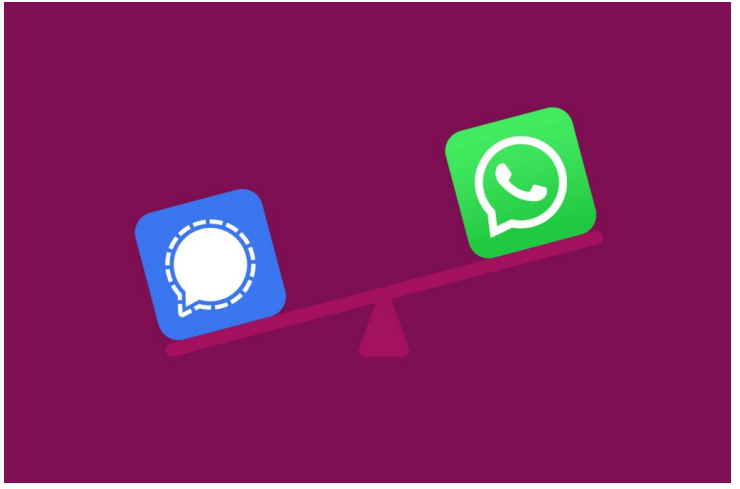




Left: https://www.expressvpn.com/blog/whatsapp-vs-signal/
Middle: https://matrix.org/docs/matrix-concepts/elements-of-matrix/
Right: CC-BY-SA 3.0 Teresa Such Ferrer https://commons.wikimedia.org/wiki/File:Ambystoma_mexicanum_-_Teresa_Such_Ferrer.jpg

Signal and Whatsapp use «The Double Ratchet Algorithm»

- End-to-end-encrypted (E2EE) messaging
- Forward secrecy (FS)
- Post-compromise security (PCS)



Post-Compromise Security: «Backward Secrecy»

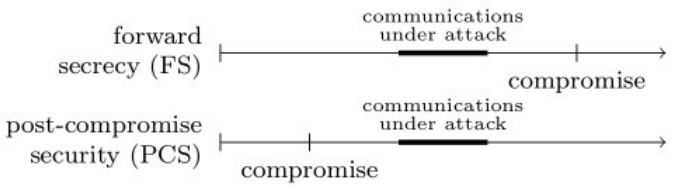


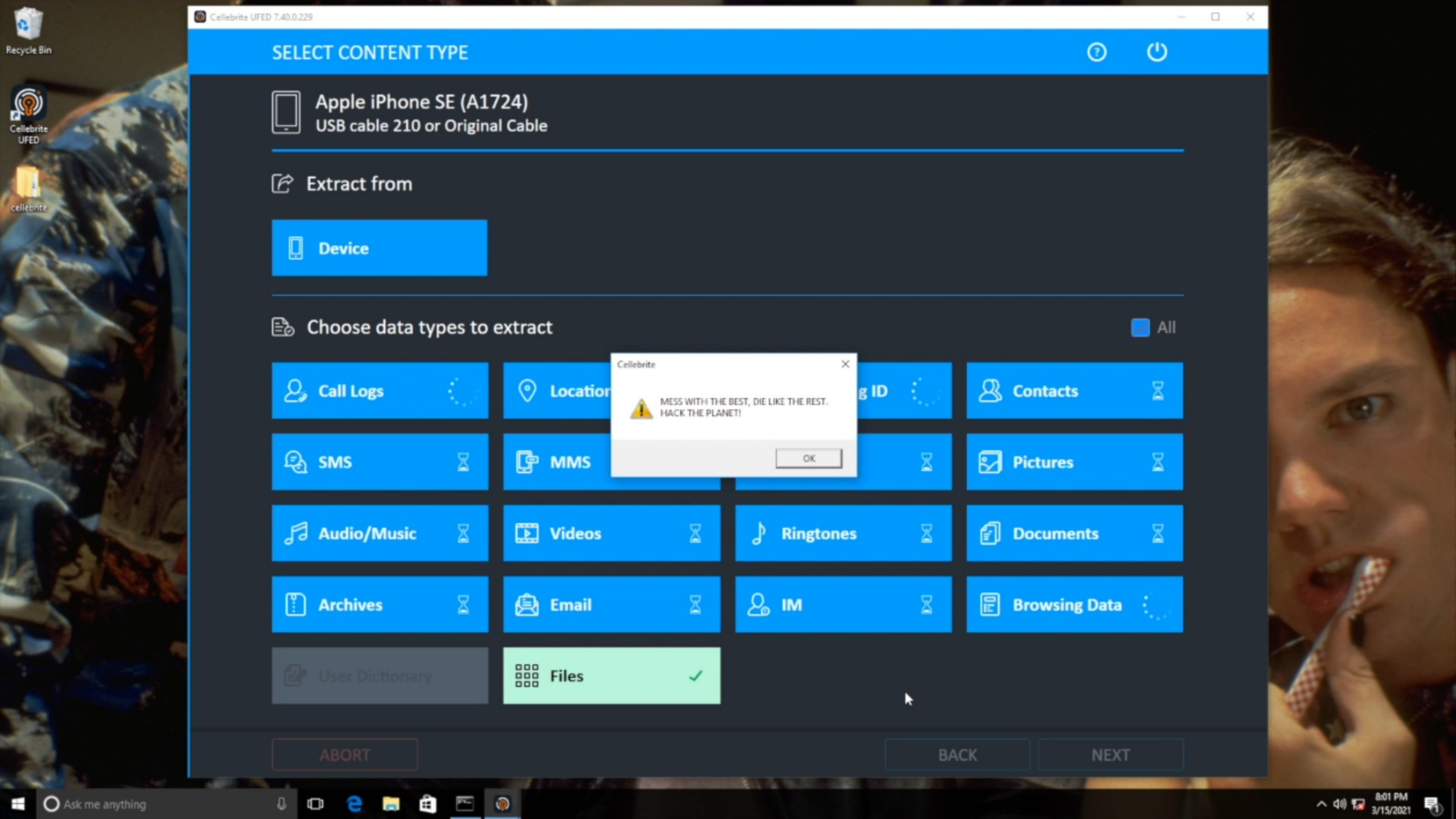
Figure 1: Attack scenarios of forward secrecy and PCS, with the communications under attack marked in **bold** and time from left to right. Forward secrecy protects against *later* compromise; PCS protects against *earlier* compromise.

https://eprint.iacr.org/2017/666.pdf

What’s the threat model for PCS?



https://signal.org/blog/cellebrite-vulnerabilities/


Recycle Bin
Cellebrite UFED
cellebrite
Cellebrite UFED 7.40.0.229
SELECT CONTENT TYPE
Apple iPhone SE (A1724)
USB cable 210 or Original Cable
Extract from
Device
Choose data types to extract
All
Call Logs
Location
g ID
Contacts
SMS
MMS
Pictures
Audio/Music
Videos
Ringtones
Documents
Archives
Email
IM
Browsing Data
User Dictionary
Files
Cellebrite
MESS WITH THE BEST, DIE LIKE THE REST.
HACK THE PLANET!
OK
ABORT
BACK
NEXT
Ask me anything
8:01 PM
3/15/2021

Agenda

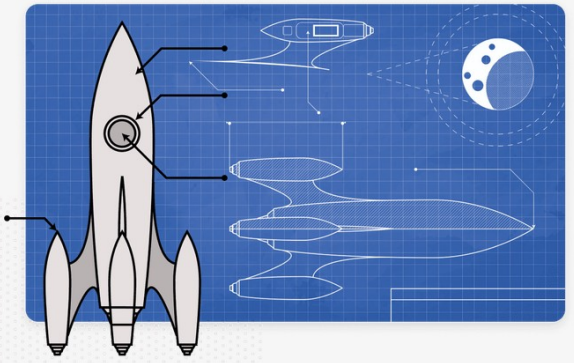
- Asynchronous E2EE 1:1-messaging **with FS and PCS**
  - Signal
  - A lot of ratchets
- E2EE group messaging
  - Signal/Whatsapp
  - Finding out about scaling issues
- **Asynchronous Ratcheting Trees**
  - «Messaging Layer Security» (MLS)
- Matrix
  - Decentralization
  - One more salamander

Some important things didn’t fit in today

- Post-quantum stuff
- Signatures/Authentication

Technical information

Specifications and software libraries for developers



Specifications

Encryption in messaging environments integrates many ideas which often need to be composed separately in different applications. We make an effort to break out ideas into independent specifications so that they can be integrated as appropriate for different projects.

XEdDSA and VXEdDSA

This document describes how to create and verify EdDSA-compatible signatures using public key and private key formats initially defined for the X25519 and X448 elliptic curve Diffie-Hellman functions. This document also describes "VXEdDSA" which extends XEdDSA to make it a verifiable random function, or VRF.

X3DH

This document describes the "X3DH" (or "Extended Triple Diffie-Hellman") key agreement protocol. X3DH establishes a shared secret key between two parties who mutually authenticate each other based on public keys. X3DH provides forward secrecy and cryptographic deniability.

PQXDH

This document describes the "PQXDH" (or "Post-Quantum Extended Diffie-Hellman") key agreement protocol. PQXDH establishes a shared secret key between two parties who mutually authenticate each other based on public keys. PQXDH provides post-quantum forward secrecy and a form of cryptographic deniability but still relies on the hardness of the discrete log problem for mutual authentication in this revision of the protocol.

Double Ratchet

This document describes the Double Ratchet algorithm, which is used by two parties to exchange encrypted messages based on a shared secret key. The parties derive new keys for every Double Ratchet message so that earlier keys cannot be calculated from later ones. The parties also send Diffie-Hellman public values attached to their messages. The results of Diffie-Hellman calculations are mixed into the derived keys so that later keys cannot be calculated from earlier ones. These properties give some protection to earlier or later encrypted messages in case of a compromise of a party's keys.

Sesame

This document describes the Sesame algorithm for managing message encryption sessions in an asynchronous and multi-device setting.

ML-KEM Braid

This document describes the ML-KEM Braid protocol. ML-KEM Braid is a sparse continuous key agreement protocol that uses ML-KEM to allow two parties to agree on a shared secret key in a way that provides post-quantum forward secrecy and post-compromise security.



https://signal.org/docs/

CC-BY-SA 3.0 Teresa Such Ferrer https://commons.wikimedia.org/wiki/File:Ambystoma_mexicanum_-_Teresa_Such_Ferrer.jpg

https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf

The Double Ratchet Algorithm

Trevor Perrin (editor) Moxie Marlinspike
Rolfe Schmidt (revision 3+)

Revision 4 , 2025-11-04

Contents

1. Introduction **3**

2. Overview **3**
2.1. KDF chains . . . 3
2.2. Symmetric-key ratchet . . . 5
2.3. Diffie-Hellman ratchet . . . 6
2.4. Double Ratchet . . . 13
2.6. Out-of-order messages . . . 17

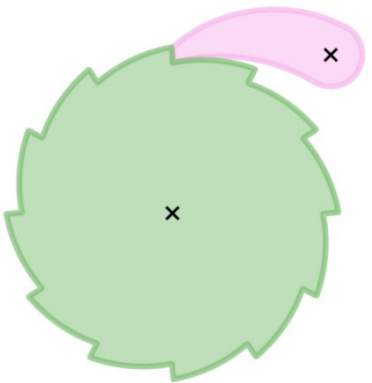
3. Double Ratchet **18**
3.1. External functions . . . 18
3.2. State variables . . . 19
3.3. Initialization . . . 19
3.4. Encrypting messages . . . 20
3.5. Decrypting messages . . . 20

4. Double Ratchet with header encryption **23**
4.1. Overview . . . 23
4.2. External functions . . . 27
4.3. State variables . . . 27
4.4. Initialization . . . 27
4.5. Encrypting messages . . . 28
4.6. Decrypting messages . . . 29

5. The Sparse Post-Quantum Ratchet **30**
5.1 Sparse Continuous Key Agreement . . . 31
5.2 External functions . . . 32
5.3 State variables . . . 32
5.4 Initialization . . . 32
5.5 Encrypting Messages . . . 33
5.6 Decrypting Messages . . . 34

What’s a ratchet?

- A tool that only spins in one direction

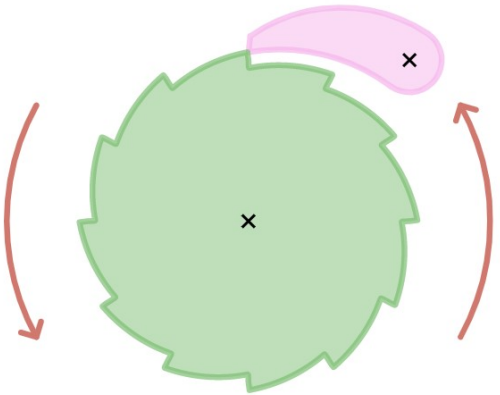




Anton Ehrola, CC BY-SA 4.0, https://commons.wikimedia.org/wiki/File:Ratchet1.jpg
Arglin Kampling CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=123530838

Signal's **Double Ratchet** combines KDF and DH

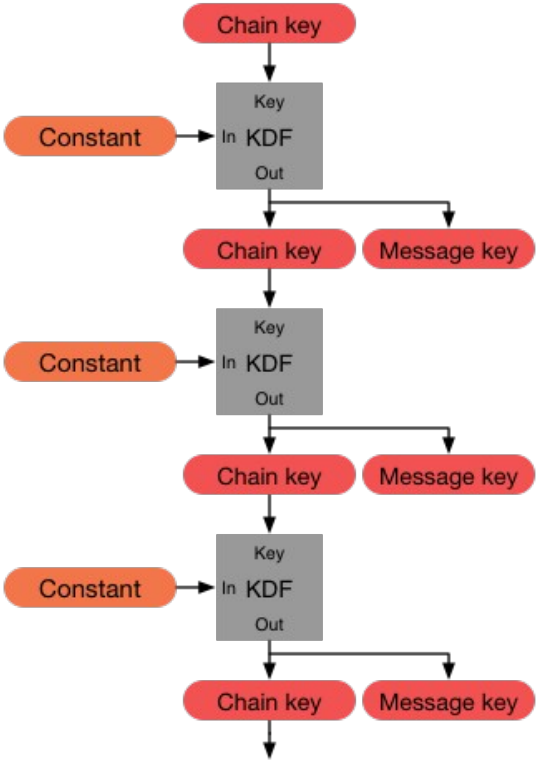
- Key Derivation Function (KDF)
  - KDF chain based ratchet
- Diffie-Hellman (DH) ratchet



- Signal protocol relies on a X3DH / PQXDH session key
  - «Extended Triple Diffie-Hellman»
  - Out of scope today!

Ratchet: Arglin Kampling CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=123530838

We can chain a Key Derivation Function (KDF)



- Resilience
  - Output keys appear random without the input
- Forward security
  - Our first ratchet!
- But no Post-Compromise Security

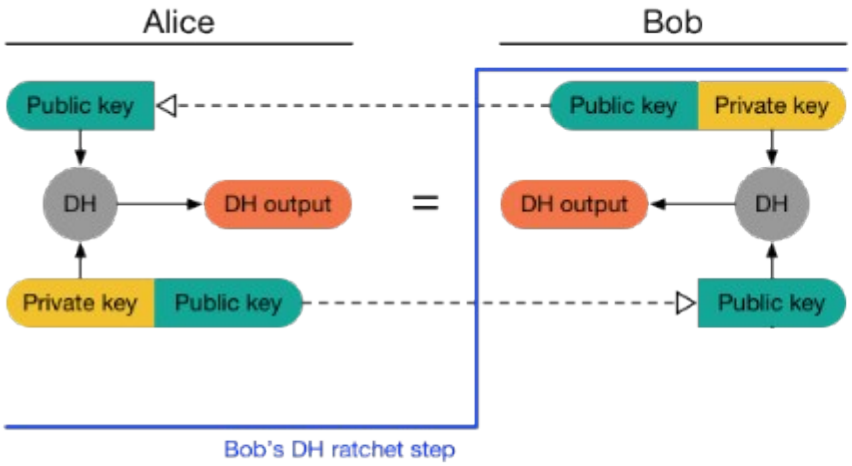
What’s a good KDF?

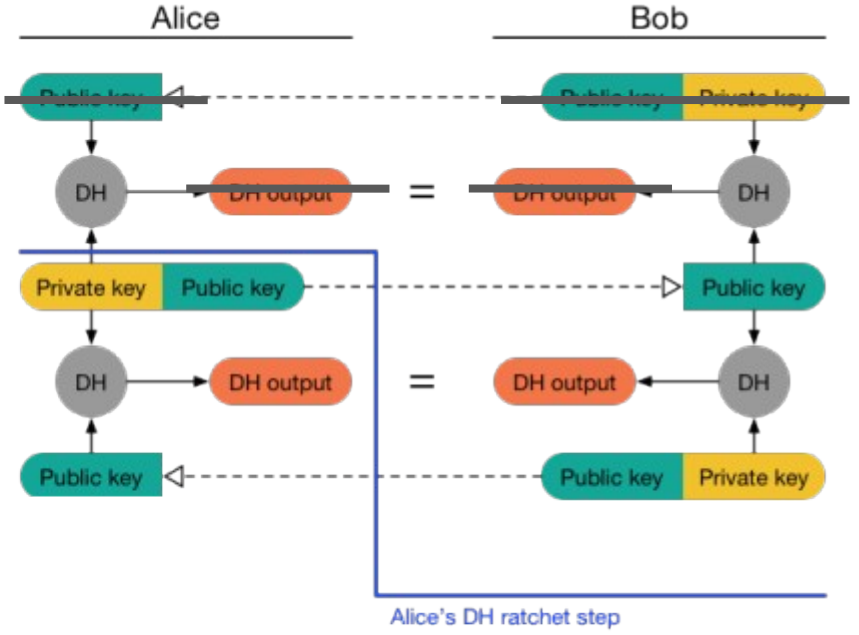
- **HMAC**: Keyed-Hashing for Message Authentication
- HMAC-based Extract-and-Expand Key Derivation Function (**HKDF**)

$$\mathrm{HMAC}(K, m) = \mathrm{H}\left((K' \oplus opad) \parallel \mathrm{H}\left((K' \oplus ipad) \parallel m \right) \right)$$

$$K' = \begin{cases} \mathrm{H}(K) & \text{if } K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

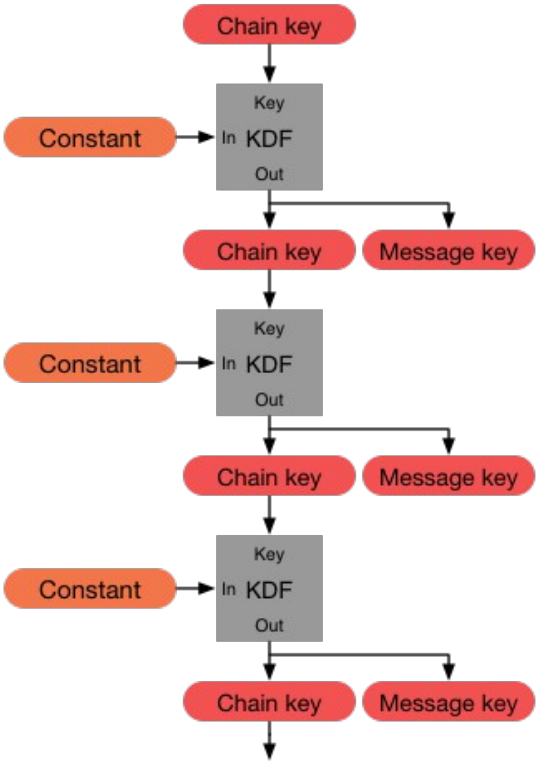
Diffie-Hellman can be a ratchet, too!



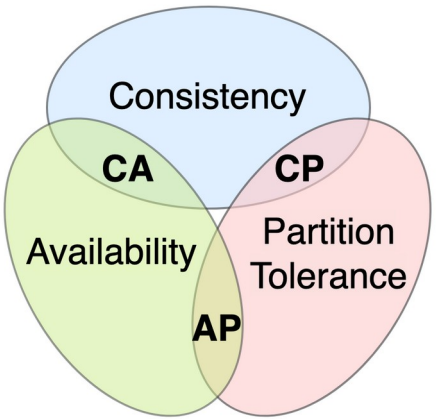


Post compromise security!

There is a real-world problem with these KDF chains

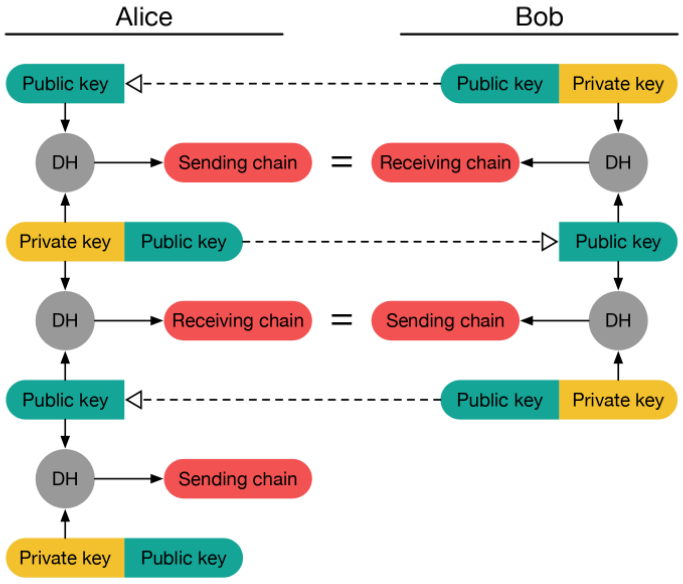


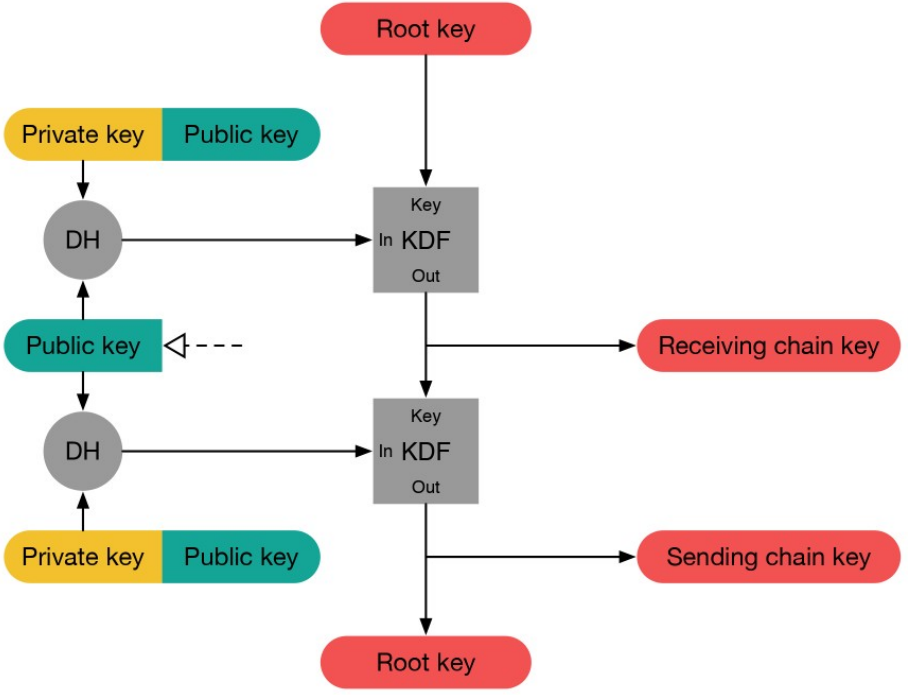
- Devices go offline
  - (ref. «Off-The-Record/OTR»)

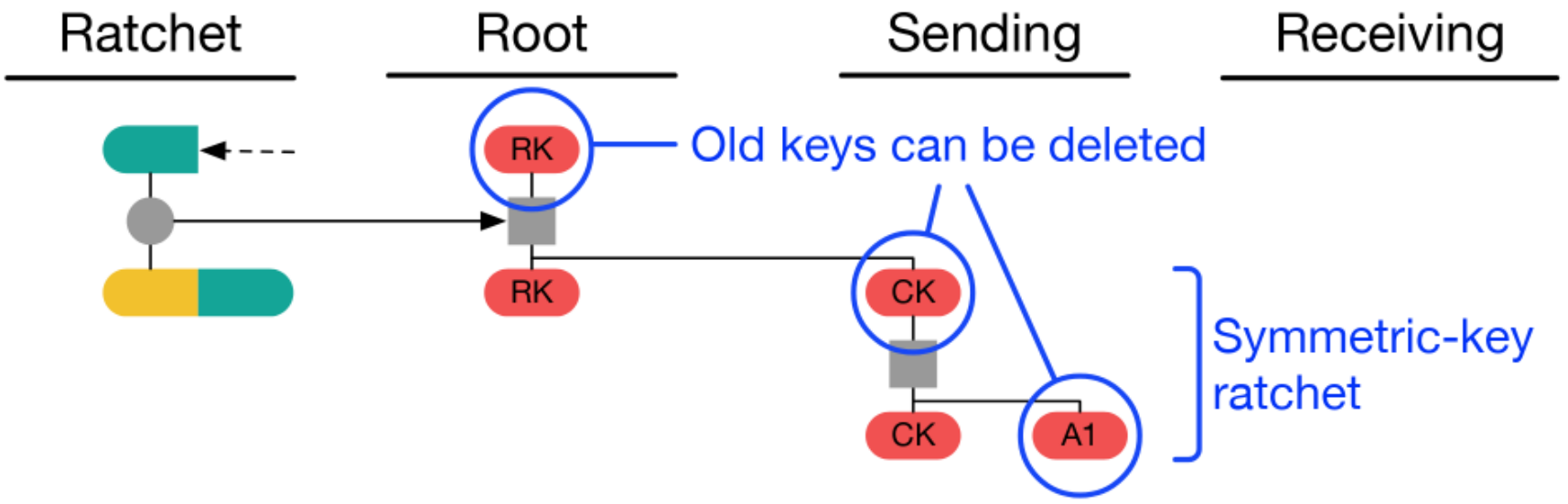


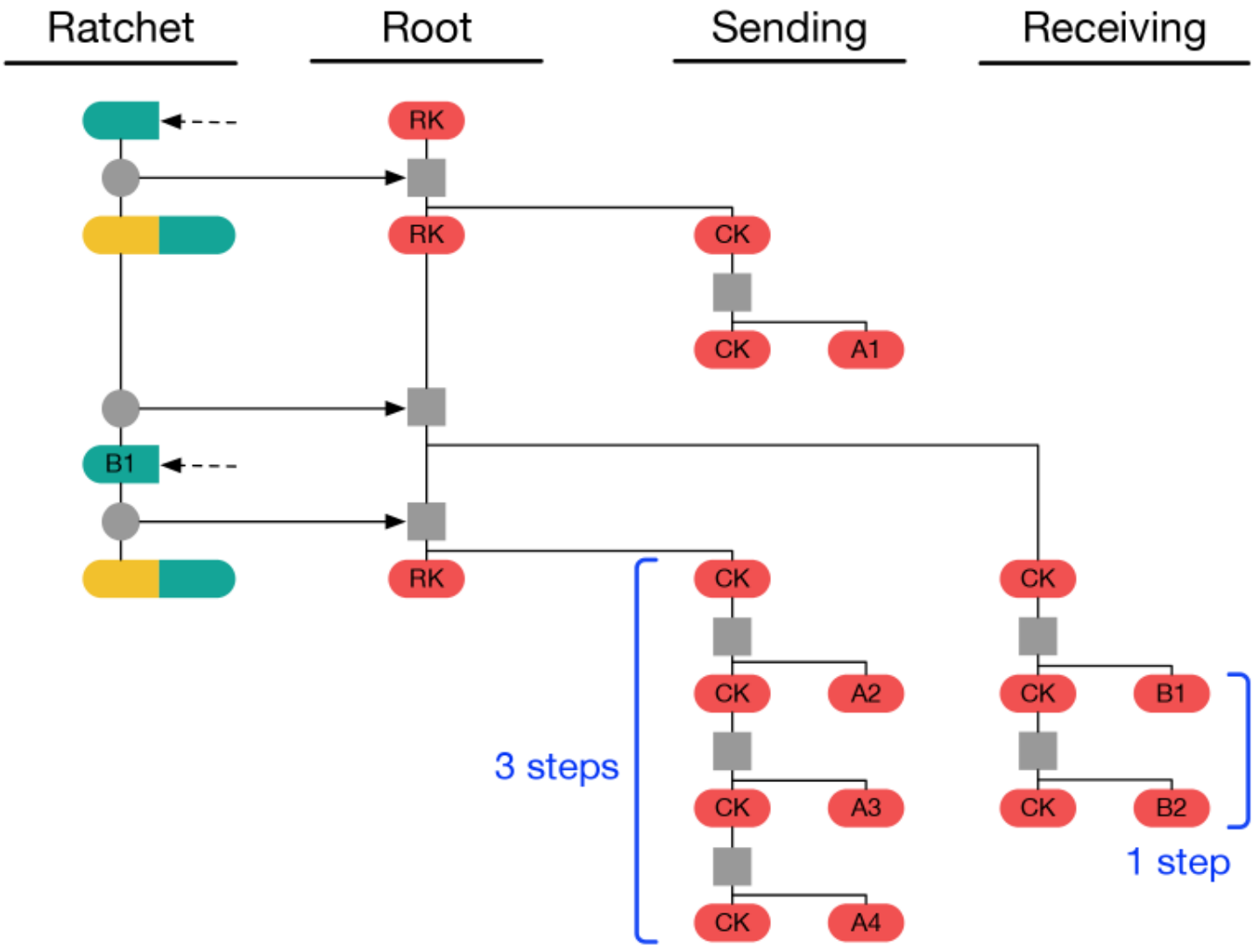
CAP-Theorem JamieMcCarthy - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=126737699

Let’s separate Receiving and Sending chain

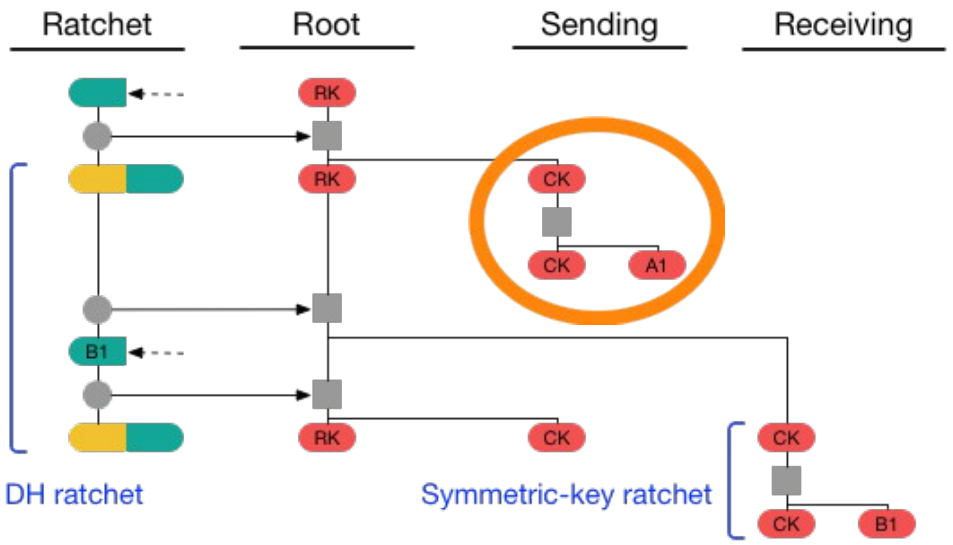





Ratchet
Root
Sending
Receiving
RK
RK
Old keys can be deleted
CK
CK
A1
Symmetric-key
ratchet


Ratchet
Root
Sending
Receiving
RK
RK
CK
CK
A1
B1
RK
CK
CK
A2
CK
A3
CK
A4
3 steps
CK
CK
B1
CK
B2
1 step

We present a somewhat usable algorithm with PCS



- Compromised chain keys are healed after a DH step



Axolotl: CC-BY-SA 3.0 Teresa Such Ferrer https://commons.wikimedia.org/wiki/File:Ambystoma_mexicanum_-_Teresa_Such_Ferrer.jpg

https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf

The Double Ratchet Algorithm

Trevor Perrin (editor) Moxie Marlinspike
Rolfe Schmidt (revision 3+)

Revision 4 , 2025-11-04

Contents

1. Introduction **3**

2. Overview **3**
2.1. KDF chains . . . 3
2.2. Symmetric-key ratchet . . . 5
2.3. Diffie-Hellman ratchet . . . 6
2.4. Double Ratchet . . . 13
2.6. Out-of-order messages . . . 17

3. Double Ratchet **18**
3.1. External functions . . . 18
3.2. State variables . . . 19
3.3. Initialization . . . 19
3.4. Encrypting messages . . . 20
3.5. Decrypting messages . . . 20

4. Double Ratchet with header encryption **23**
4.1. Overview . . . 23
4.2. External functions . . . 27
4.3. State variables . . . 27
4.4. Initialization . . . 27
4.5. Encrypting messages . . . 28
4.6. Decrypting messages . . . 29

5. The Sparse Post-Quantum Ratchet **30**
5.1 Sparse Continuous Key Agreement . . . 31
5.2 External functions . . . 32
5.3 State variables . . . 32
5.4 Initialization . . . 32
5.5 Encrypting Messages . . . 33
5.6 Decrypting Messages . . . 34

5.7 Clearing past epoch state . . . 35

6. The Triple Ratchet: combining Secure Messaging protocols for hybrid security **37**
6.1 Overview . . . 37
6.2 State variables . . . 37
6.3 External functions . . . 37
6.4 Initialization . . . 37
6.5 Encrypting Messages . . . 38
6.6 Decrypting Messages . . . 38

7. Implementation considerations **38**
7.1. Integration with PQXDH . . . 38
7.2. Recommended cryptographic algorithms . . . 40

8. Security considerations **41**
8.1. Secure deletion . . . 41
8.2. Recovery from compromise . . . 42
8.3. Cryptanalysis and ratchet public keys . . . 42
8.4. Deletion of skipped message keys . . . 42
8.5. Deferring new ratchet key generation . . . 43
8.6. Truncating authentication tags . . . 43
8.7. Implementation fingerprinting . . . 43
8.8 Choice of SCKA protocol . . . 43
8.9 Effect of dropped messages on PCS . . . 44
8.10 Deletion of old KDF Chain state . . . 44
8.11 Harvest Now, Decrypt Later Attacks . . . 44

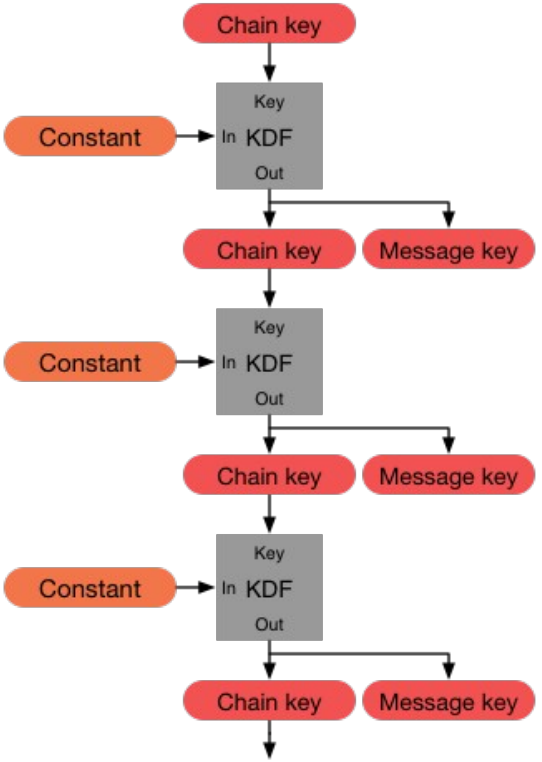
9. IPR **44**

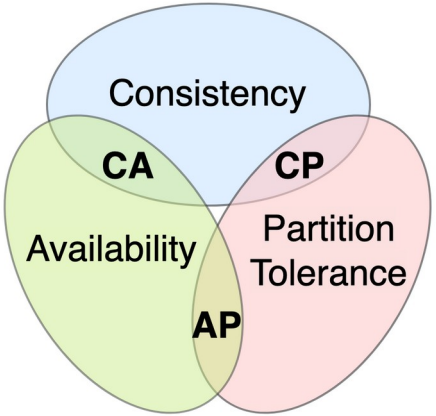
10. Acknowledgements **44**

9. References **45**

How do we achieve group messaging?

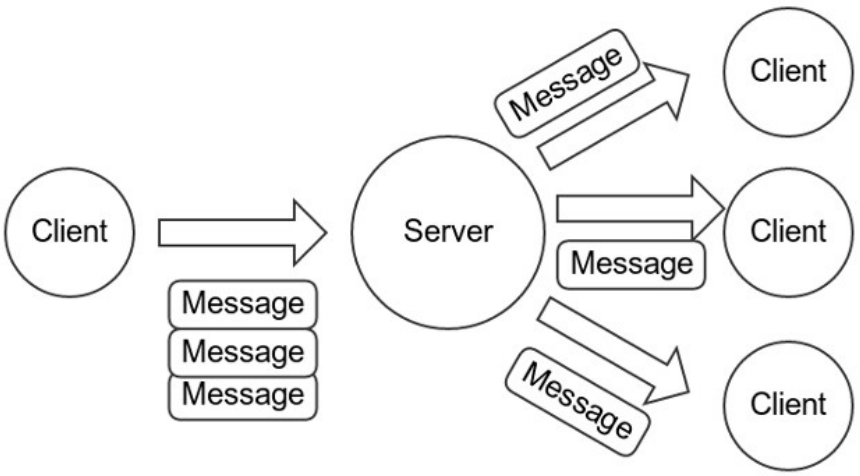
One shared, central ratchet?

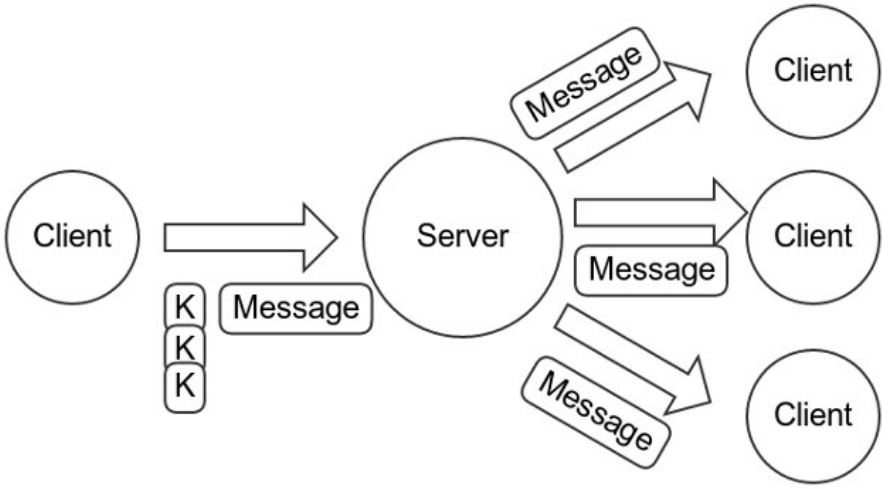




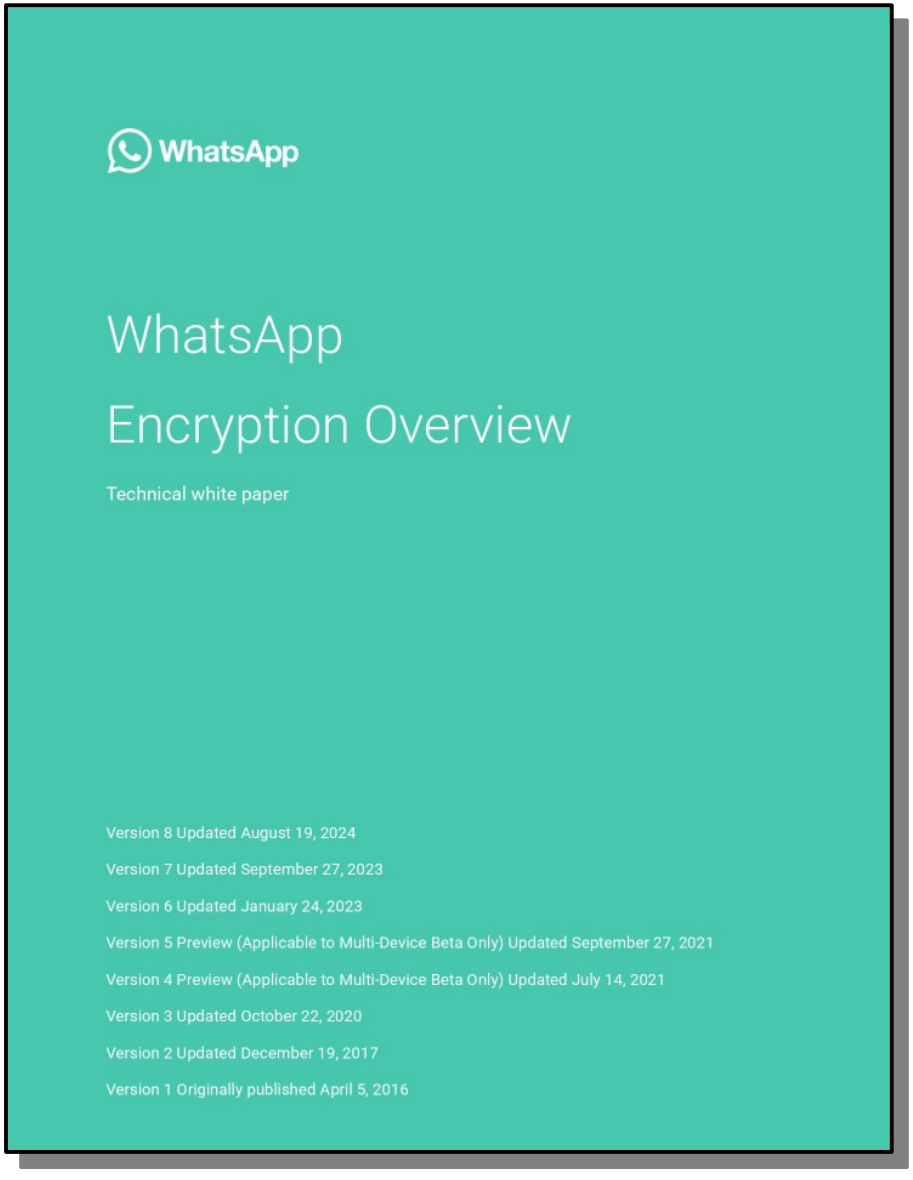
CAP-Theorem JamieMcCarthy - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=126737699

Peer to peer? «*Client side fanout*»



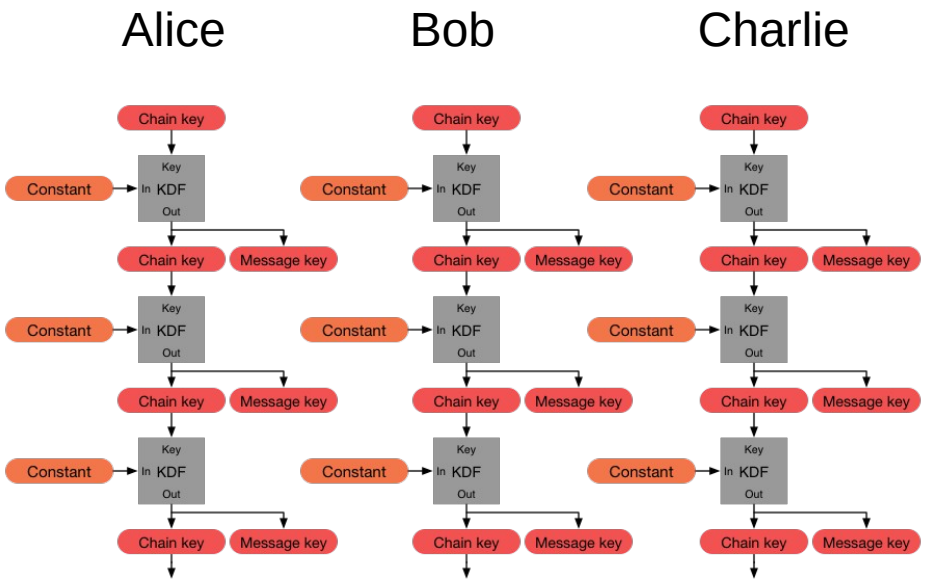


https://signal.org/blog/private-groups/ (2014)

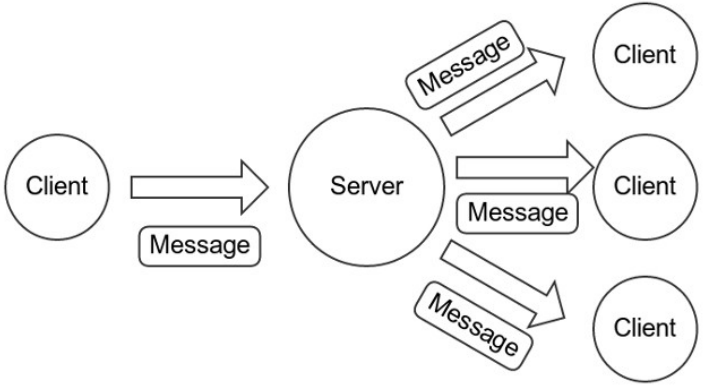


https://scontent.fosl5-1.fna.fbcdn.net/v/t39.8562-6/456045578_868167208055607_7997729792527463495_n.pdf?_nc_cat=107&ccb=1-7&_nc_sid=b8d81d&_nc_ohc=E8UIJWYZgw8Q7kNvwE837or&_nc_oc=AdlV2URfrPxJbJS_kvmvlDxZObg1-NuZHhWE-TfsDpd7gpE8iLMXQqKzRCfXTFFva50&_nc_zt=14&_nc_ht=scontent.fosl5-1.fna&_nc_gid=2YP2r0ROpL5hzKi1OEyA9A&oh=00_Afj8oW-n-agm_Wk6yjzKW8LoNFYmfrjuA0F2Diajkvldtg&oe=692B6D63

Sender keys: «*Server side fanout*»



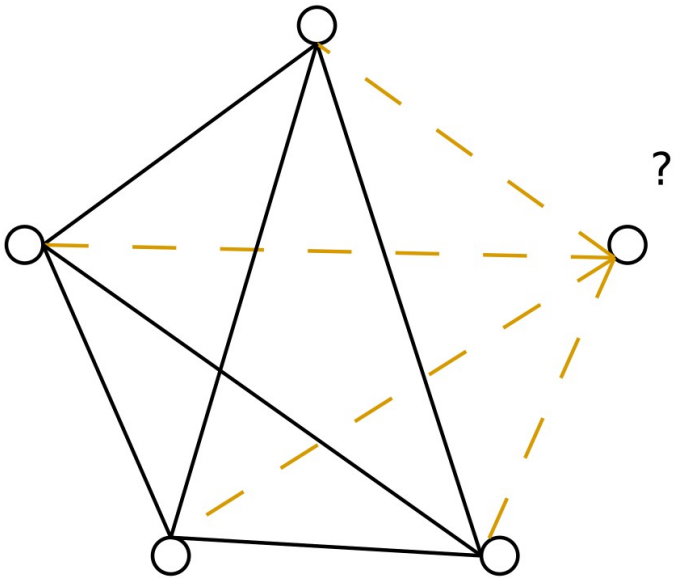
- Every group member creates a sending chain
- Distribute chain key via pairwise double ratchet to all other members
- Every member maintains n-1 receiving chains



Signal

Rekeying is expensive

- Quadratic/Linear complexity



You can add up to 1024 members to a group if you're an admin or if admins have chosen to keep their group open for anyone on WhatsApp to join.

New group members won't be able to see events or messages sent in the group before they joined.

- Size limit of 1000

https://faq.whatsapp.com/841426356990637/
https://support.signal.org/hc/en-us/articles/360007319331-Group-chats

Messaging Layer Security (MLS)

MLS

Messaging Layer Security (MLS) is an IETF working group building a modern, efficient, secure group messaging protocol.

View My GitHub Profile

What is Messaging Layer Security (MLS)?

Messaging Layer Security (MLS) is a security layer for encrypting messages in groups of size two to many. It is being built by the MLS working group and designed to be efficient, practical and secure. The proposed MLS specification is in two parts:

- an architecture document (source, diffs) setting out the context, problem domain and security requirements, and
- a protocol document (source, diffs) defining the protocol itself.

The right place to ask any questions or keep on top of development is the mailing list, or you can check out the current documents. There's also a public Wire group for chatting.

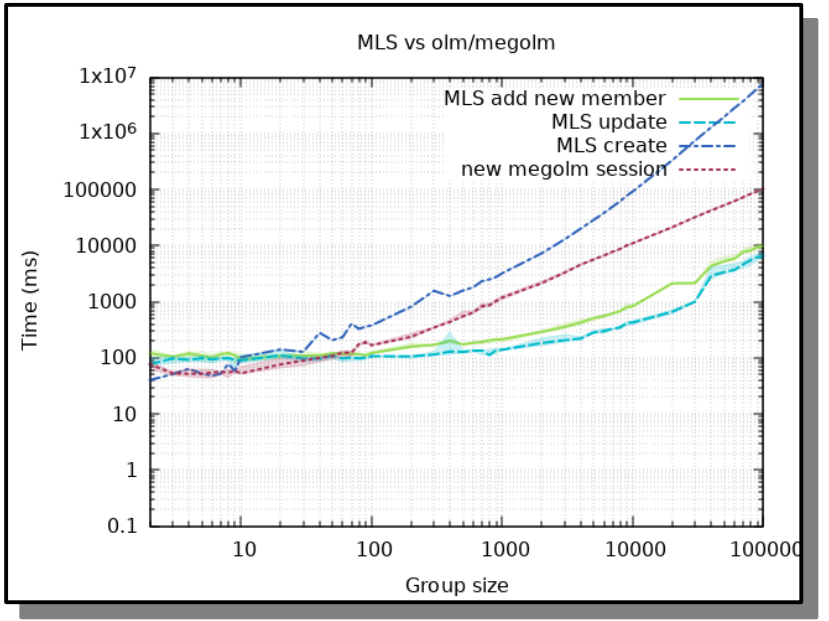
Contributing

If you have insights, comments or just want to help, please do join in! We're particularly interested in hearing from coders who would like to help implement draft versions, and from researchers who would like to formally analyse draft specifications, but all input is valuable. Be aware that all contributions fall under the IETF's Note Well.

FAQ

Should I use this right now?

Yes! The protocol has been approved by the IESG and has been published as RFC9420. There are several implementations, of which some are open source.



https://messaginglayersecurity.rocks/
https://arewemlsyet.com/

On Ends-to-Ends Encryption:

Asynchronous Group Messaging with Strong Security Guarantees

Katriel Cohn-Gordon*[1], Cas Cremers[2], Luke Garratt[1], Jon Millican[3], and Kevin Milner[1]

[1]Department of Computer Science, University of Oxford
[2]CISPA Helmholtz Center for Information Security, Saarland Informatics Campus, Germany
[3]Facebook

Version 2.3, March 2nd, 2020†

Abstract

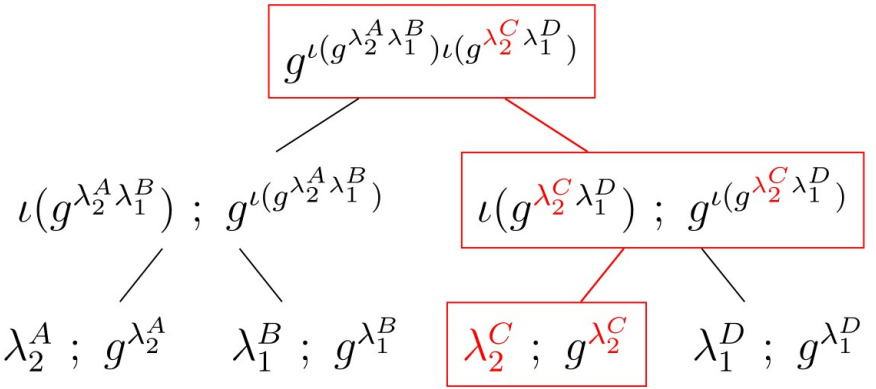
In the past few years secure messaging has become mainstream, with over a billion active users of end-to-end encryption protocols such as Signal. The Signal Protocol provides a strong property called post-compromise security to its users. However, it turns out that many of its implementations provide, without notification, a weaker property for *group* messaging: an adversary who compromises a single group member can read and inject messages indefinitely.

We show for the first time that post-compromise security can be achieved in realistic, asynchronous group messaging systems. We present a design called Asynchronous Ratcheting Trees (ART), which uses tree-based Diffie-Hellman key exchange to allow a group of users to derive a shared symmetric key even if no two are ever online at the same time. ART scales to groups containing thousands of members, while still providing provable security guarantees. It has seen significant interest from industry, and forms the basis for two draft IETF RFCs and a chartered working group. Our results show that strong security guarantees for group messaging are practically achievable in a modern setting.

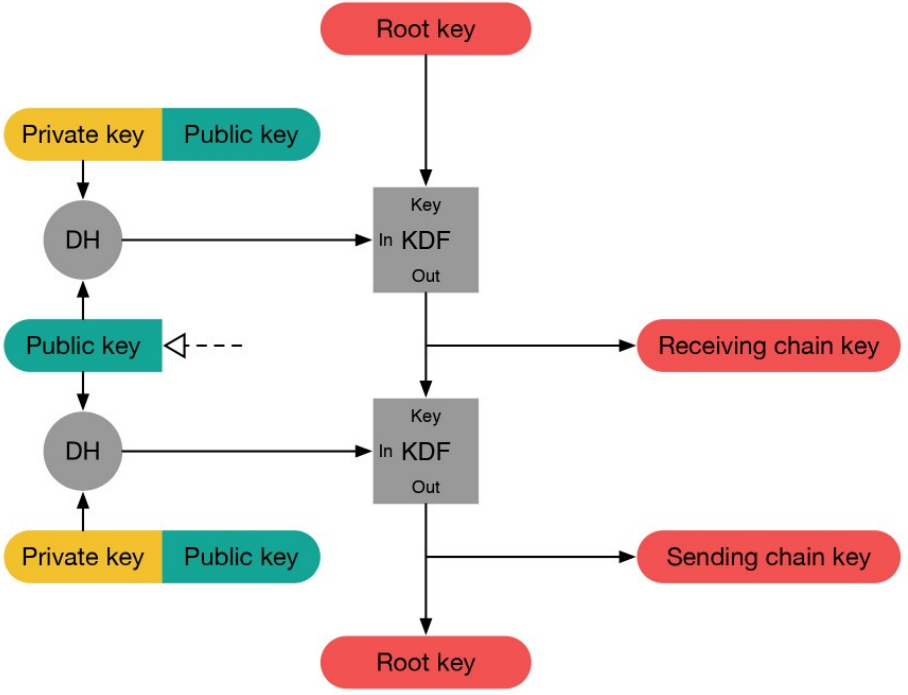
*K.C-G. thanks Merton College and the Oxford CDT in Cyber Security for their support.
†An extended abstract of this paper appears at ACM CCS 2018 [15]; this is the full version. A summary of changes is given in Appendix F.

Asynchronous Ratcheting Trees (ART)

https://eprint.iacr.org/2017/666.pdf



What's so hard about generalizing the Double Ratchet?

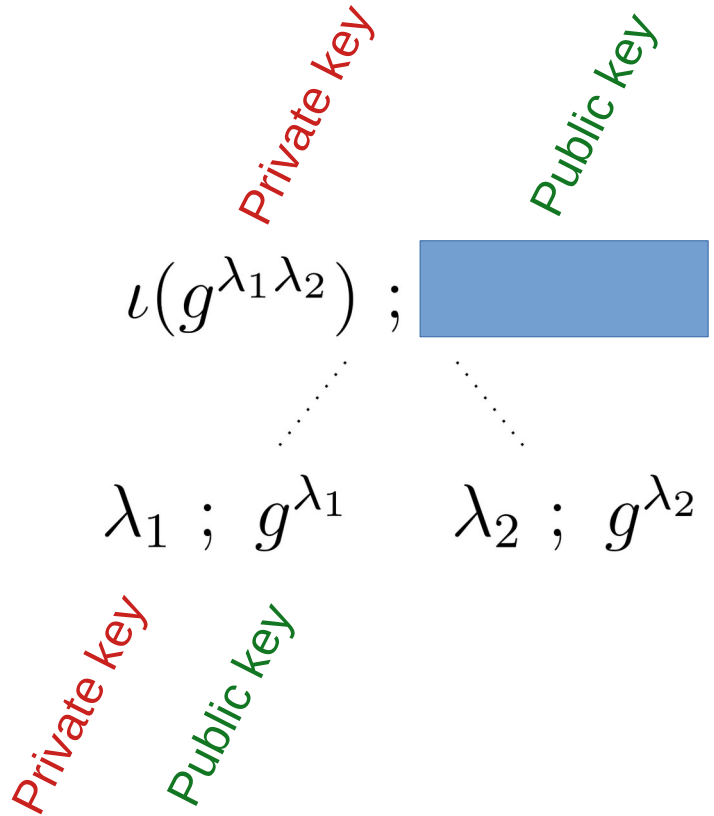


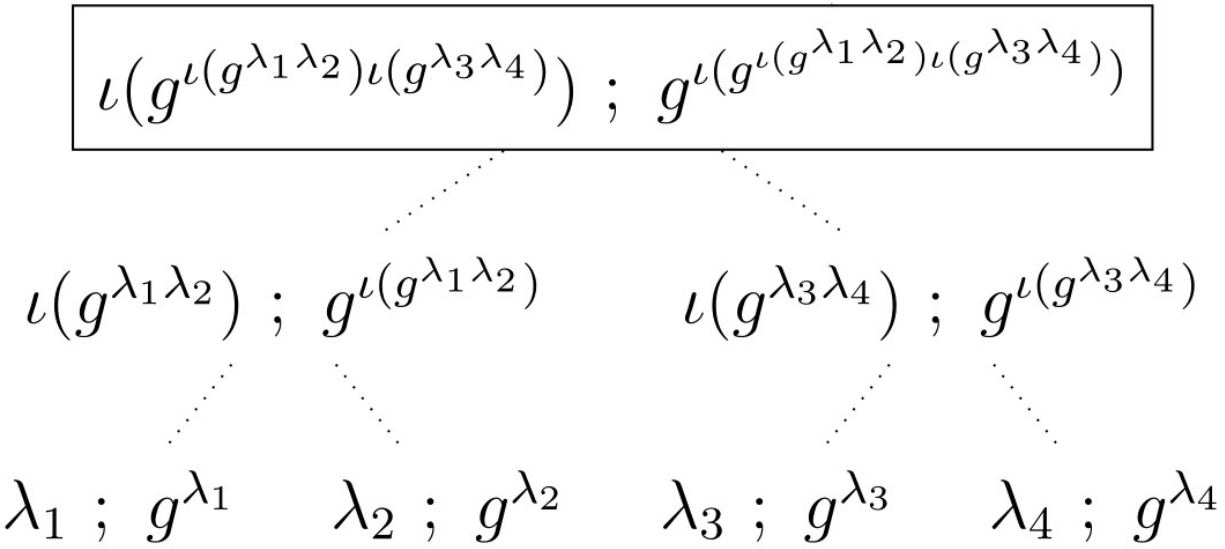
n-party Diffie-Hellmann

- Given $g^{x_0}, \ldots, g^{x_n}$
- and a single x_i
- derive a value *grk* which is hard to compute without knowing one of the x_i

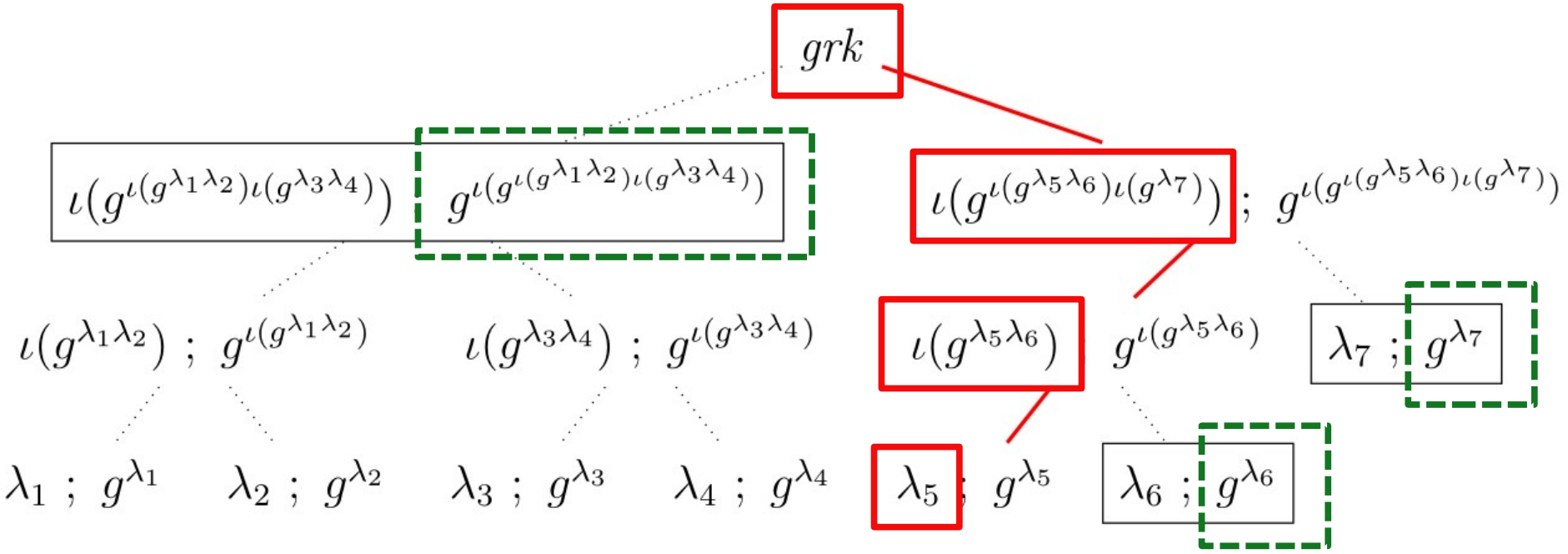
Known open problem (as of 2020, for n>3)

Tree-based group DH





λ_5 and the public *copath* give us *grk*



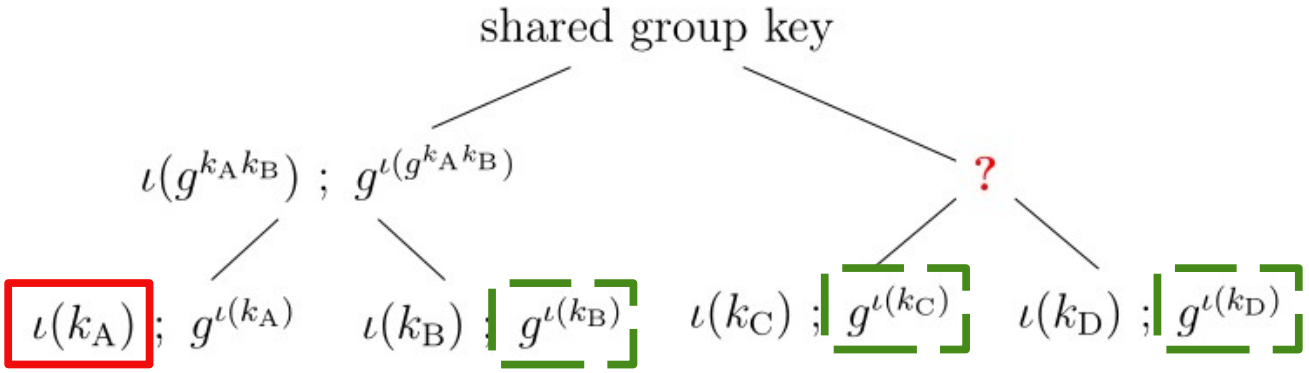
We are two steps away from Asynchronous Ratcheting Trees

- Key update mechanism
- Make it asynchronous

> We show for the first time that post-compromise security can be achieved in realistic, asynchronous group messaging systems. We present a design called Asynchronous Ratcheting Trees (ART), which uses tree-based Diffie-Hellman key exchange to allow a group of users to derive a shared symmetric key even if no two are ever online at the same time.

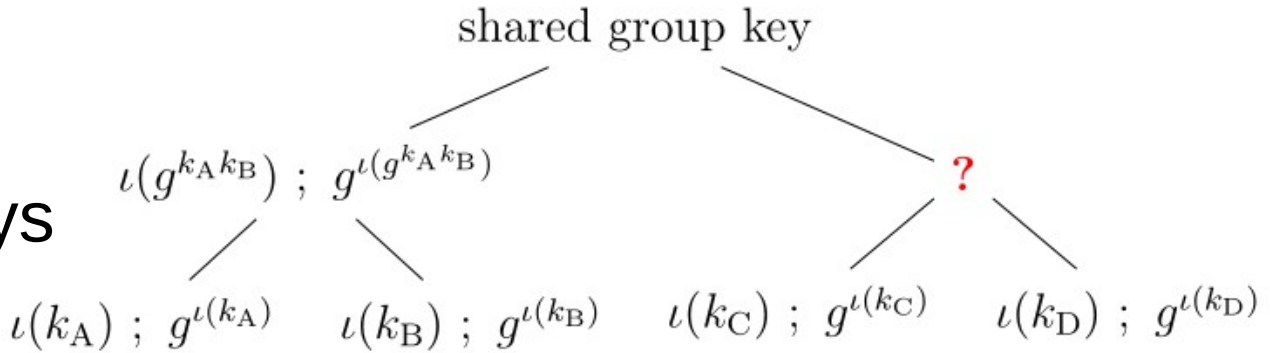
How can Alice initialize the ART?

- Alice makes up private keys for each user (!)
- She uses those to create a tree
- Alice shares the private keys with the users
- We apply our **key update mechanism**



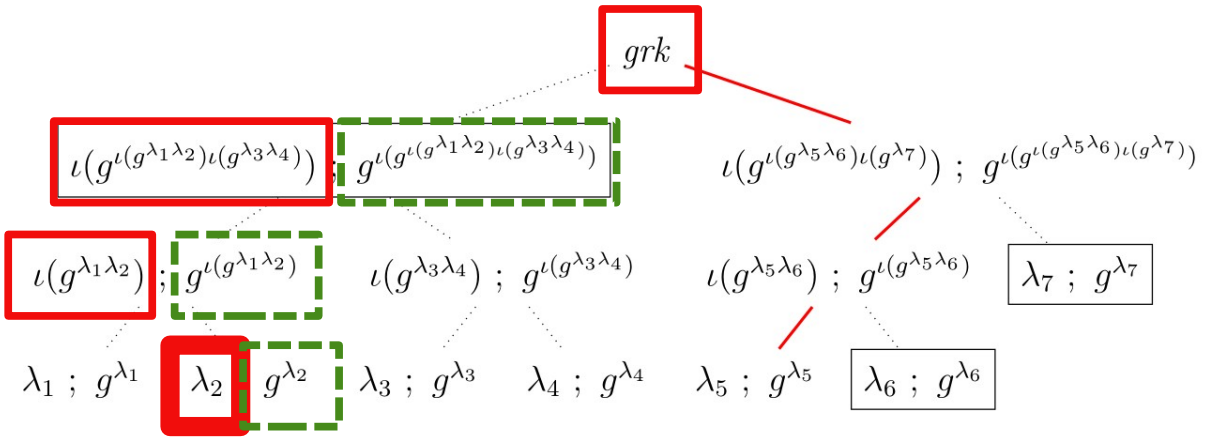
How can Alice initialize the ART?

- All users publish ephemeral DH public keys («*prekeys*»)
- Alice generates a secret setup key *suk*
- Alice creates a private key for each user
  - *DH(suk, prekey)*
- She publishes SUK & the tree public keys

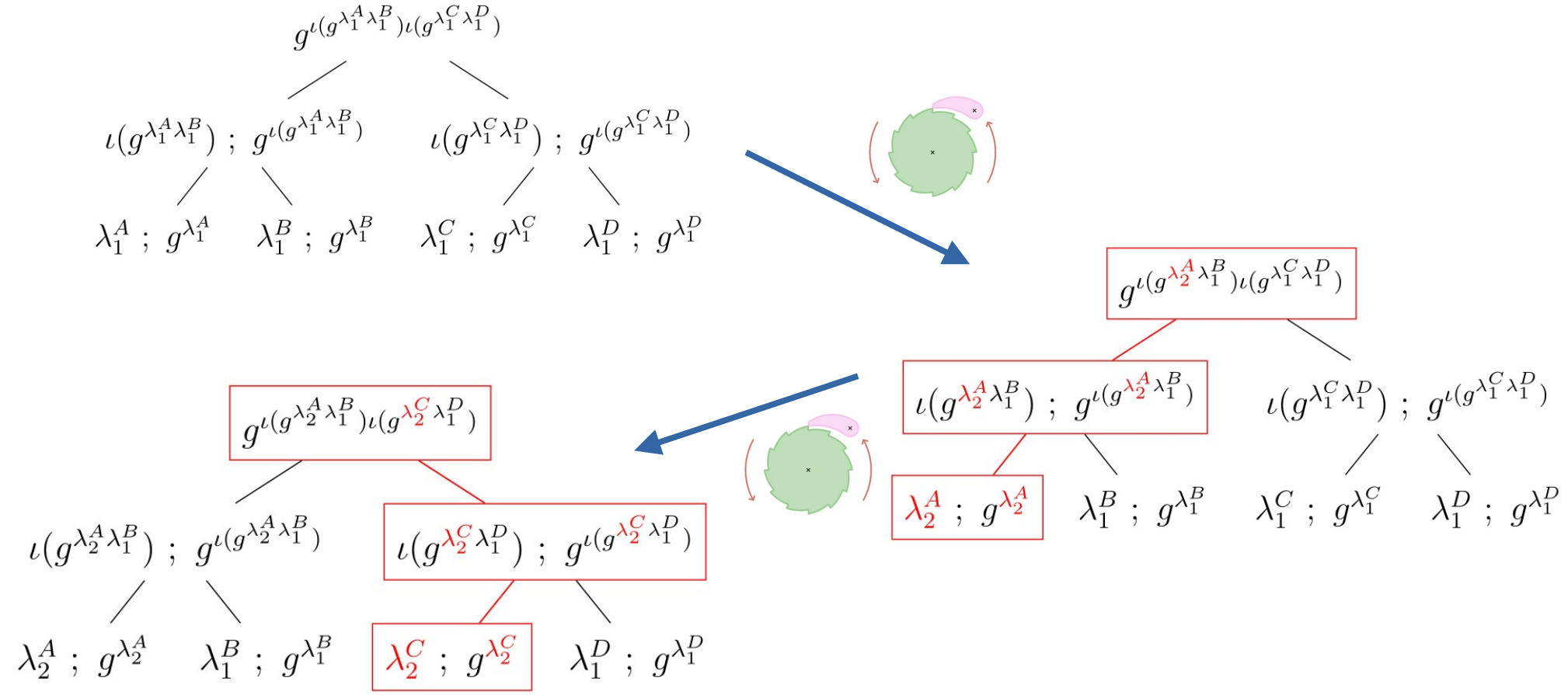


How do we update keys?

- Simple: Just update the path to the root
- Publish new public keys along the path



We now have a FS/PCS ratchet!



Most operations run in O(log(n))!

		number of exponentiations		number of symmetric encryptions		bandwidth		PCS
		sender	per other	sender	per other	sender	per other	
sender keys	setup	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	✗
	ongoing	0	0	1	1	$O(1)$	$O(1)$	
pairwise Signal	setup	$O(n)$	$O(n)$	0	0	$O(n)$	$O(n)$	✓
	ongoing	$O(n)$	$O(1)$	$n-1$	1	$O(n)$	$O(1)$	
our solution	setup	$O(n)$	$O(\log(n))$	0	0	$O(n)$	$O(n)$	✓
	ongoing	$O(\log(n))$	$O(\log(n))$	1	1	$O(\log(n))$	$O(\log(n))$	

MLS adds a «secret tree»

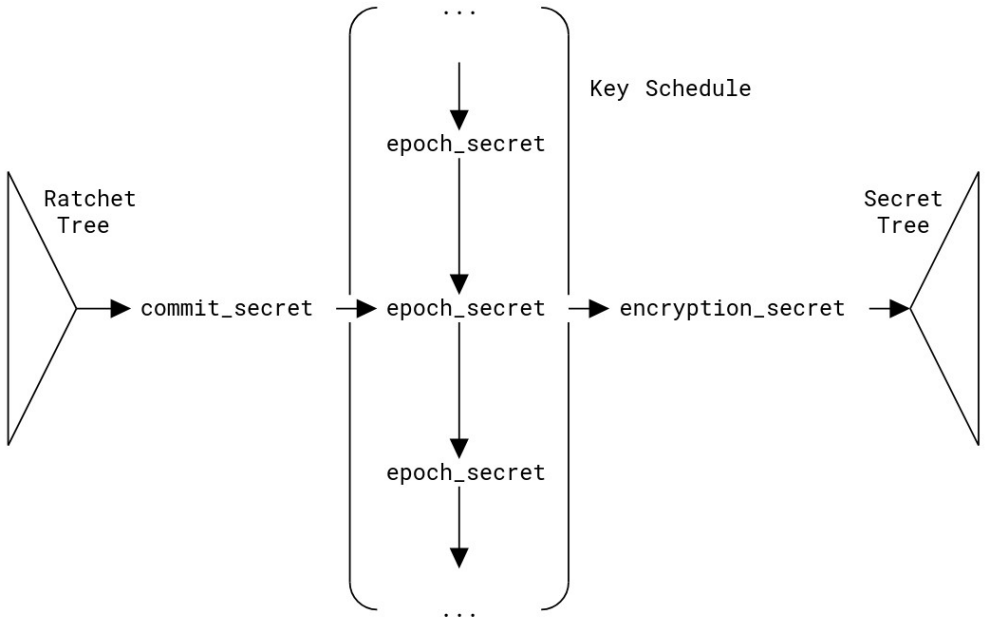


Figure 1: Overview of MLS Group Evolution

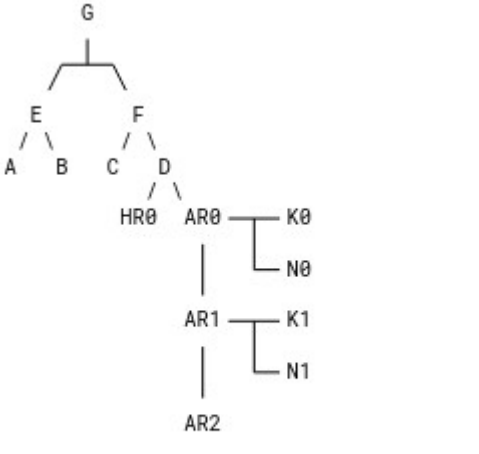
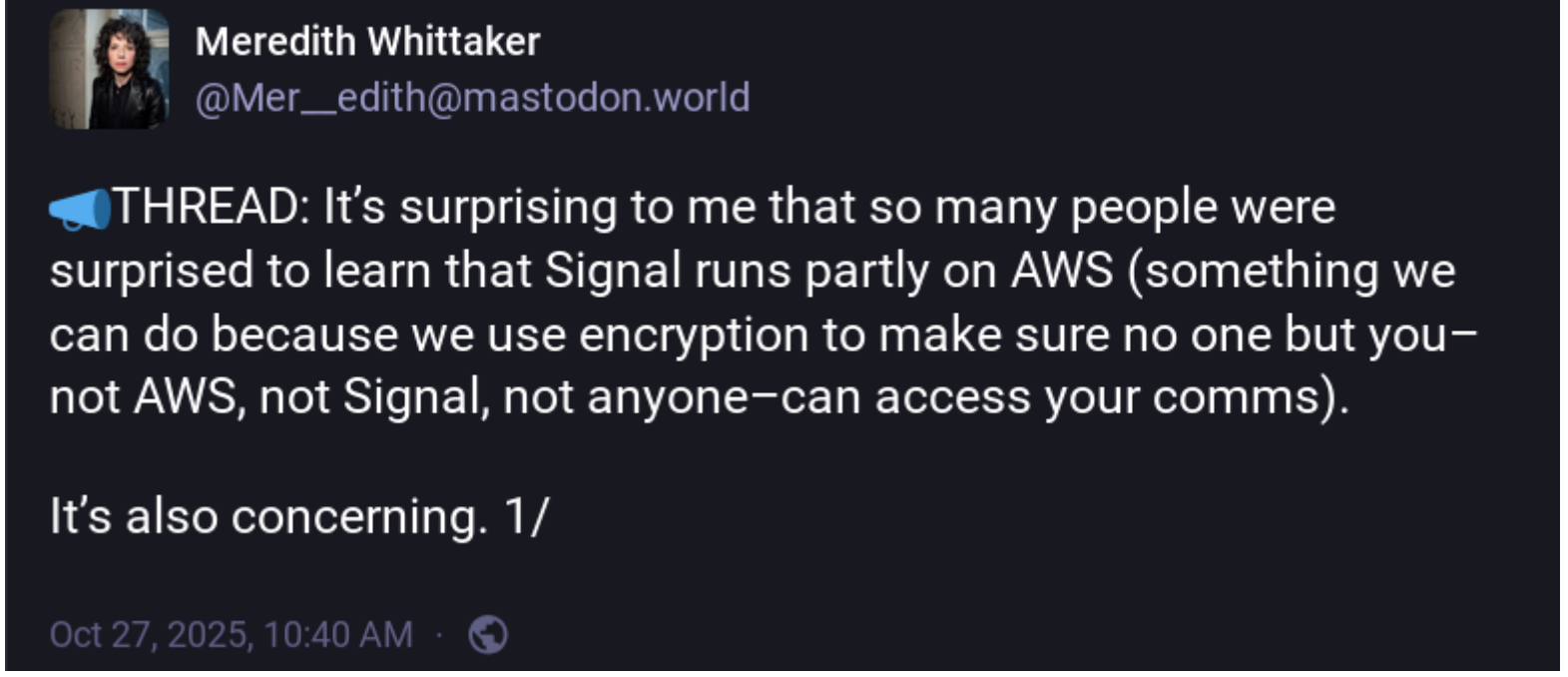


Figure 27: Secret Tree for a Four-Member Group

How about availability?



Meredith Whittaker
@Mer__edith@mastodon.world

📣THREAD: It's surprising to me that so many people were surprised to learn that Signal runs partly on AWS (something we can do because we use encryption to make sure no one but you–not AWS, not Signal, not anyone–can access your comms).

It's also concerning. 1/

Oct 27, 2025, 10:40 AM ·

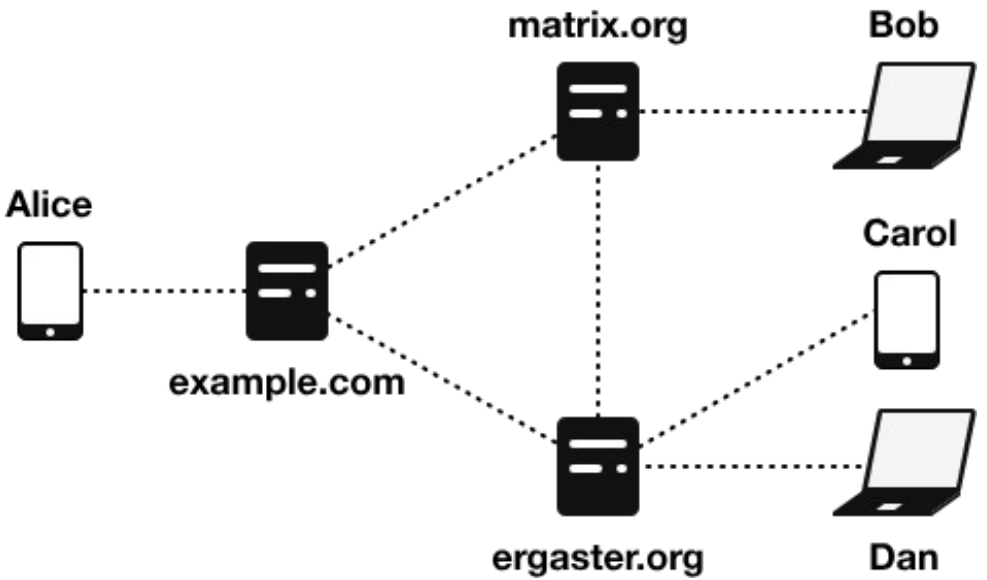


Meredith Whittaker
@Mer__edith

Oct 27

Instant messaging demands near-zero latency. Voice and video in particular require complex global signaling & regional relays to manage jitter and packet loss. These are things that AWS, Azure, and GCP provide at global scale that, practically speaking, others (in the western context) don't. 5/

7



https://matrix.org/docs/matrix-concepts/elements-of-matrix/
Image on next slide: https://element.io/blog/nato-ni2ce-messenger-utilises-the-power-of-decentralised-communication/

Comparing communication architectures.

How an open standard federated network could work for NATO and its members.



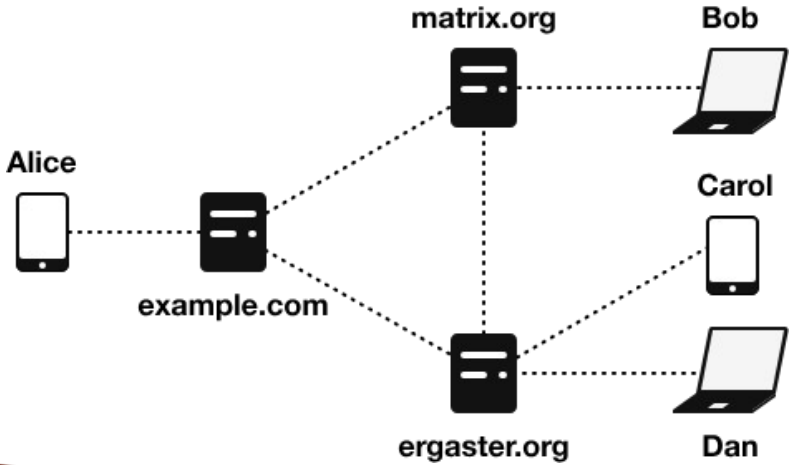


	Centralised	Decentralised
CONTROLLER	The vendor (Meta)	The end user's organisation
PROTOCOL	Proprietary	Open standard
CODE	Closed source	Open source
REVENUE	Targeted advertising (data mining)	Subscription to enterprise distribution
GOVERNANCE	US law	By the country where the server is located
CONTINUITY	Global outages	Fault tolerant
OWNERSHIP	Unilateral: A single point of ownership and control	Multilateral: No single point of ownership and control

Source: Element

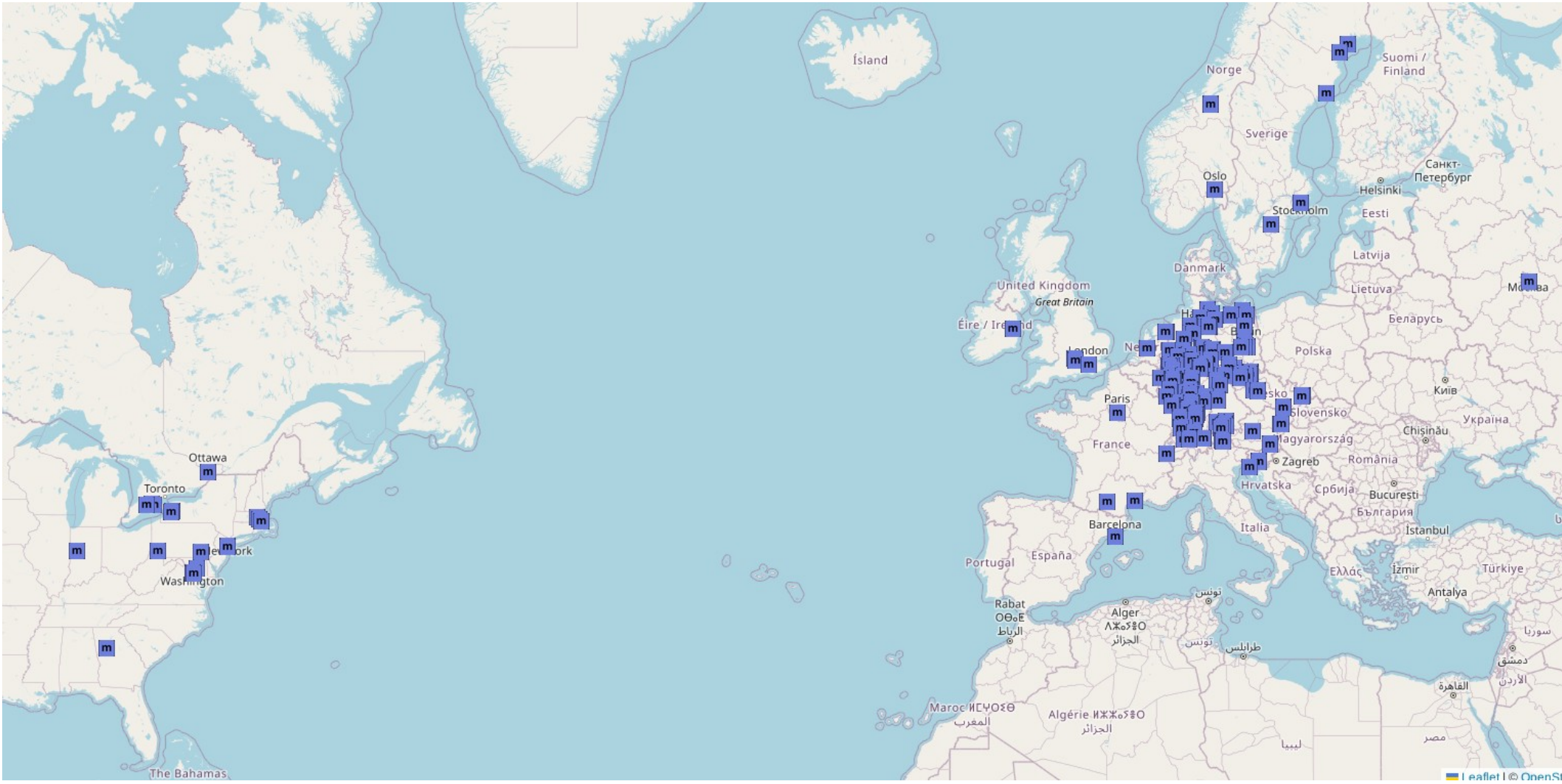
Matrix is a decentralized E2EE-capable protocol

- Open Standard
- Decentralized
- Based on Olm
  - Double Ratchet
- And **Megolm**





https://github.com/matrix-org/vodozemac
https://matrix.org/docs/matrix-concepts/elements-of-matrix/



https://maptrix.derhagen.eu/map.html

Megolm is a different beast

$$R_{i,0} = \begin{cases} H_0\left(R_{2^{24}(n-1),0}\right) & \text{if } \exists n | i = 2^{24}n \\ R_{i-1,0} & \text{otherwise} \end{cases}$$

$$R_{i,1} = \begin{cases} H_1\left(R_{2^{24}(n-1),0}\right) & \text{if } \exists n | i = 2^{24}n \\ H_1\left(R_{2^{16}(m-1),1}\right) & \text{if } \exists m | i = 2^{16}m \\ R_{i-1,1} & \text{otherwise} \end{cases}$$

$$R_{i,2} = \begin{cases} H_2\left(R_{2^{24}(n-1),0}\right) & \text{if } \exists n | i = 2^{24}n \\ H_2\left(R_{2^{16}(m-1),1}\right) & \text{if } \exists m | i = 2^{16}m \\ H_2\left(R_{2^{8}(p-1),2}\right) & \text{if } \exists p | i = 2^{8}p \\ R_{i-1,2} & \text{otherwise} \end{cases}$$

$$R_{i,3} = \begin{cases} H_3\left(R_{2^{24}(n-1),0}\right) & \text{if } \exists n | i = 2^{24}n \\ H_3\left(R_{2^{16}(m-1),1}\right) & \text{if } \exists m | i = 2^{16}m \\ H_3\left(R_{2^{8}(p-1),2}\right) & \text{if } \exists p | i = 2^{8}p \\ H_3\left(R_{i-1,3}\right) & \text{otherwise} \end{cases}$$

- It sometimes drops FS on purpose
- And moves PCS to the application layer



Matrix and MLS - Water, Oil and Mayonnaise

Nico



The blender?

Decentralized state vs central synchronisation

Centralization requirement for one shared context & membership, kinda.

State resolution complicates perfect forward secrecy

- If you want to encrypt to one shared set of users and throw away keys ASAP, you can't wait indefinitely for parallel modifications of membership

HEIDELBERG LAUREATE FORUM

