Messaging Layer Security

Past, Present, and Future

Richard Barnes

WSM 2019, Darmstadt, DE


BBN
TECHNOLOGIES


IETF®


Let's
Encrypt


TLS 1.3

Past

Context

Lots of secure messaging apps

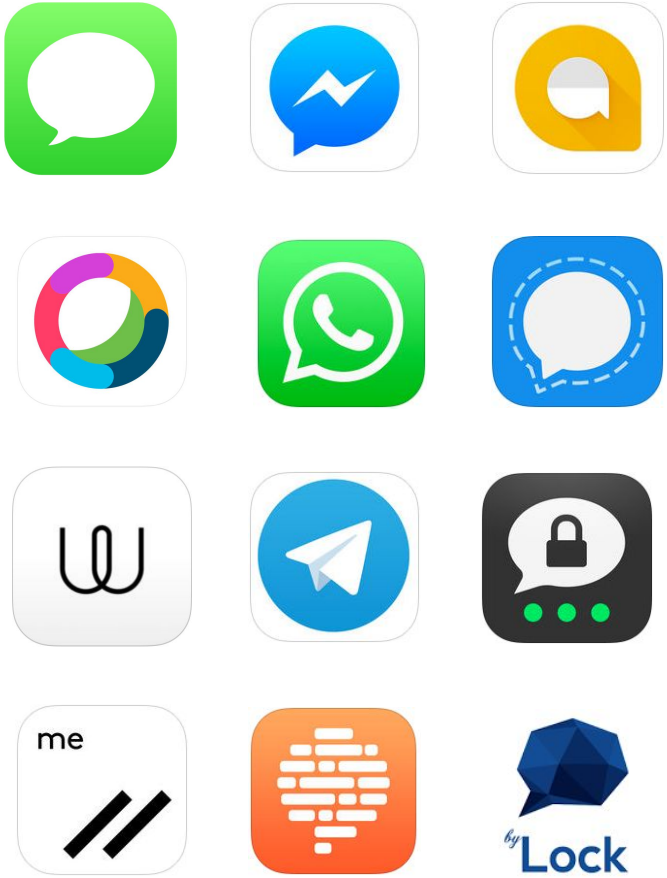
Some use similar protocols…

… some are quite different

… but all have similar challenges

Wildly different levels of analysis

Everyone maintaining their own libraries



In the beginning...

Industry/academic collaboration on TLS 1.3
What's the next thing?
A spec for secure messaging might be useful...

Groups are hard with libsignal
What about trees?

On Ends-to-Ends Encryption:
Asynchronous Group Messaging with Strong Security Guarantees

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

Say hi to your new Facebook friend, Jon.

Hey Jon! How are you?

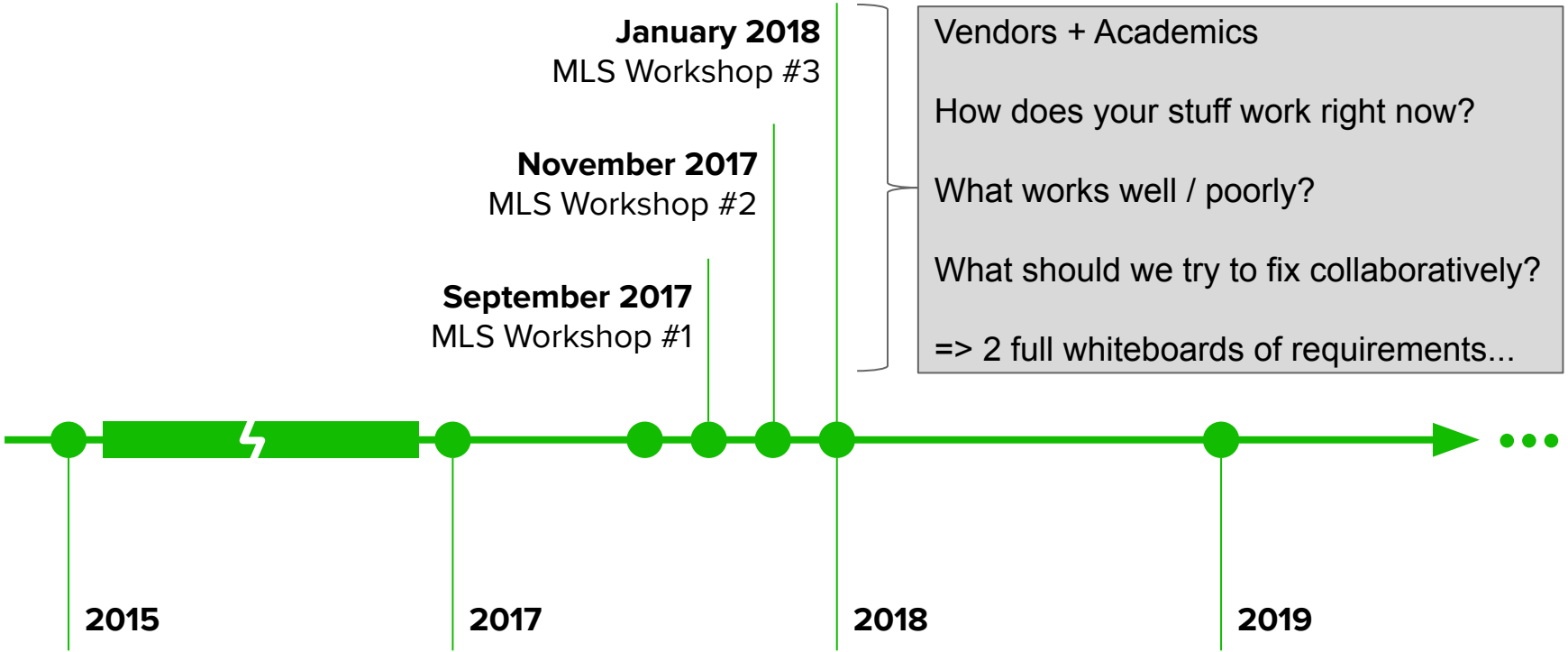
Saw the tree-keying paper yesterday, looks like good work. Reaching out in case you're willing to answer some questions about notation 🙂

Hey Richard 🙂 I'm good thanks, how are you doing?

Sure I can try!

And thanks for reading it! 🙂

Things Start to Come Together



Top-Level Goals

Detailed specifications for an **async** **group** **messaging security** protocol

- **Async** - No two participants online at the same time
- **Group** - Support large, dynamic groups
- **Messaging security** - Modern security properties (FS / PCS)

Code that is reusable in multiple contexts...

... and interoperable between different implementations

Robust, open security analysis and involvement from the academic community

Long-Tail Goals

- Delivery acks and nacks/retries
- Recovery from state loss
- Resilience against intermittent message loss
- Support "identity key per device" model (vs. "sync identity key across devices")
- Avoid leaking the number of devices per user
- Add device when offline (new phone, old phone dead)
- Culling old devices
- History/history integrity

... and several more

MLS vs. TLS

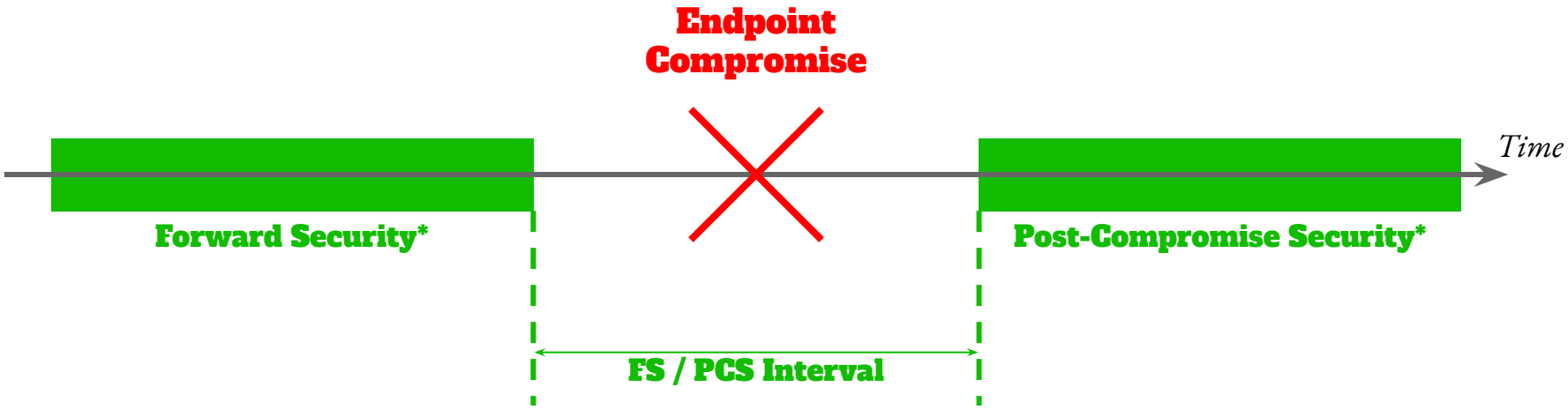
Lots of actors - 2 vs. 10^N

Long lived sessions - seconds vs. months

Lots of mobile devices involved

... each of which is involved in multiple long-lived sessions

Significant probability that some member is compromised at some time in the life of the session

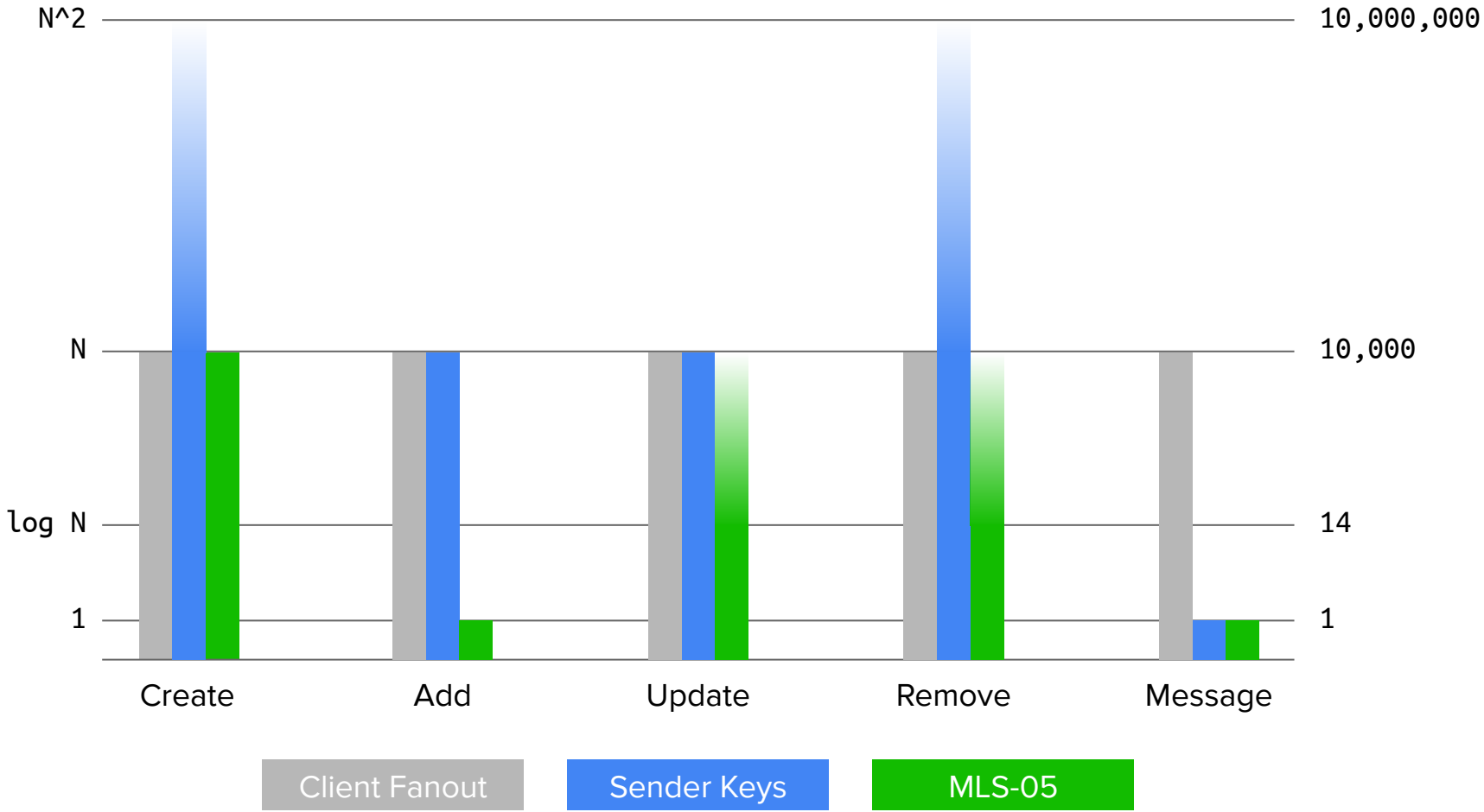


* ... with regard to a participant

Prior Art

mpOTR, (n+1)sec	No PCS
S/MIME, OpenPGP	Linear scaling, difficult to achieve PCS
Client fanout Signal, Proteus, iMessage, et al.	Linear scaling, but good async / PCS properties
Sender Keys WhatsApp, FB, OMEMO, Olm, et al.	Linear scaling, PCS possible but very expensive

Goal: FS/PCS with sub-linear scaling as much as possible


N^2
N
log N
1
10,000,000
10,000
14
1
Create
Add
Update
Remove
Message
Client Fanout
Sender Keys
MLS-05

Digression: Enterprise E2E

Digression: Enterprise Requirements

Good News:

Strong motivations to deny messaging provider access

Customer organization can operate some infra independent of provider

… for example, identity/SSO systems

Less-Good News:

Large, centrally managed groups

- E.g., “everyone in Finance”
- Central control tied to HR

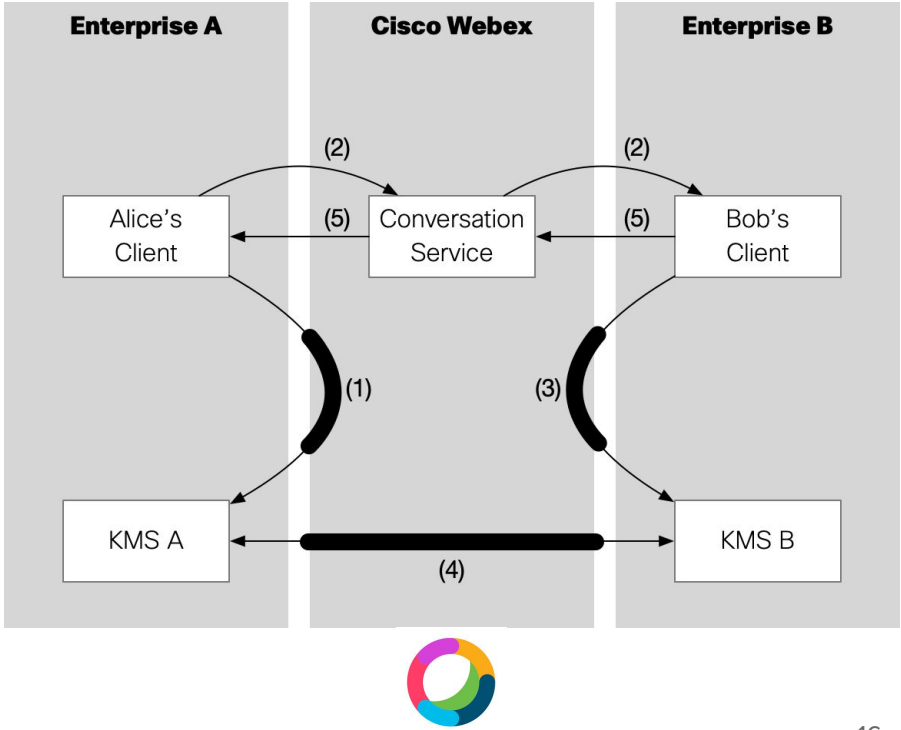
Compliance requirements that require plaintext: recording, discovery, DLP, etc.

Intermediate Solution: Trusted Infra

Webex Teams provides a kind of E2E by splitting having the customer organization operate a key server

Clients for that organization get keys from that server, and everything Webex sees is in ciphertext

Ditto for other services that need plaintext: Search indexing, document previews, etc.



Motivations for more P2P E2E

“Why am I buying a cloud service if I still have to run a server?”

Whoever runs the key server is trusted with all of the customer org’s content, so no compartmentalization for executives, lawyers, security researchers, etc.

Passing around keys encourages more complex application architectures

Forward Secrecy / Post-Compromise Security need to be added manually

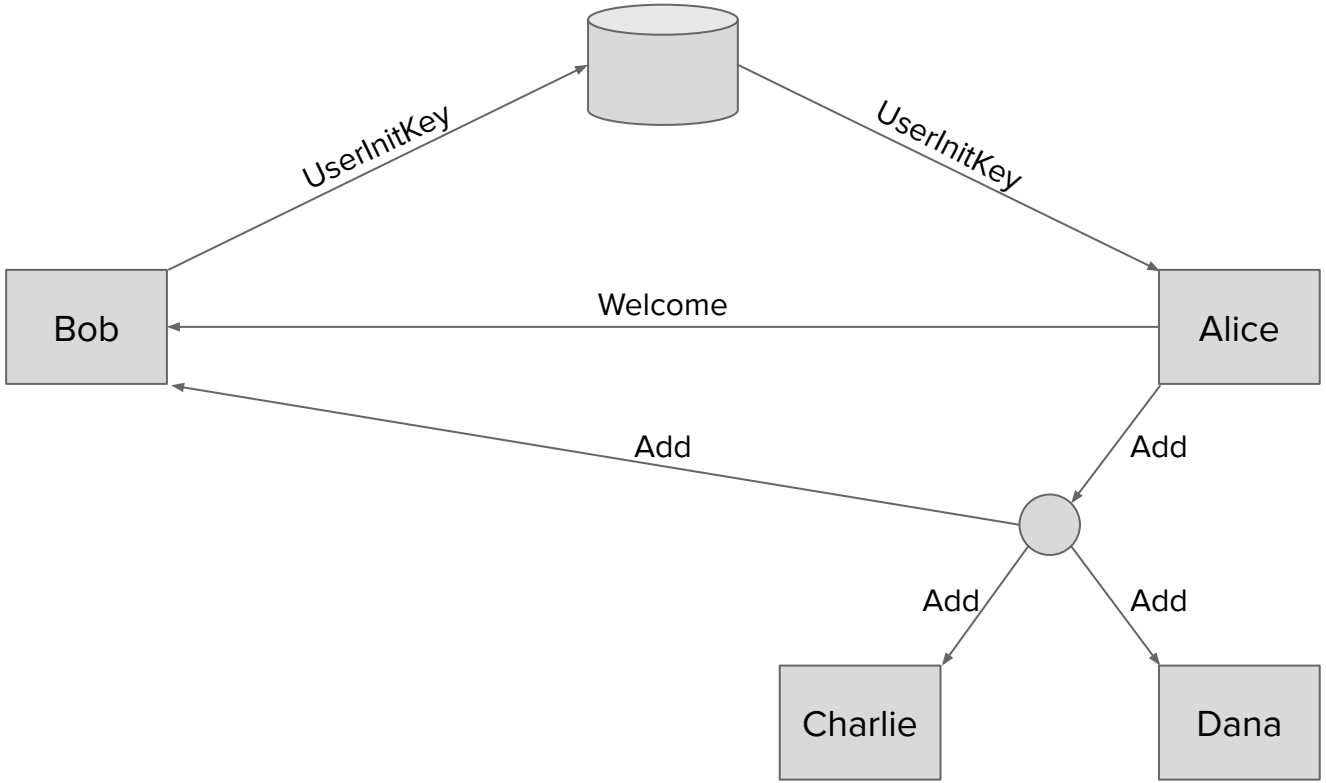
Present

TLS -> MLS

UserInitKey = Authenticated client capabilities + key share

1xRTT instead of 1.5xRTT enables ✨**asynchronous operation**✨

Add


UserInitKey
UserInitKey
Bob
Welcome
Alice
Add
Add
Add
Add
Charlie
Dana

Update / Remove / Synchronization

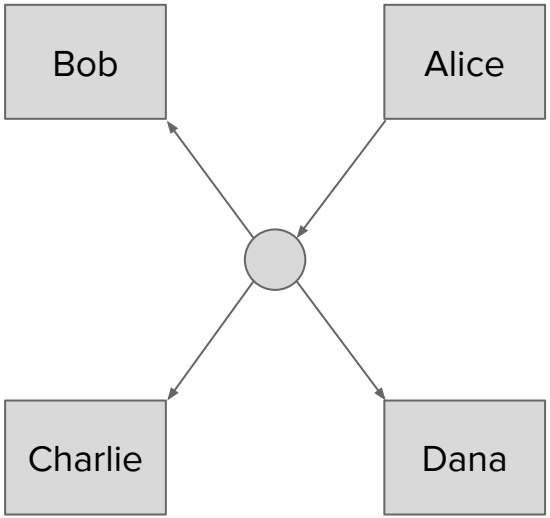
Update and Remove messages are just broadcast to the whole group

All messages must be received in the same order by all clients

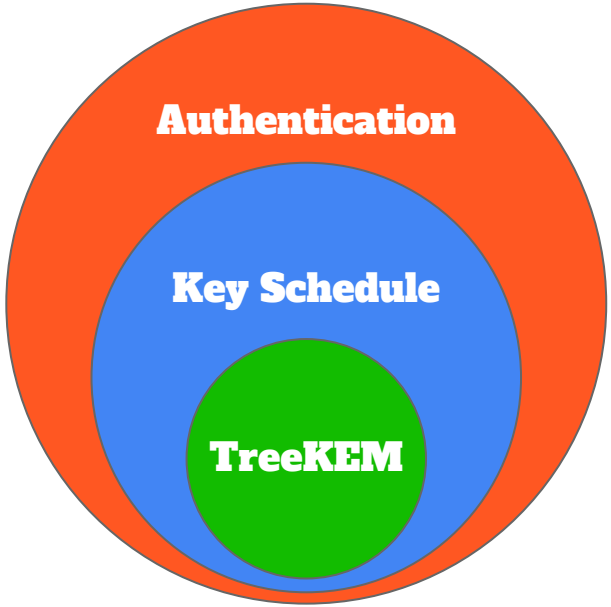
Need for locking / fail+resend

Welcome needs to fate-share with Add

Unlike sender keys / client fan-out, one message to the whole group



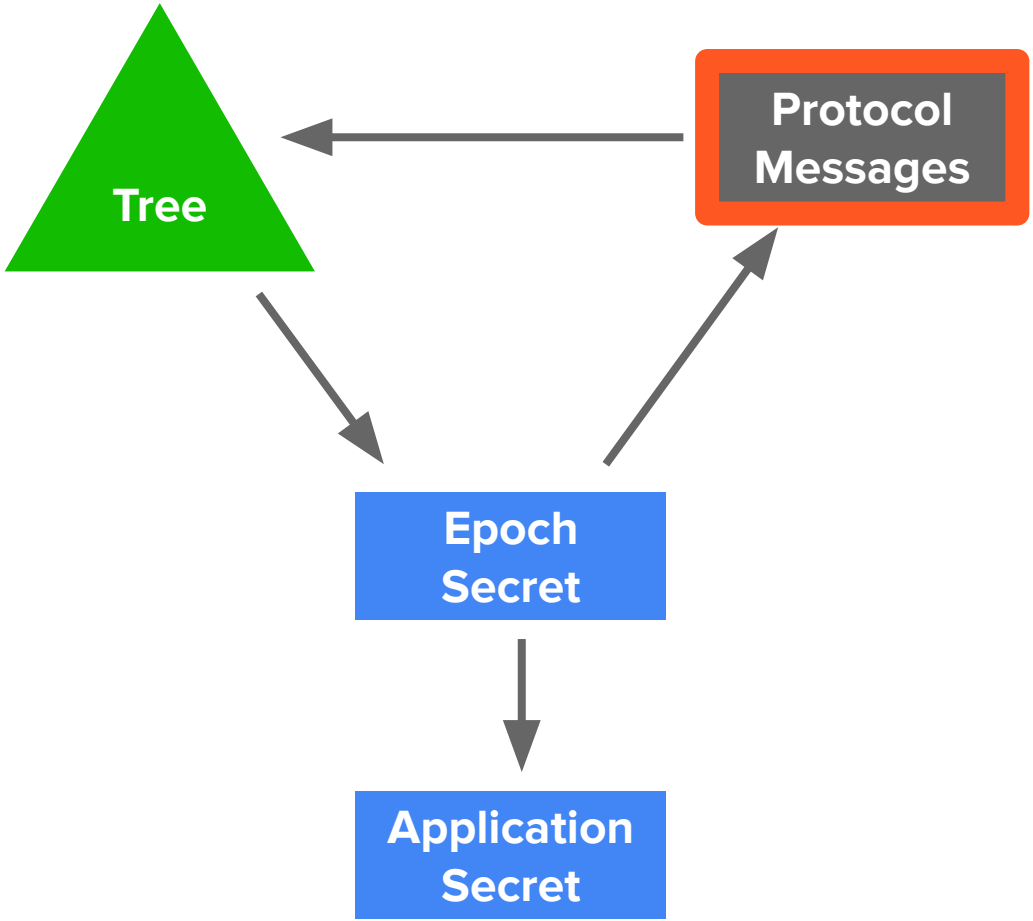
MLS Internals



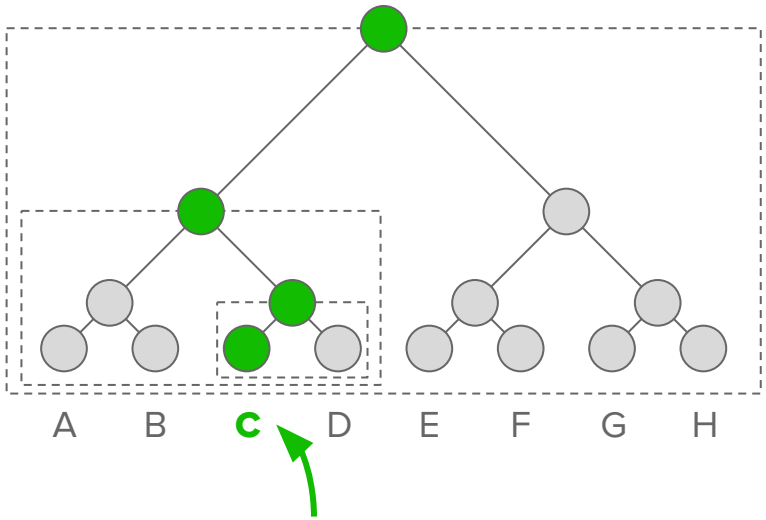
TreeKEM establishes shared secrets that make it easy to send a key to all but one member

The **Key Schedule** combines the secrets from TreeKEM into a common history of the group

The **Authentication** layer attests to sender identities and confirms agreement on the state of the group


Tree
Protocol Messages
Epoch Secret
Application Secret

Trees of Keys



C has private keys for green nodes

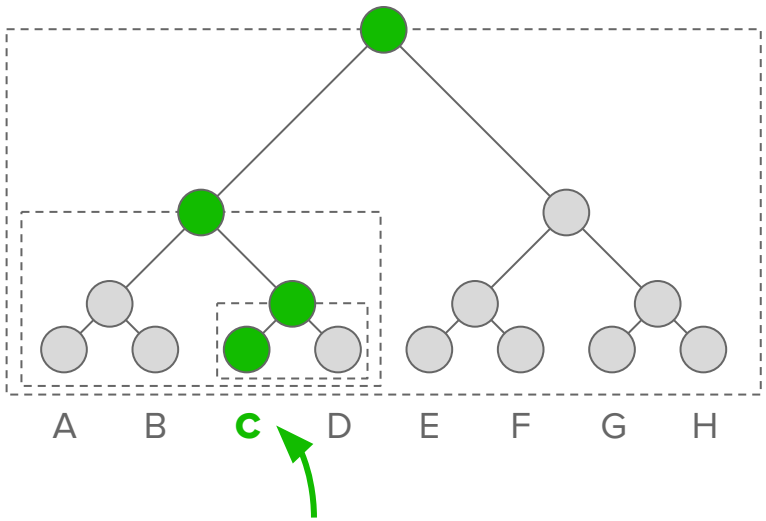
Each node in the tree either contains a key pair or is marked “blank”

Each member “occupies” a leaf node

Private keys follow the **tree invariant**:

> *The private key for a node in the tree is known to a member of the group if and only if that member's leaf is a descendant of the node or equal to it.*

Trees of Keys



C has private keys for green nodes

This has a couple of nice consequences:

- Intermediate nodes represent subgroups you can DH with / encrypt to
- Root private key is a secret shared by the members of the group at a given time

Protocol maintains this state through group operations (**Add**, **Update**, **Remove**)

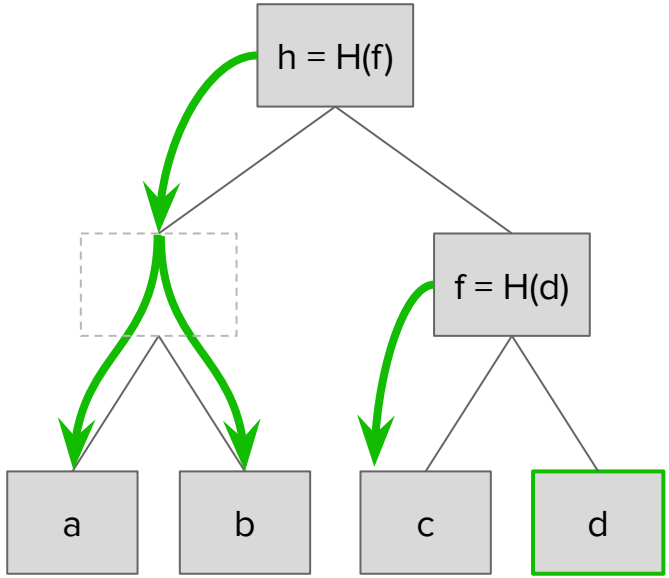
The KEM in TreeKEM

For Update and Remove, you want to send fresh entropy to all but one member

Update: Old version of self
Remove: The evicted member

To simultaneously encrypt new entropy and update the tree in **O(log N)**:

Derive from hashes up the tree
Encrypt the hash to the other child



Protocol Messages Update The Tree

Add:

Add leaf to the tree

Group hashes forward

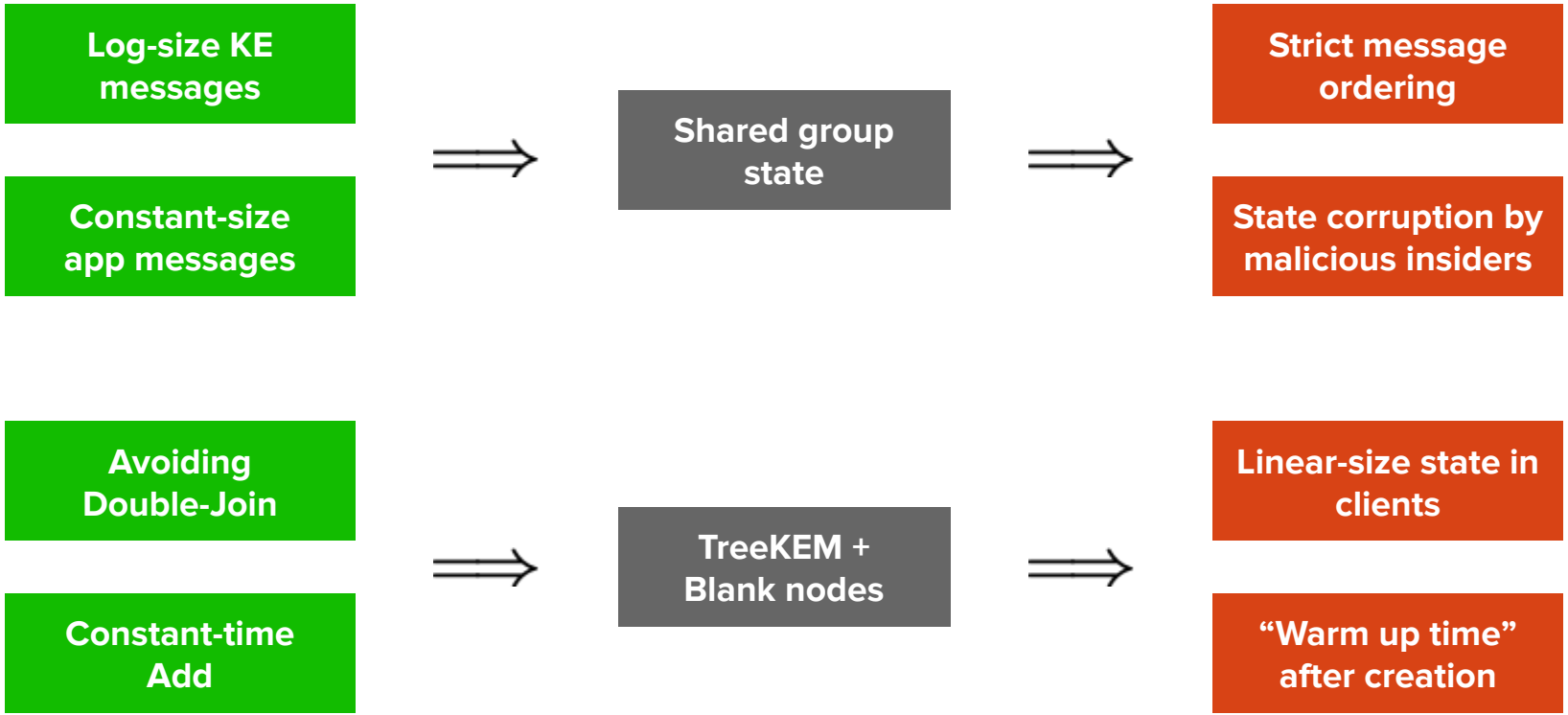
Encrypt secret to new joiner

Remove / Update:

Encrypt fresh entropy to everyone but the evicted participant

Future

Trade-Offs


Log-size KE messages
Constant-size app messages
Shared group state
Strict message ordering
State corruption by malicious insiders
Avoiding Double-Join
Constant-time Add
TreeKEM + Blank nodes
Linear-size state in clients
“Warm up time” after creation

Worry / Wonder / Hope

Worry: Overhead / efficiency

Worry: Malicious insiders

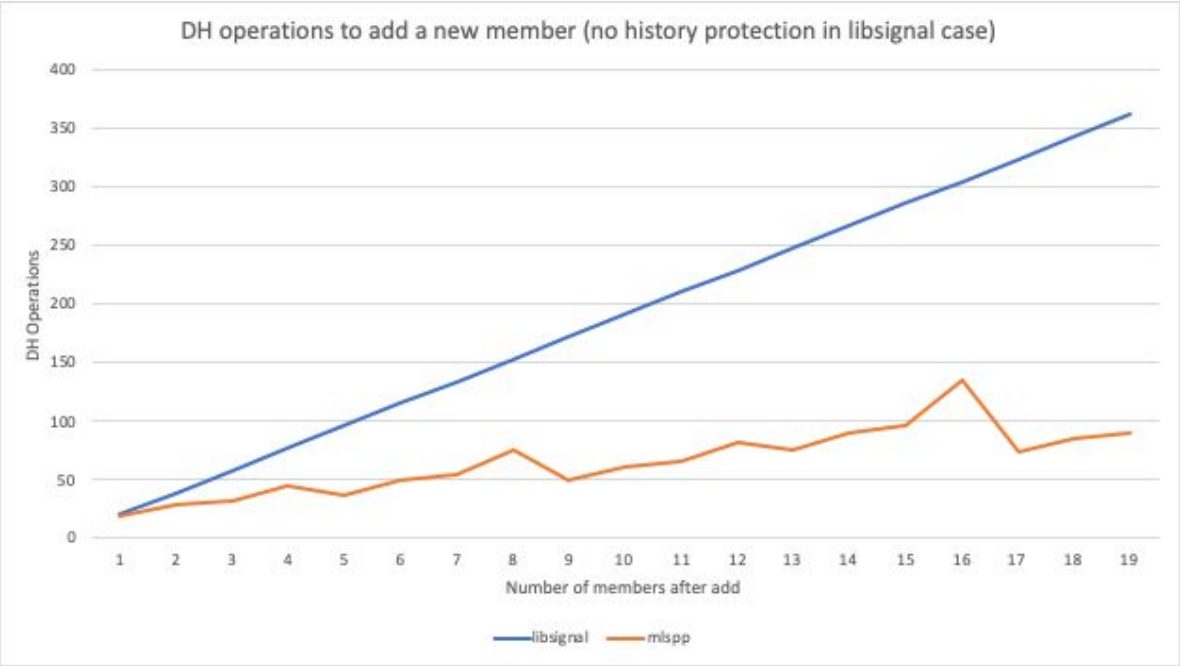
Wonder: Transport interactions

Hopeful: More applications

Hopeful: Grand Unified theory?

Overhead: The Good News

We’re doing better than the state of the art!



K * N
[K = cost of 1-1 channel]

O(N + log N)

Overhead: Less Good News

Group stuff is inherently kind of expensive (signatures!)

For contrast: There is active debate about whether TLS is small enough for IoT

Theoretical minimum TLS handshake: 64 / 177 / 113 bytes

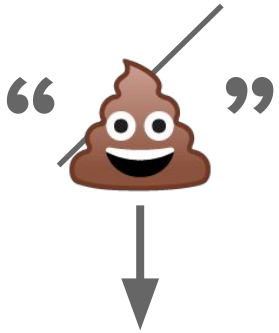
Proposed alternative [EDHOC]: 39 / 120 / 85 bytes

Min MLS Welcome: **205** bytes = DH key + Sig key + 2x 32B secrets + framing

Application message overhead: **125** bytes = Sig + tag + nonce + tag + ...

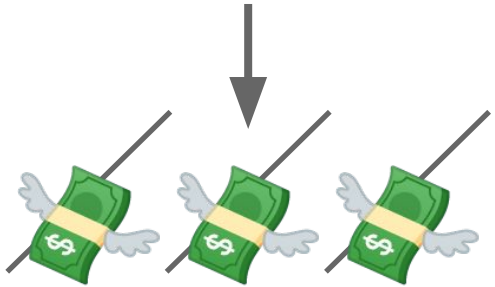
... vs **21** bytes for TLS = Auth tag + framing

4-byte content



101-byte message

```
232df8e2251ec383eda58ca3cc0d2f7d5278f29edafa202a86
5637d7389a3708c463cbad67d9d438ccdb5e117241912172e4
4175c627450a09c8221ede96ad9a649d10f53d368f89b2bc08
4add2af538af3544a4564dddab09a3f9584113c4122c8417a1
9d28a5d4b9ece9a960fa9780f7cee204707b8337f747446081
8866ba0c
```



... millions of users ...

Overhead: Open Questions

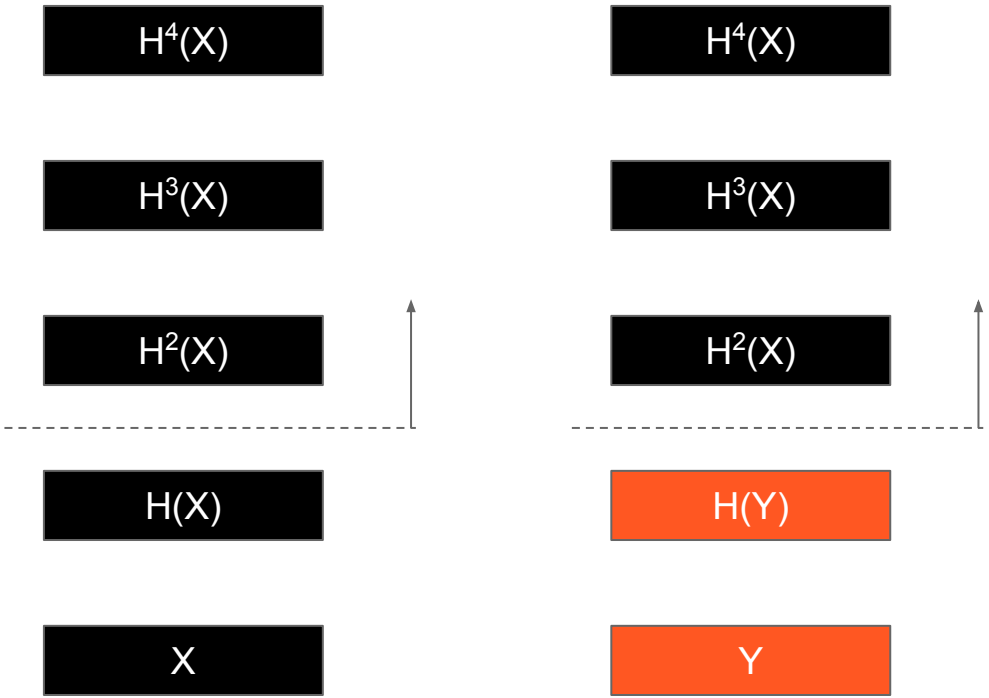
To what degree are the message sizes necessary vs. engineering-induced?

Can the protocol be re-encoded more efficiently?

Is it possible to remove some protections in some cases?

For example, maybe some use cases don’t care about individual attribution of application layer messages

Malicious Insiders: The Problem



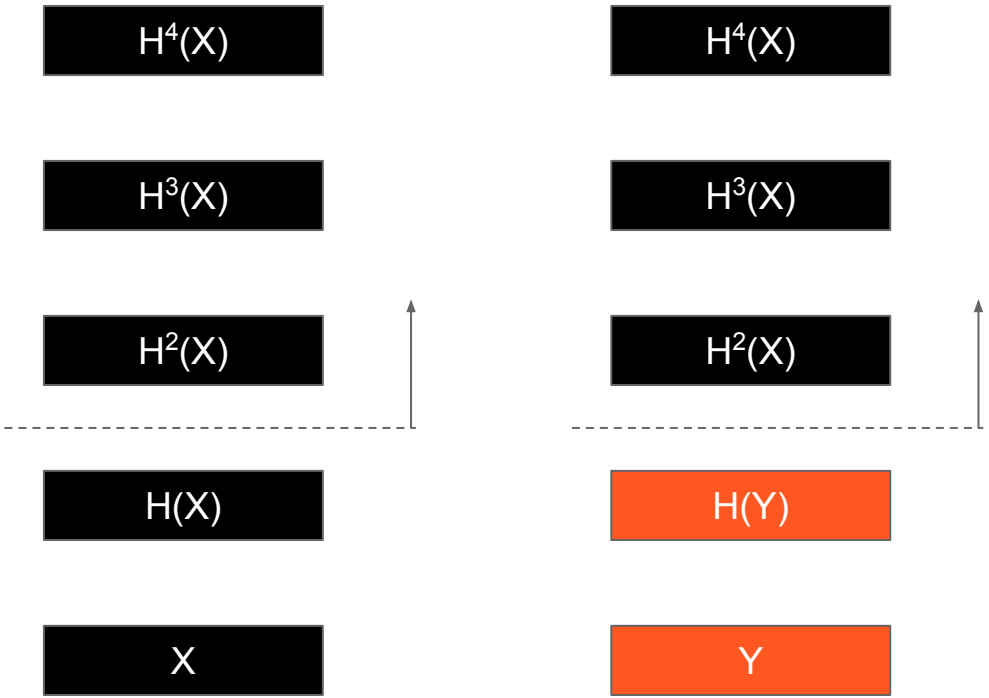
MLS paths are supposed to be the encryption of a hash chain…

But each recipient can only verify from a given point upward

So a malicious user can split the tree

And it will only be visible to a subset of users (here 4 of 32)

Malicious Insiders: Open Questions



Pretty clear at this point that ZKPs aren't feasible (proof?)

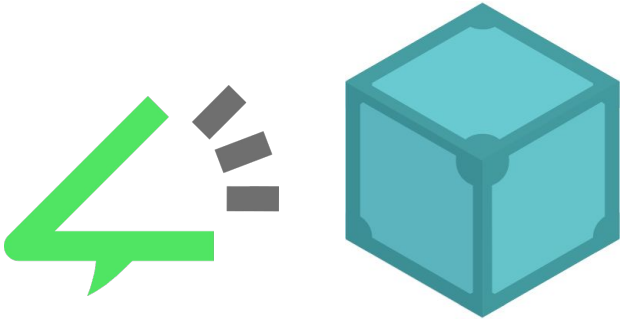
Naïve approach to reporting corruption reveals private information

Can we provide tools to prove corruption without revealing private info?

Transport Interactions

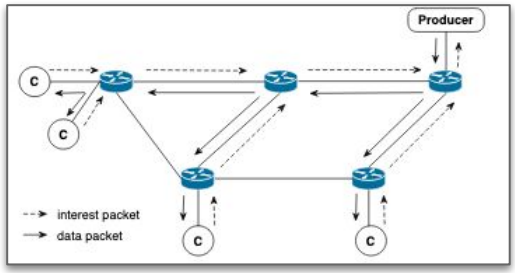
Right now, we basically assume a messaging app's delivery infrastructure

- Synchronized delivery is not a huge problem
- Metadata protection is nice, but server has to know how to do fanout



What about better stuff?

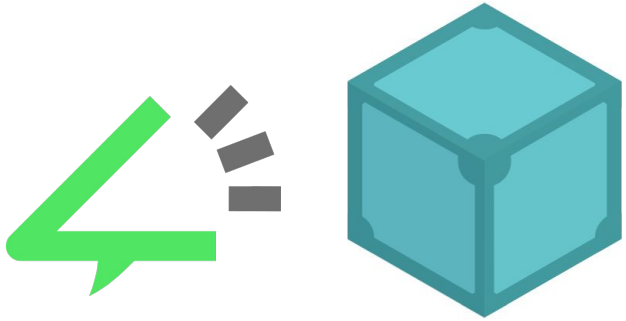
Loopix



Transport Interactions

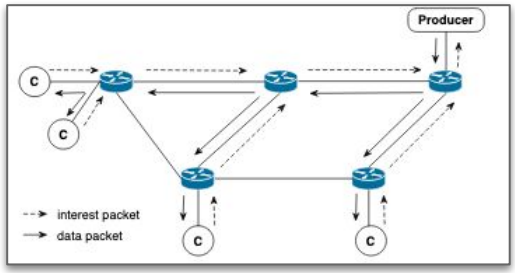
For mixnets / more private transports: Is the metadata protection we have now good enough?

For decentralized transports: How do we accommodate the need for synchronized delivery?



- Decentralized sync protocols?
- Independent histories?
- Allow forking?

Loopix



New Applications

MLS started with secure messaging apps, but it's really just an async group AKE

There are other cases where groups need to establish shared keys

- WebRTC conferencing, IPTV, IoT pub/sub, SD-WAN, …

Even more cases where you an async 2-message 1-1 AKE might be handy

- ESNI, DNS?, HTTP?, …

Grand Unified Theory?

TLS defines a 1-1 AKE

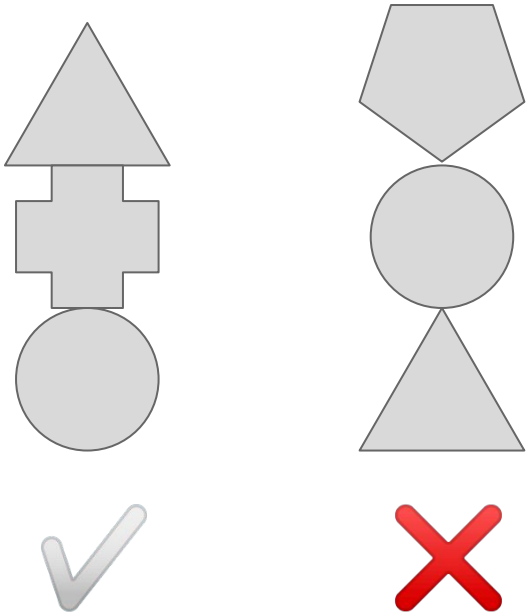
MLS defines a group AKE, thus in particular a 1-1 AKE

Right now, these look rather different

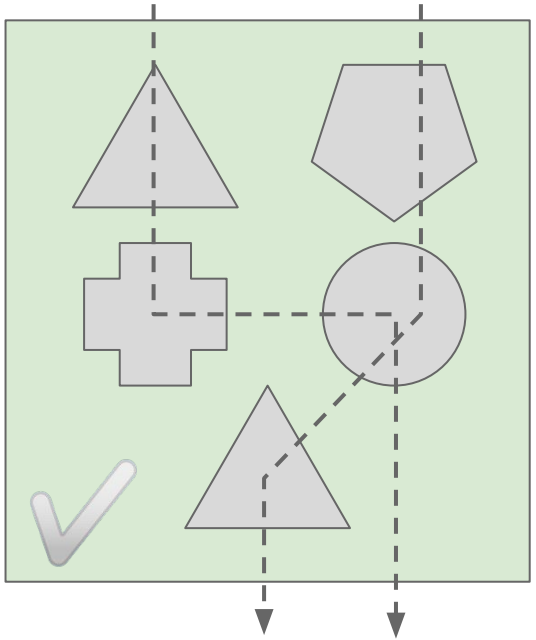
Could we define some unified framework within which both are special cases?

Complementary Approaches to Diversity

Noise: Assemble then Prove



TLS / MLS: Prove then Profile



And beyond...

Past, Present, and Future

Emerged from a blend of practical, industry needs and ideas from academia

Now starting to firm up into a protocol that is usable and provable

… including at least four different implementations!

Still a few meaty problems to be solved before we can call it done

… and even more opportunities over the next horizon


CISCO