MLS @ IETF 103

Slides for RLB slots

Add / Remove without Double-Join

Flow

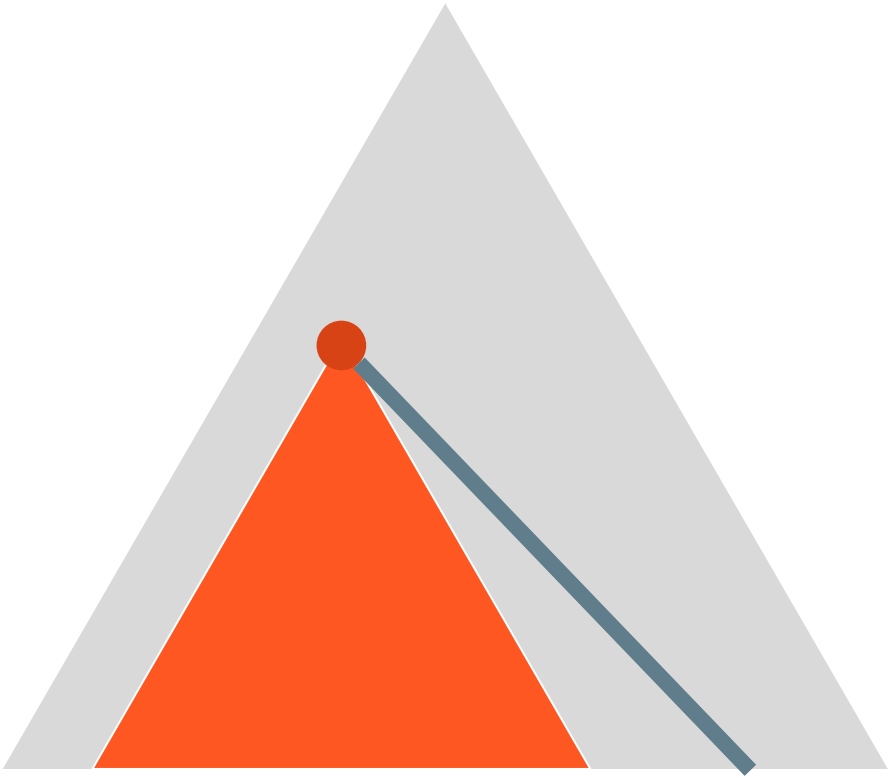
[[The Tree Invariant]]

[[Add w/ Double-Join]]

[[Blanking + Resolution]]

[[Add w/o DJ]]

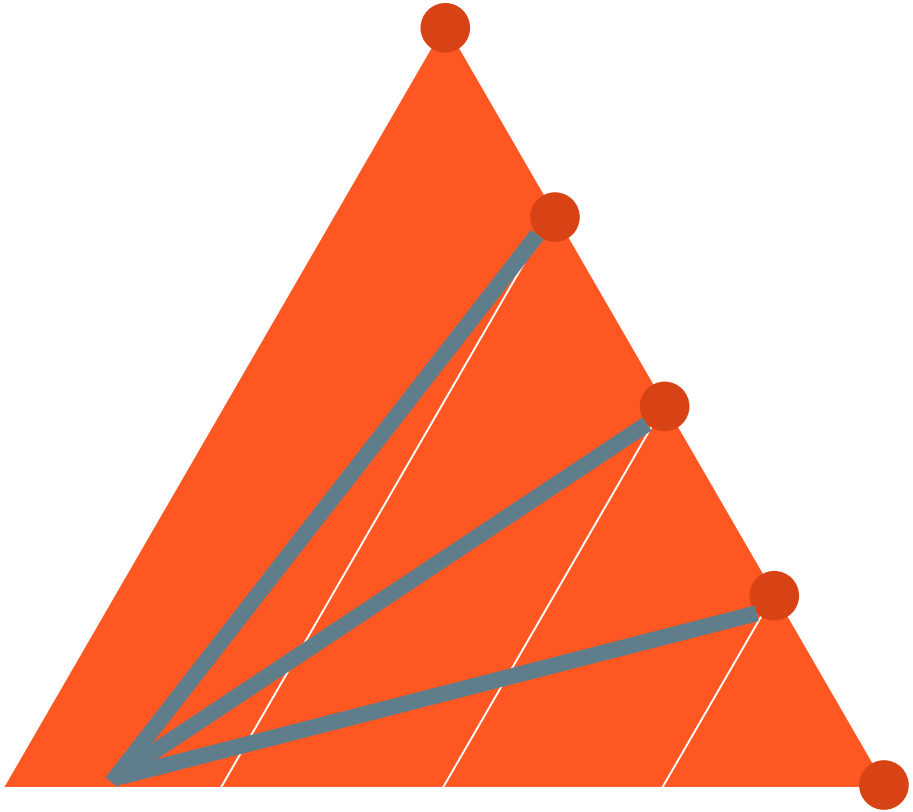
[[Efficiency]]



The Tree Invariant

The private key for a node in the tree shall be known to the descendants of that node, and them alone.

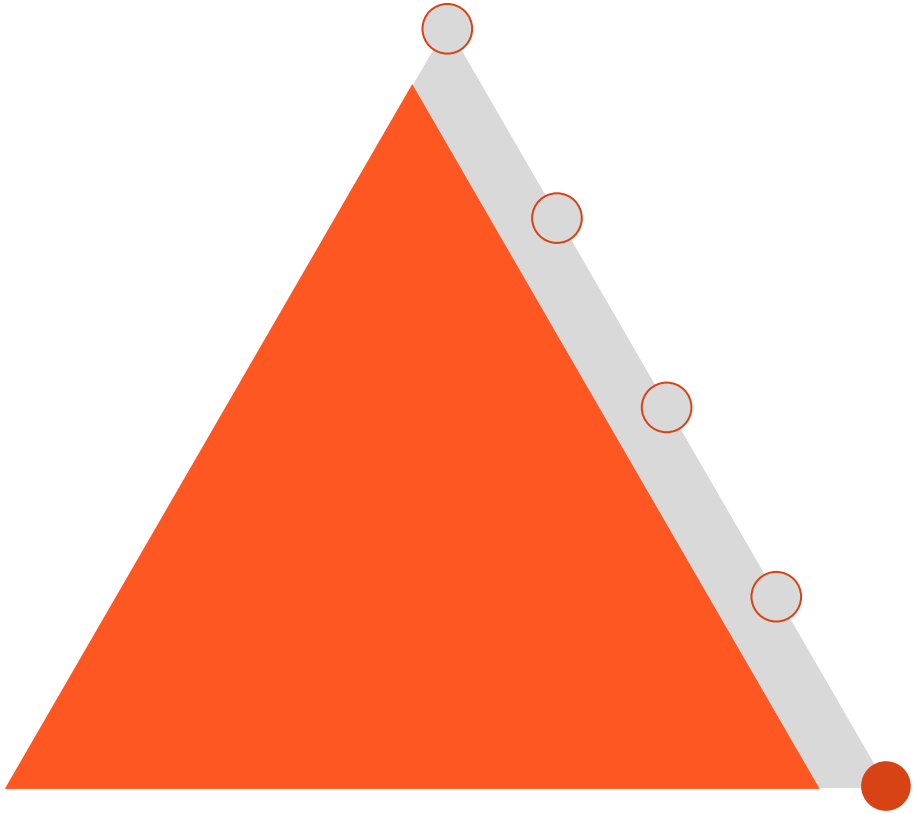
A “double join” is a violation of this invariant.



Add / Rem w/ Double Join

In prior versions, Add and Remove caused double joins

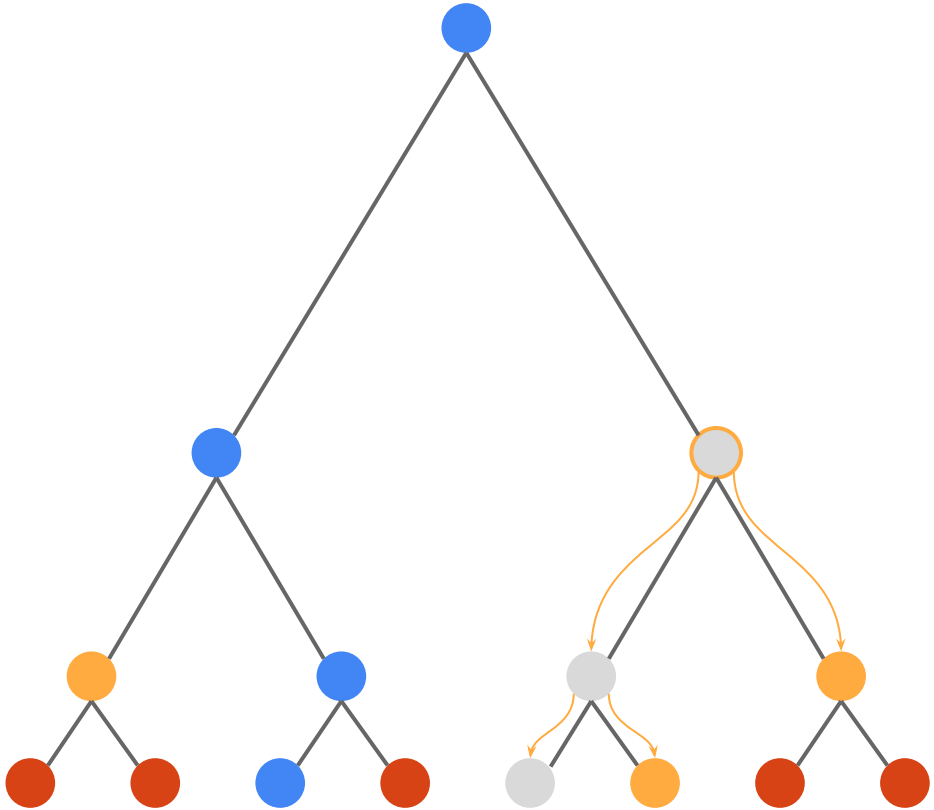
This is because the sender sets the intermediate nodes



No More Double Joins

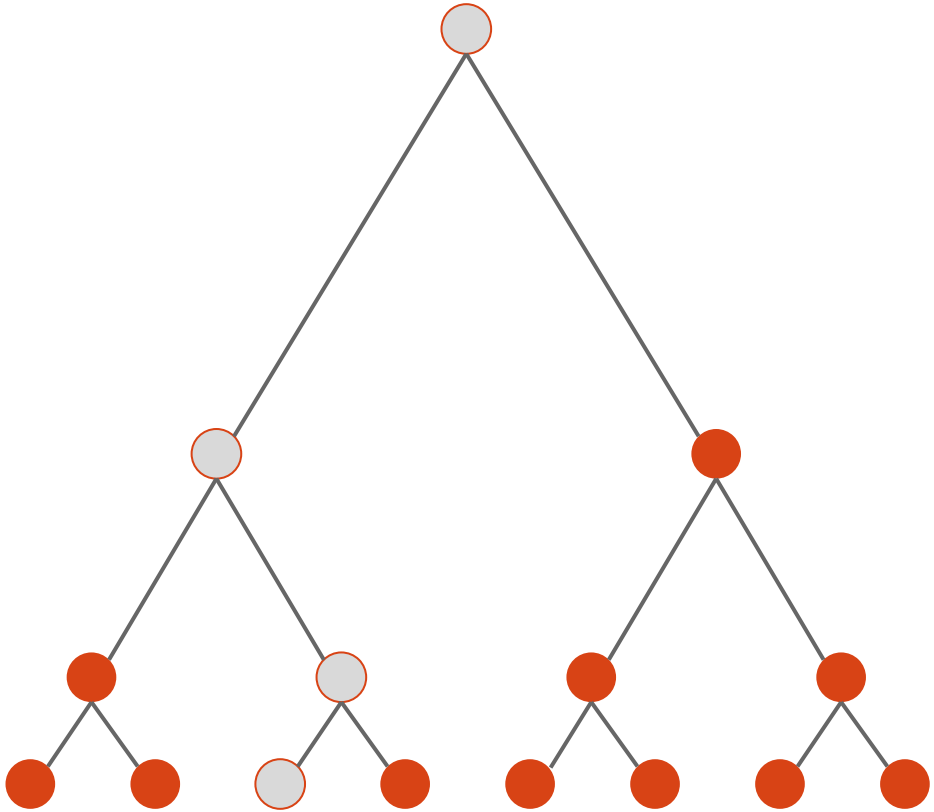
Allow nodes to be blank

Instead of setting to a double-joined value, leave it blank



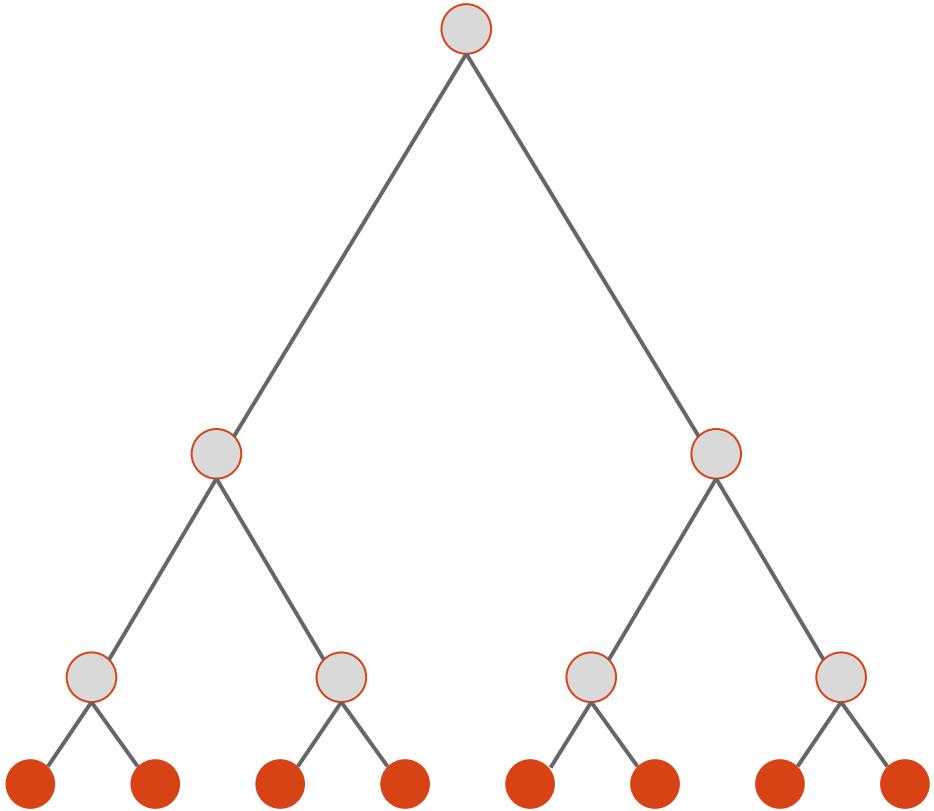
Resolution

When you want to send an update and you would encrypt to a blank node, you instead encrypt to its populated descendants



Remove

To remove a node, just blank out its direct path



Init

To set up a new tree, just put the members' DH public keys (from UserInitKey) in the leaves

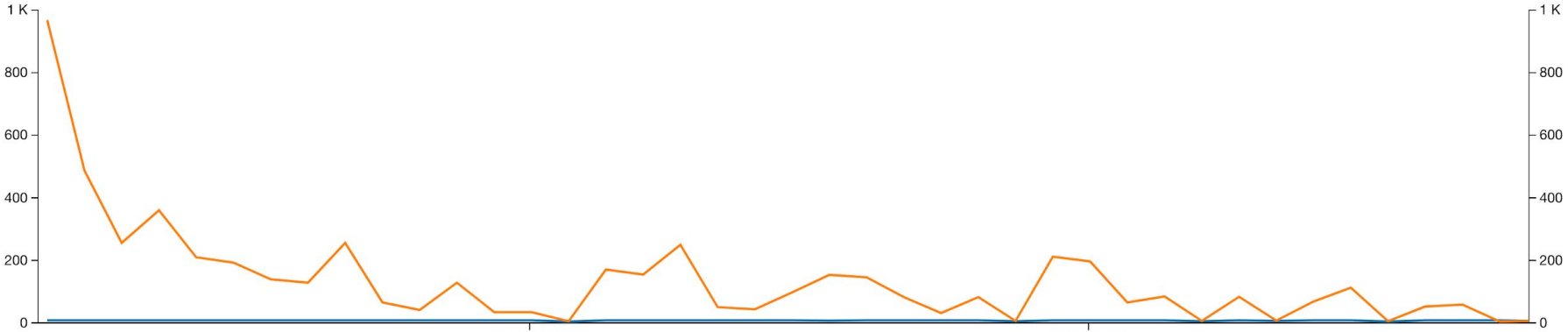
The first update is linear

Efficiency

Fragmented trees lead to worse-than-log-size operations

In particular, on Init, there's a "warm up" phase O(N) -> O(log N)

Simulating a 1000-member group doing random operations...



Key Confirmation

Basically SIGMA

draft-01 guaranteed that:

If two parties arrive at **different rosters**...

... then they arrive at **different keys**

The only way to realize you had different keys was message decryption failure

draft-02 adds a key confirmation MAC

If processing of the HS message succeeds...

...then the sender and receiver have the same view of the roster

```
struct {
    uint32 prior_epoch;
    GroupOperation operation;

    uint32 signer_index;
    SignatureScheme algorithm;
    opaque signature<1..2^16-1>;
    opaque confirmation[Hash.length];
} Handshake;
```

To MAC or not to MAC

Two parallel PRs:

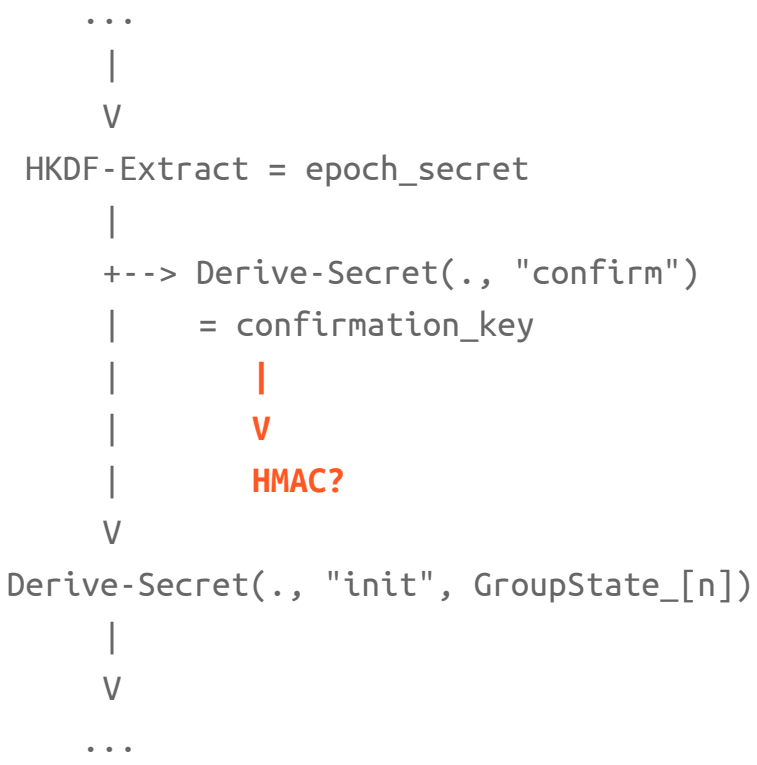
https://github.com/mlswg/mls-protocol/pull/71

https://github.com/mlswg/mls-protocol/pull/72

Option 1: Derive a value from the key schedule and publish it in the HS message

Option 2: Derive a value from the key schedule and publish a MAC with it in the HS message

But HKDF already uses HMAC!



Efficiency vs. Confidentiality

Two Questions

1. Do we want to allow out-of-band roster / tree distribution?

2. Should we expose information to the server that allows it to passively cache roster / tree information?

Send by commit instead of by value

```
struct {
  opaque group_id<0..255>;
  uint32 epoch;
  Credential roster<1..2^32-1>;
  PublicKey tree<1..2^32-1>;
  opaque transcript_hash<0..255>;
  opaque init_secret<0..255>;
} Welcome;
```

```
struct {
  opaque group_id<0..255>;
  uint32 epoch;
  opaque roster_hash<0..255>;
  opaque tree_hash<0..255>;
  opaque transcript_hash<0..255>;
  opaque init_secret<0..255>;
} Welcome;
```

Assumes OOB distribution of roster, key

Could be server-based or client-based (e.g., encrypted Roster / Tree messages)

Expose information for server assist

The only way to avoid a linear-size upload is for the server to cache the roster / tree info gleaned from HS messages in transit

Tree => Public keys for tree nodes*

Roster => Identities / credentials*

Both => Basically no HS encryption

Two modes?

- O(N) Welcome + Full HS encryption
- O(1) Welcome + No HS encryption

```
struct {
  uint32 prior_epoch;
  GroupOperation operation {
    Add{ DH, cred, sig },
    Update{ path },
    Remove{ index, path }
  }
  uint32 signer_index;
  SignatureScheme algorithm;
  opaque signature<1..2^16-1>;
  opaque confirmation[Hash.length];
} Handshake;
```

* Assuming no composable encryption scheme