

MLS PROTOCOL

draft-ietf-mls-protocol-08

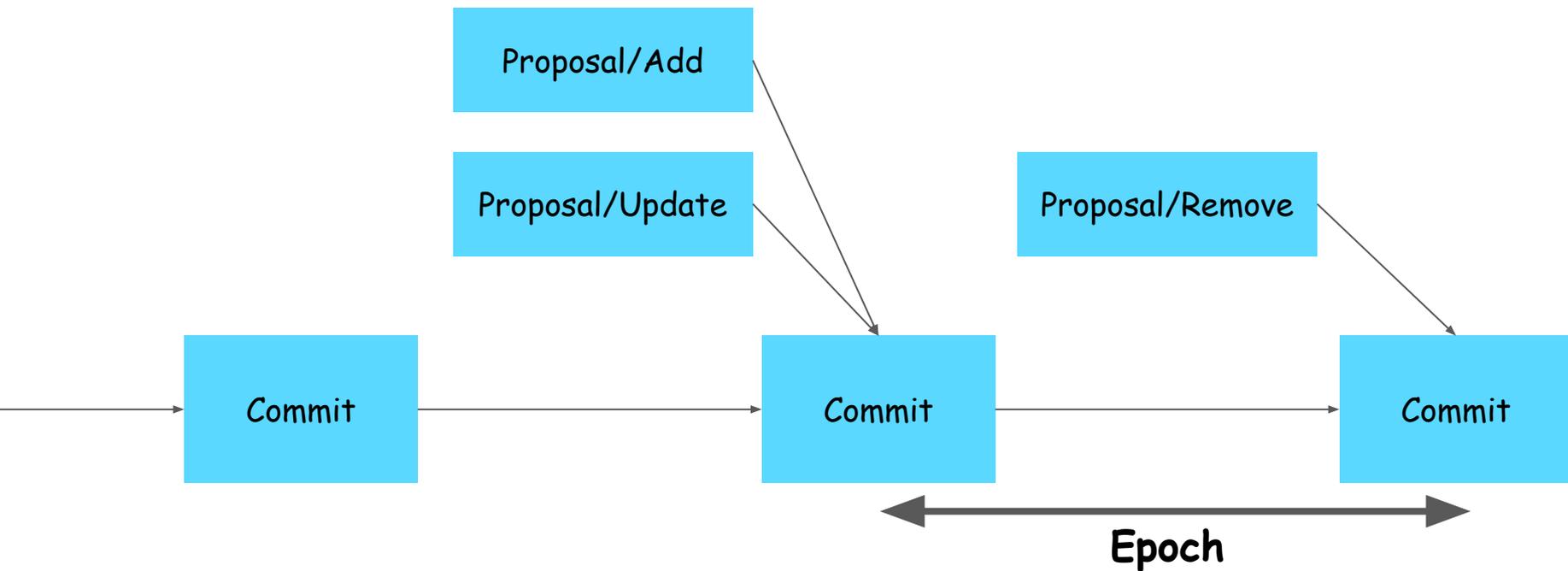
Richard Barnes, Raphael Robert,
Benjamin Beurdouche

THINGS TO COVER

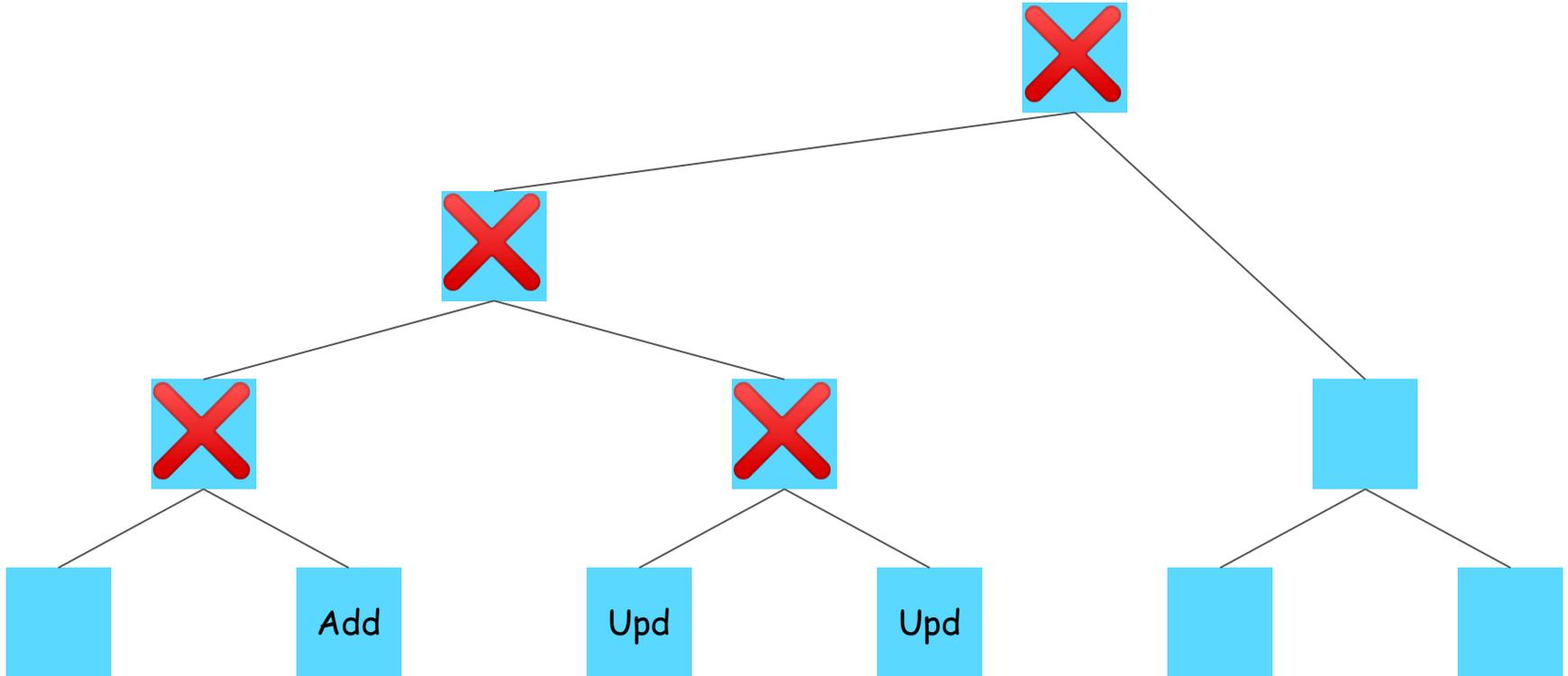
- Changes since interim, especially:
 - Proposals
 - Unified Init / Welcome
 - Downgrade protection
 - External proposals
- Remaining issues, especially:
 - Handshake encryption
 - Signing key roll-over
 - Send-to-group-from-outside
- Discussion of performance metrics / simulation

CHANGES SINCE INTERIM

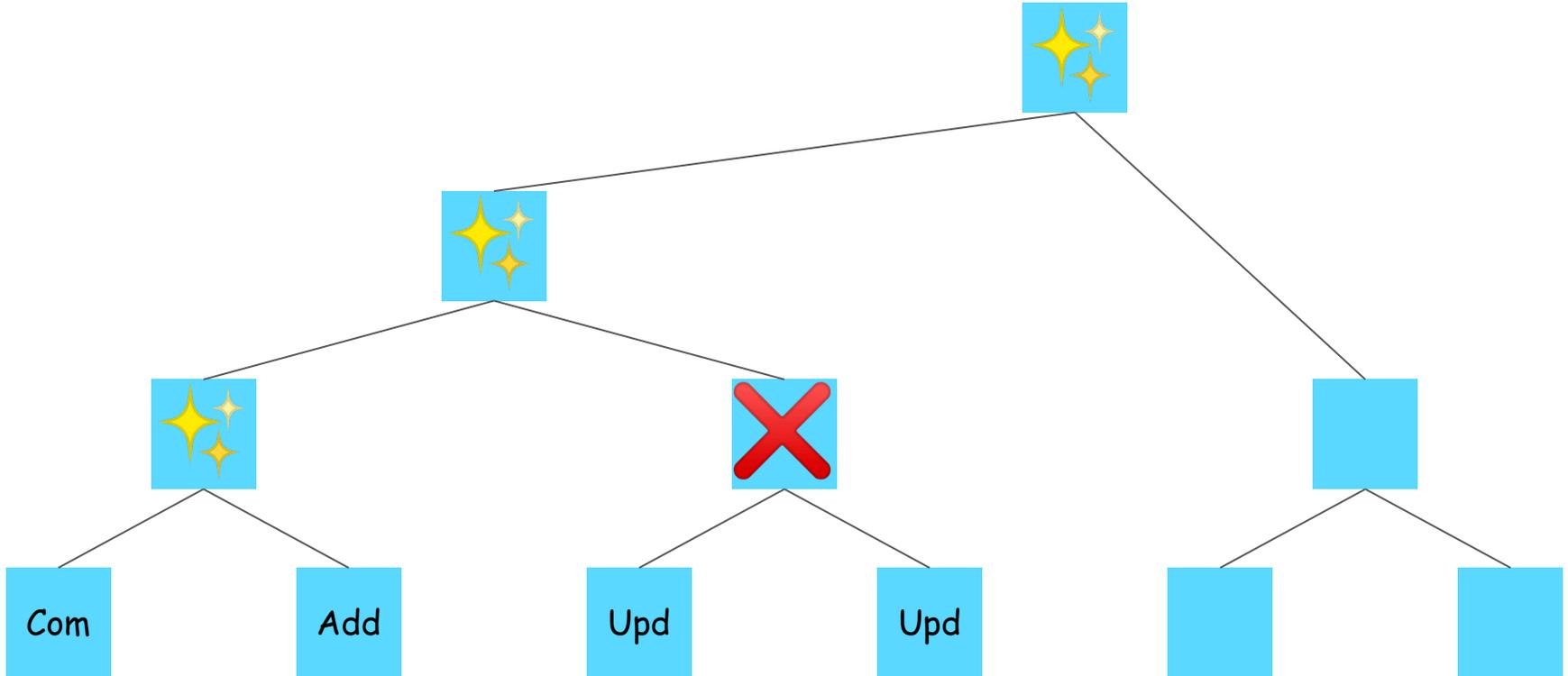
PROPOSAL / COMMIT



PROPOSALS DEGRADE THE TREE



COMMITS REBUILD THE TREE



UNIFIED INIT / WELCOME

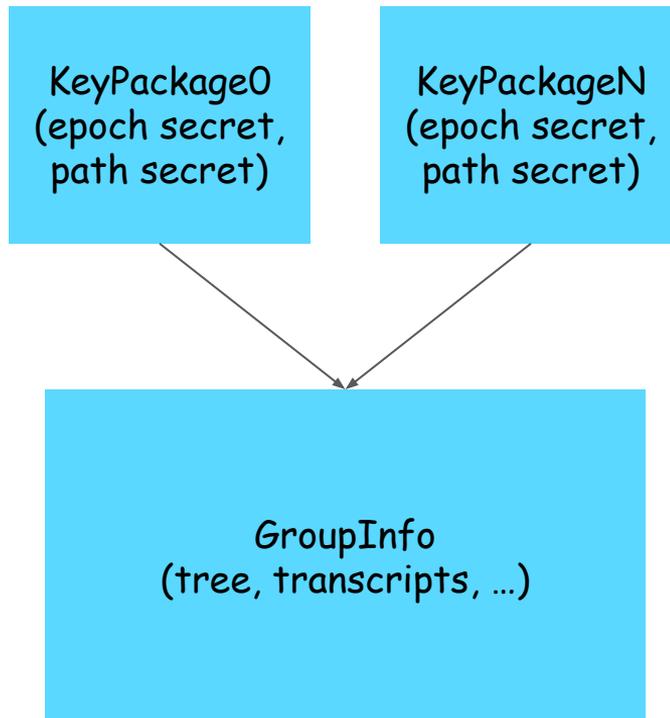
With Proposals, multiple Adds per epoch

=> Make Welcome support multiple users

=> Init = Welcome*

Group Creation:

- Creator makes a one-member group
- ... sends itself N add proposals + commit
- ... sends resulting Welcome (= Init) to new members



EXTERNAL PROPOSALS

Add and Remove can be generated by people not in the group

Signal this with special values in `MLSPlaintext.sender`

`0xFFFFFFFF` = Self-add, signed by key in CIK

User-initiated add!

`0xFFFFFFFF00` - `0xFFFFFFFFFE` = pre-configured

Server-initiated
remove!

Server-initiated add
... almost!

DOWNGRADE PROTECTION / CIK EXPIRATION

ClientInitKeys now have a single version, ciphersuite

Problem: CIK server can downgrade by not providing CIKs for good values

Solution: Extensions! supported_versions and supported_ciphersuites

Group creator **MUST** verify it has CIKs for best options

While we're at it ... expiration = seconds since epoch

CHANGES SINCE DRAFT-08 (OOPS)

FURTHER IMPROVEMENT TO WELCOME (#247)

```
update_secret -> HKDF-Extract = epoch_secret
```

```
+ |  
+ +--> HKDF-Expand(., "mls 1.0 welcome", Hash.length)  
+ | = welcome_secret  
+ |
```

Don't require sender to generate keys

```
+ opaque confirmation<0..255>;  
  uint32 signer_index;  
  opaque signature<0..255>;  
} GroupInfo;
```

All recipient to verify correct processing

PROPOSALIDS ARE BUSTED (#246)

```
select (MLSPlaintext.content_type) {
  case application:
    opaque application_data<0..2^32-1>;

  case proposal:
    Proposal proposals<1..2^32-1>;

  case commit:
    Proposal proposals<1..2^32-1>;
    Commit commit;
    opaque confirmation<0..255>;
}
```

```
struct {
  uint32 sender;
  opaque hash<0..255>;
} ProposalID;
```



PROPOSALIDS ARE BUSTED (#246)

Current

```
struct {  
    Proposal proposal<*>;  
    Commit commit;  
    opaque signature;  
} MLSPlaintext;
```

One Entry

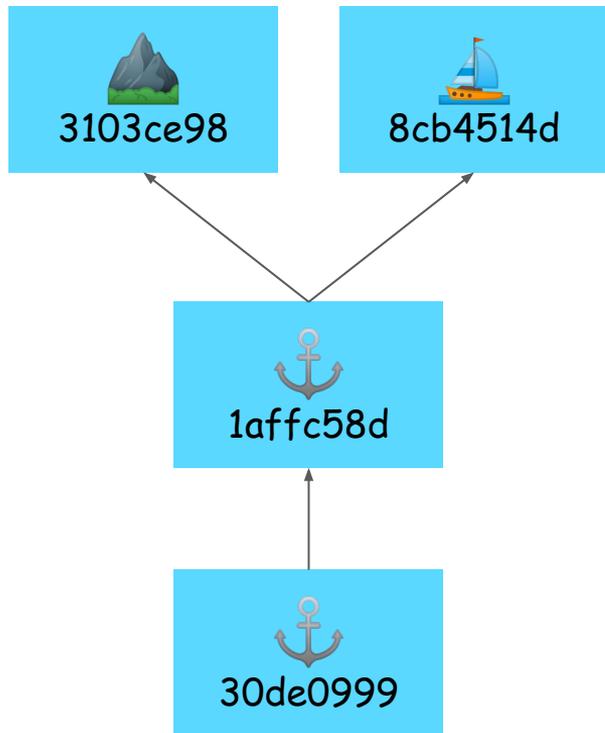
```
struct {  
    select {app/prop/comm}  
    opaque signature;  
} MLSPlaintext;
```

Generic Framing

```
struct {  
    select {app/prop/comm}  
} MLSPtContent;  
  
struct {  
    MLSPtContent vals<*>;  
    opaque signature;  
} MLSPlaintext;
```

Amortization of signature
vs.
Complexity / !(key separation) (?)

UNPREDICTABLE EPOCHS (#245)



Decentralized apps (e.g. Matrix) need to tolerate forks in group history

Right now, epoch is a linear counter

Proposal: epoch pseudo-random, derived from group state

Small enough to not bloat messages

Large enough to avoid collisions

Concerns about server being able to keep up

YET TO COME...

MLS EXPORTER (#198)

Use MLS to generate group secrets needed for other applications: SRTP, encrypted backup... (Same mechanism as TLS)

```
1204      +--> Derive-Secret(., "app", GroupContext_[n])
1205      |   = application_secret
1206      |
1207 +     +--> Derive-Secret(., "exporter", GroupContext_[n])
1208 +     |   = exporter_secret
1209 +     |
1210      +--> Derive-Secret(., "confirm", GroupContext_[n])
1211      |   = confirmation_key
1212      |
```

MLS EXPORTER (#198)

Use MLS to generate group secrets needed for other applications: SRTP, encrypted backup... (Same mechanism as TLS)

```
1273 + ## Exporters
1274 +
1275 + The main MLS key schedule provides an `exporter_secret` which can
1276 + be used by an application as the basis to derive new secrets called
1277 + `exported_value` outside the MLS layer.
1278 +
1279 + ~~~~~
1280 + MLS-Exporter(Label, Context, key_length) =
1281 +     HKDF-Expand-Label(Derive-Secret(exporter_secret, Label),
1282 +         "exporter", Hash(Context), key_length)
1283 + ~~~~~
1284 +
```

TREE OF SIGNATURES (#253)

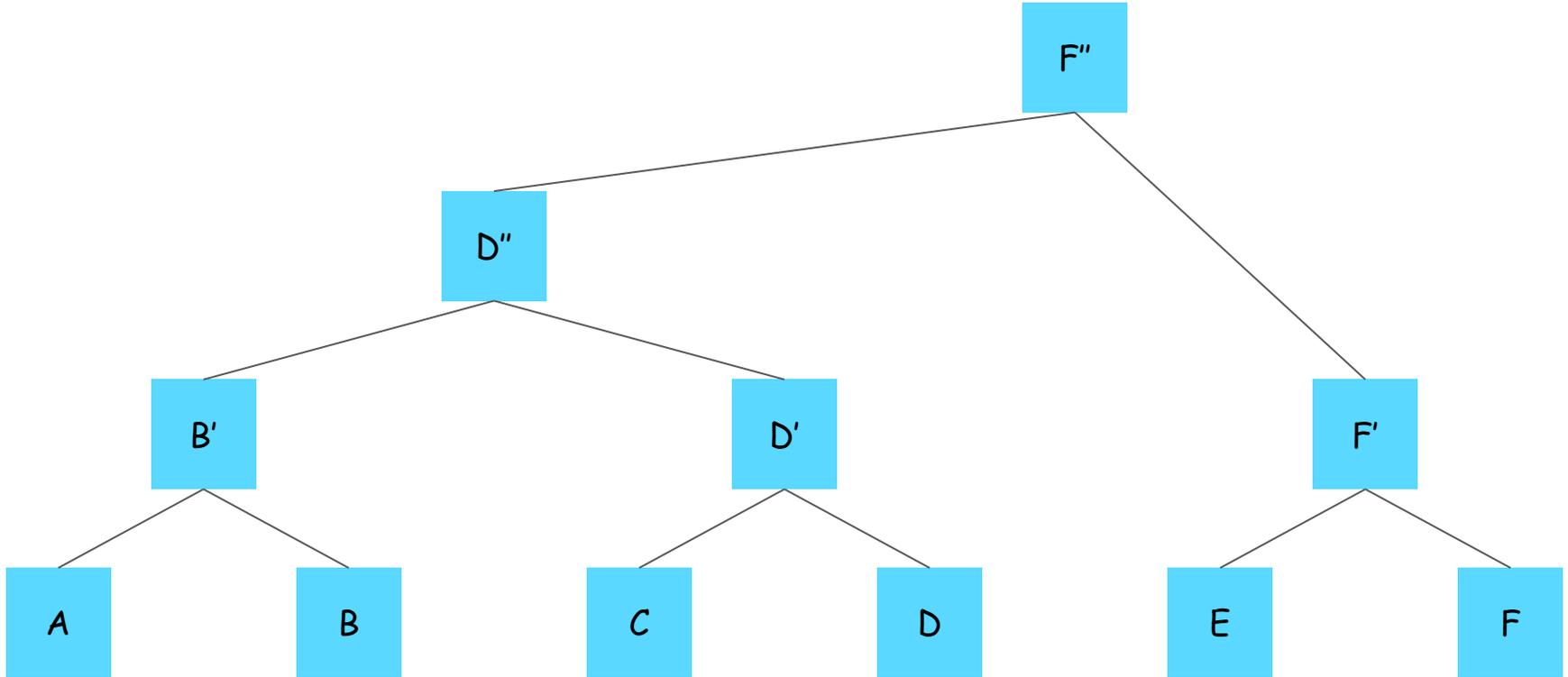
The newcomers need more than the credentials...

For now, a newcomer has to completely trust the sender of the welcome package to provide an honest public tree.

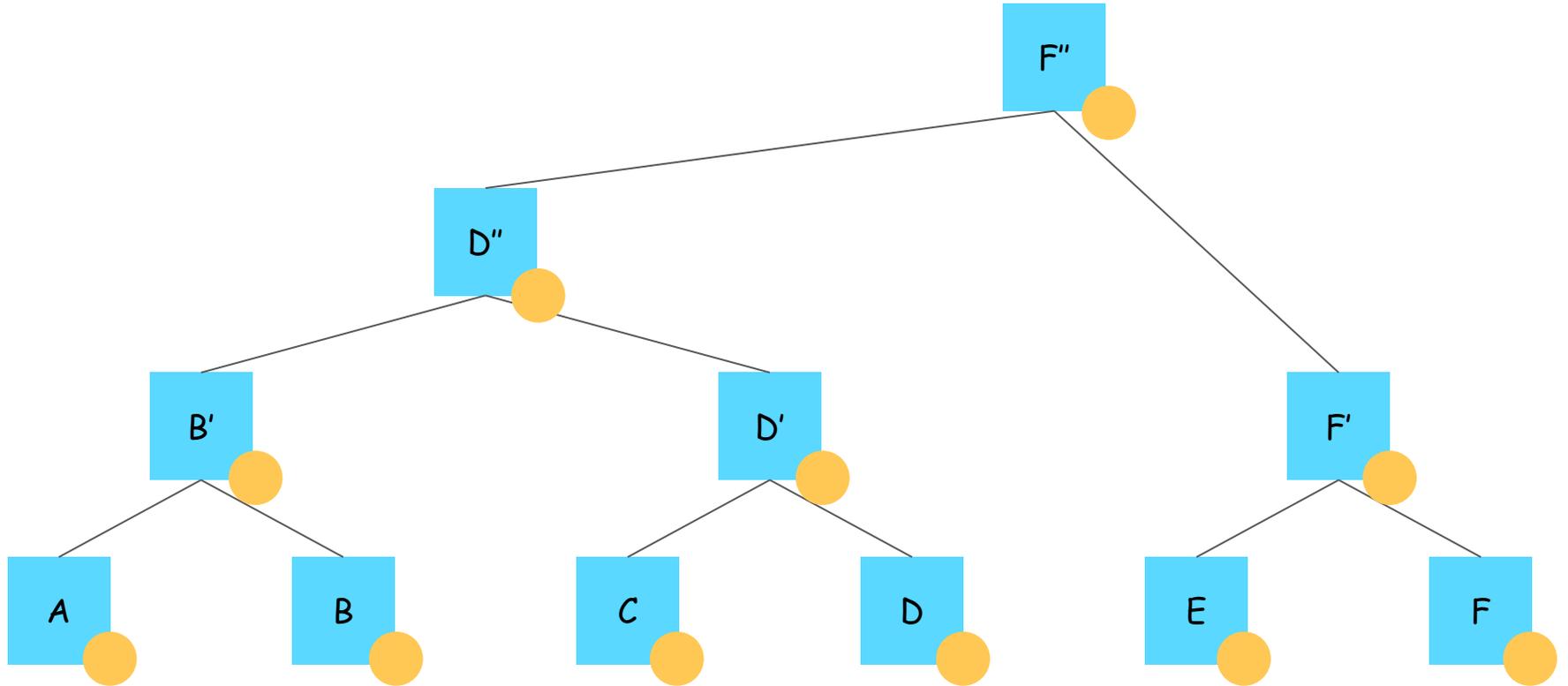
Until the newcomer see everyone update, there are not much guarantees.

If each node of the tree is signed, the newcomer can figure out the influence of the current members over the current public state of the group

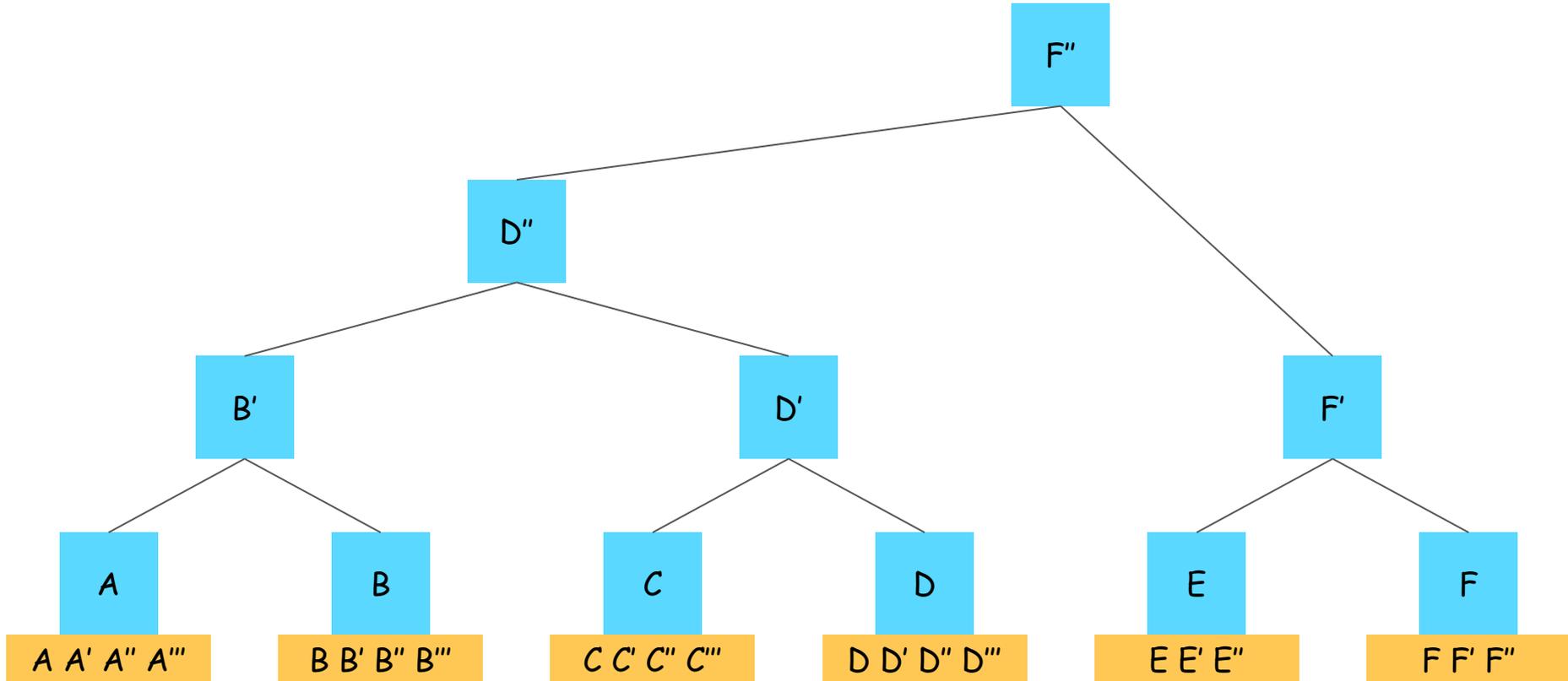
TREE OF SIGNATURES (#253)



OPTION #1: SIGNATURES ON NODES



OPTION #2: LEAF SIGNATURES + HASHES



TREE SIGNING OPTIONS

	Sig ops	Tree size overhead
Option #1: Signatures on nodes	$\log N$	$N * \text{sig}$
Option #2: Signatures on leaves	1	$N \log N * \text{hash}$
Option #3: YOLO*	0	0

* don't defend against malicious adders

PSK (#251)

Inject a PSK into the Key schedule: from an expensive PQ group key exchange, static PSKs...

```
1196 + PSK (or 0) -> HKDF-Extract = early_secret
1197 +           |
1198 +           Derive-Secret(., "derived", "")
1199 +           |
1200 +           v
1201 + update_secret -> HKDF-Extract = epoch_secret
```

This case leaves the application to handle everything. This might be too easy for the applications to screw up... so there is an alternative...

PSK (#25) ALTERNATIVE

Inject a PSK into the Key schedule: from an expensive PQ group key exchange, static PSKs...

Define a new Group Operation called PSK which contains a PSK identifier and Extract with the PSK instead of an update to get the Epoch secret.

I prefer that, and this might actually be better...

-> Discuss and assign to Ben

RLB: cf.
draft-jhoyla-tls-extended-key-schedule

CLIENT INIT KEYS IN LEAVES (#254)

The newcomers need more than the credentials, they need the CIKs from other members so that they can check their identity and the content of the leaves.

- Gives you an HPKE public key blessed by the signature key
- Allows newcomer to look at other members extension, PV, CS...

Because the CIK contains a signature, it fits well the signature tree story. Each time the Leaf KEM key is updated, the signature is updated.

This will allow Signature Key rotation.

OTHER STUFFF ! (FOR LATER..)

Putting the Client Init Keys in the Leaf will allow to do Signature Key rotation.

Splitting Identity Key and Signature Key will allow more privacy and a better PCS story if we can rotate the signature keys. The compromise scenario of the sig keys can be very different that the compromise of the long-term identity key which might be stored in an HSM or crypto token or whatever...

More privacy stuff...

MEASUREMENT / SIMULATION

Webex Teams (anec)data in "transcript" form

... sequence of Add / Remove / Message

Enough verisimilitude?

Other providers?

What to measure?

- DH ops (fixed / variable)
- Signature ops
- Encryptions / MACs / digests?

What protocol things do we want to look at?

- Commit schedules
- Operational rules (e.g., update-on-join)
- Proposal / Commit / App data packing

**Volunteers to run simulations /
collect data / analyze data?**