Additional Credentials

draft-barnes-mls-addl-creds
### Additional Credential Types

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Recommended</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0003</td>
<td>userinfo-vc</td>
<td>Y</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>0x0004</td>
<td>multi</td>
<td>Y</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>0x0005</td>
<td>weak-multi</td>
<td>Y</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

- **Credential A**
  - Cred Sig Key A
- **Credential B**
  - Cred Sig Key B
- **Credential C**
  - Cred Sig Key C
  - Sig Key

### Verifiable Credentials + Multi-credentials

Successful adoption call post-IETF-117, just needs adoption post-recharter (!!!)
A question of location

This is a separate document for now.

But the rechartering is going slooooooooow

It’s not that much text.

Just chuck it in draft-ietf-mls-extension
Just like Update but...

struct {
  +    uint32 replaced;
    LeafNode leaf_node;
-} Update;
+} Replace;

The affected member is explicit, not inferred from the framing

=> Someone else can make the proposal
Someone else can make the proposal...

Good: Faster PCS

Right now, missed updates have to get thrown away (unlike Add/Remove)

A committer can “re-originate” an Update using Replace (like Add/Remove)

Less good: “Rollback” attacks

Malicious insider compromises secrets associated to an old Update

... then commits a Replace rolling the victim back to the compromised leaf
3. Leaf Node Epoch Extension

The `leaf_node_epoch` extension simply describes the epoch in which a LeafNode was created:

```c
uint64 leaf_node_epoch;
```

*Figure 1: Content of the `leaf_node_epoch` extension*
Questions

Functionality seems interesting?

Security risk seems manageable?

Separate doc or -mls-extensions?
Light Clients

draft-kiefer-mls-light
Problem: Download and Memory

MLS requires that clients download, validate, and store the group’s ratchet tree.

Each participant only uses one MLSCiphertext in a Commit.

... but has to download $O(N)$ data to verify the Commit signature.

In large groups, these objects can be LARGE.

In a 1,000-participant Webex meeting (empty tree):

Tree: 3.5MB in memory / 2.3MB gzip’ed

Commit: 391KB
Light Clients

A **light client** is a member of the group that **does not have the ratchet tree**

A light client **cannot commit**

A light client **cannot process a normal Commit**

Instead:

- Light client joins with only a Welcome
- DS transforms Commit into per-light-client LightCommit

**A light client can join and follow the group with O(log N) download / memory**
Corollaries

Each group must have at least one full (non-light) client

Someone has to do the commits, otherwise nobody can be added!

Clients can transition between light and full

- Light -> Full: Download and validate the tree
- Full -> Light: Delete local copy of the tree

DS needs to be aware of which clients are light / full
Operating a Group with Light Clients

- Alice (full)
- Bob (light)
- Charlie (light)
- Diana (full)
- Emma (full)

- Commit(Add)
- Welcome
- CommitC
- CommitD
- Commit(Add)
Incremental Authentication

Sometimes a light client wants to authenticate a specific member

- Verify their own membership
- Verify the client that added them
- Verify a specific other client (e.g., active speaker)

Present a “slice” through the ratchet tree – basically a Merkle tree proof chaining to the tree hash
Summary

Ratchet Tree and Commits are heavy in large groups

Light clients can join and follow with $O(\log N)$ download and memory

... at the cost of not being able to authenticate the whole group

Three main changes:

- Skip the tree validation on join
- DS slices Commit into per-client Light Commits
- Tree slices allow for authentication of specific other members