MLS Protocol

draft-ietf-mls-protocol-10

Things to cover

- Highlights of draft-10
- WGLC issues
  - More robust tree hashing
  - Signing PublicGroupState
- Other issues from GitHub

Highlights of draft-10

Better MLSPlaintext authentication

membership_tag so that you always verify membership before anything else

confirmation_tag covers signature as well as message content (cf. Finished)

```
struct {
  /* group_id, epoch, sender, auth_data */

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

       case proposal:
       Proposal proposal;

       case commit:
       Commit commit;
       opaque confirmation<0..255>;
  }

  opaque signature<0..2^16-1>;
} MLSPlaintext;
```

```
struct {
  /* group_id, epoch, sender, auth_data */

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

       case proposal:
       Proposal proposal;

       case commit:
       Commit commit;
  }

  opaque signature<0..2^16-1>;
  optional<MAC> confirmation_tag; // iff Commit
  optional<MAC> membership_tag;
} MLSPlaintext;
```

Pre-shared Keys

PreSharedKey input to the key schedule allows multiple PSKs per epoch

PSKs are proposed in Proposals, signaled in Welcome

Learnings from TLS 1.3+:

- Proper key separation through structured labels and nonces
- Combination of multiple PSKs


```
psk_input_[i]    = KDF.Extract(0, psk_[i])
psk_secret_[i]   = ExpandWithLabel(psk_input_[i], ...)
psk_secret       = psk_secret_[0] || ... ||
psk_secret_[n-1]

struct {
  PreSharedKeyID psk;
} PreSharedKey;
```

Inline Proposals

Proposals can be provided **by value** as well as **by hash reference**

One proposals vector to keep ordering clear and allow proposal extensibility

- Proposals applied in batches by type
- Update, Remove, Add, PSK, ReInit
- New Proposal types define order
- Changes effective immediately

```
enum {
  reserved(0),
  proposal(1)
  reference(2),
  (255)
} ProposalOrRefType;

struct {
  ProposalOrRefType type;
  select (ProposalOrRef.type) {
      case proposal:  Proposal proposal;
      case reference: opaque hash<0..255>;
  }
} ProposalOrRef;

struct {
  ProposalOrRef proposals<0..2^32-1>;
  optional<UpdatePath> path;
} Commit;
```

Functionality / Security

- Sub-group branching: create a duplicate group from an existing one (by using PSKs)
- Re-init: restart group with new parameters like new protocol version or ciphersuite (by using PSKs)
- The ratcheting tree is no longer necessarily part of the Welcome message (only its hash)
- An authentication secret is derived from the key schedule that apps can use as part of a verification mechanism for member-to-member authentication
- Indirection for signing keys: allow for deniable signing keys (after draft 10)

Efficiency

- Adding members can again be done in constant time
- The UpdatePath no longer contains secrets for new joiners: fewer operations
- The Secret Tree that produces keys for AEAD for application messages (with efficient FS and unique key/nonce pairs) is now also used for handshake message encryption

TREE HASHING

Flash back to January

Bhargavan / Beurdouche analysis highlighted the need for tree signing, realized as a **parent hash**

Debate at interim over interaction with **tree hash**

Does the tree hash cover the parent hash?

… or does the parent hash cover the tree hash?

At the time, went with the former

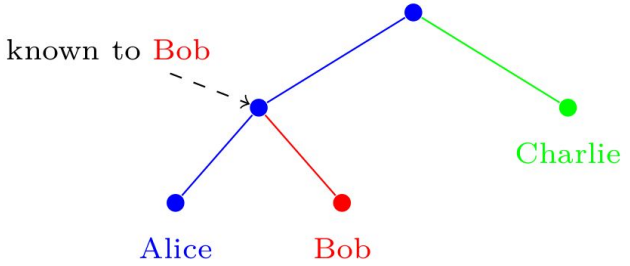
On further thought...

Alwen, Jost, Mularczyk identified a concrete attack based on the omission of off-path information from parent hash

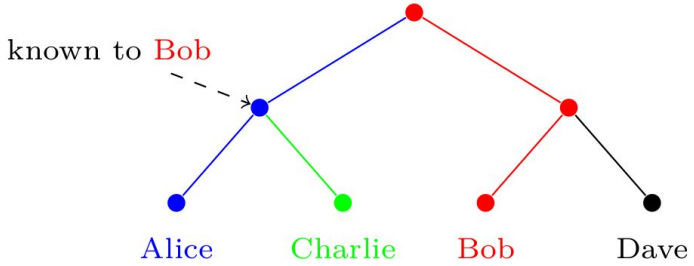
... and a corresponding PR that addresses it by adding more information to the parent hash

On further thought...

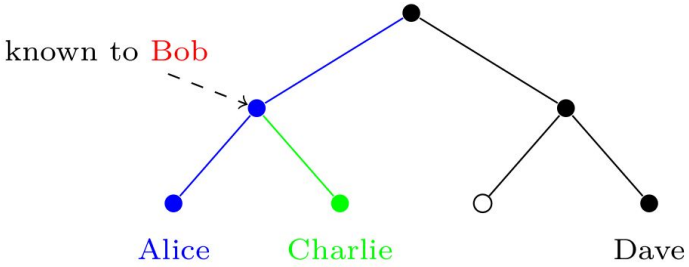
The ratchet tree in a real group.



The ratchet tree created by malicious Bob inviting Dave.



The ratchet tree after Dave commits removing Bob.



On further thought...

The fix intuition: extend Parent Hash to (indirectly) include who was told which key.

Effectively, parent hash needs to cover:

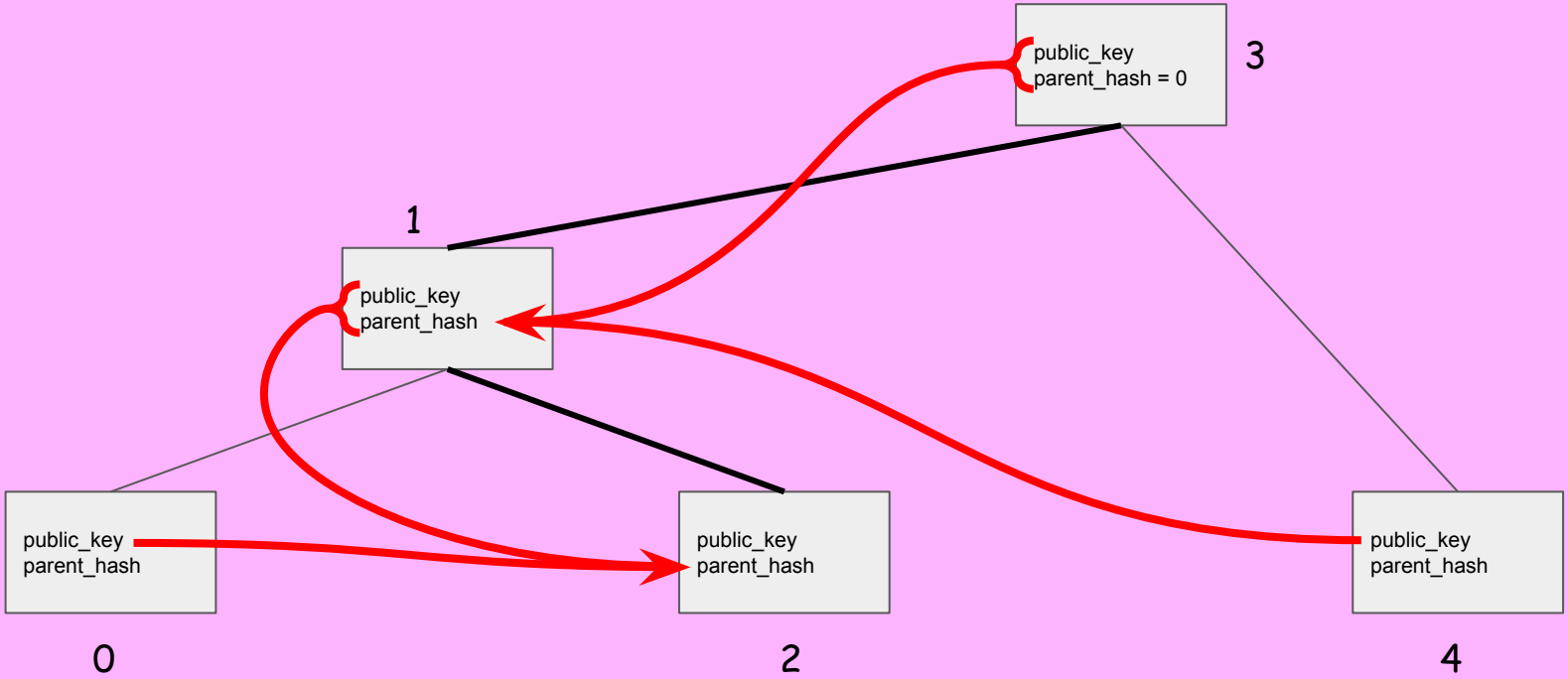
* The parent hash of the parent node (for chaining)
* The public key of the parent node
* The set of public keys to which the parent's private key was encrypted

(Tree hash still covers parent hash)

```
// PR #436

struct {
    HPKEPublicKey parents_public_key;
    opaque parents_parent_hash<0..255>;
    HPKEPublicKey original_sibling_resolution<0..2^32-1>;
} ParentHashInput;
```

Parent Hashes for update path starting at leaf 2.



EXTERNAL COMMITS

External Commit

Motivation: so far a new member could not join a group on its own. One of the existing members needed to send a Welcome message, prohibiting an asynchronous join.

Existing members guard the group membership by rejecting Commits that add invalid members (as per application layer policy). This remains the case with External Commits.

External Commits re-use the existing Commit mechanism, only a few details are changed.

External Commit

If a group wants to allow people to join, publish PublicGroupState with an external_public_key held by the group

Joiner sends "external Commit" to add themselves, asymmetrically advance key schedule

```
external_priv, external_pub =
        KEM.DeriveKeyPair(external_secret)

struct {
      CipherSuite cipher_suite;
      opaque group_id<0..255>;
      uint64 epoch;
      opaque tree_hash<0..255>;
      opaque interim_transcript_hash<0..255>;
      Extension extensions<0..2^32-1>;
      HPKEPublicKey external_pub;
} PublicGroupState;
```

```
struct {
    opaque kem_output<0..2^16-1>;
} ExternalInit;

// Joiner
kem_output, context = SetupBaseS(external_pub, ...)
init_secret = context.export(...)

// Members
context = SetupBaseR(kem_output, external_priv, ...)
init_secret = context.export(...)
```

The Obvious Weakness

PublicGroupState has no signature on it => DS can lie about external_pub

Joiner thinks they've joined the group

... but they're really just in a session with the DS (though DS cannot decrypt the commit secret)

```
struct {
    CipherSuite cipher_suite;
    opaque group_id<0..255>;
    uint64 epoch;
    opaque tree_hash<0..255>;
    opaque interim_transcript_hash<0..255>;
    Extension extensions<0..2^32-1>;
    HPKEPublicKey external_pub;
} PublicGroupState;
```

The Obvious Solution

Add a signature by a group member

Doesn't authenticate the liveness of the whole group, but tree signing helps

Main question has been deniability

- Signature covers membership
- But signature keys can be deniable

PR #433

```
struct {
    CipherSuite cipher_suite;
    opaque group_id<0..255>;
    uint64 epoch;
    opaque tree_hash<0..255>;
    opaque interim_transcript_hash<0..255>;
    Extension extensions<0..2^32-1>;
    HPKEPublicKey external_pub;
    uint32 signer_index;
    opaque signature<0..2^16-1>;
} PublicGroupState;
```

Other security considerations

The key schedule is discontinued

The accumulated entropy of the group is lost

No obvious attack, but slightly different security guarantees

Entirely optional: members can just not publish the public group state and/or not process External Commits

Other Issues on GitHub...

To draft-11 and beyond...