PSKs and Their Use in MLS

Britta Hale [1] Konrad Kohbrok [2]

April 20, 2020

[1]Naval Postgraduate School (NPS)

[2]Aalto University

Use Cases for PSKs

Re-Initialize: Restart a group, e.g. to change ciphersuite.

Recover: Restart a group to recover from a broken/de-synced group state.

Branch: Create a new (sub-)group from an existing group.

External: Feed an external PSK into a group for added security.

Summary

Use Case	Group State	Proposal	New Group
Re-Initialize	Intact	Yes	Yes
Recover	Broken	No	Yes
Branch	Intact[1]	No	Yes (subset of members)
External[2]	Intact	Yes	No

[1] A **Branch** can reference multiple groups to include PSKs from.

[2] In addition to proposing the use of an external PSK, its use can be mandated in any **Welcome** message, potentially in addition to other key material, e.g. an external PSK can be used in addition when a branching a group.

Re-Initialize

- Introduce **Re-Init** proposal with which parties can propose a re-initialization of the group.
  Question: Should the proposal contain relevant information, e.g. the proposed new ciphersuite?
- After receiving a committed **Re-Init** proposal, the group MUST NOT be used to transmit messages anymore
- The committer of the proposal MUST create a new group including all members of the original group and send out a **Welcome** message that indicates the use of a PSK derived from the recovery secret of the last epoch before the **Re-Init** commit (potentially in addition to one or more other, external PSKs).

Use Case?

Change cipher suite. Others?

Recovery

- No need for a proposal, because the assumption is that group members no longer share the same state.
- Any party that wants to recover the group can create a new group and indicate that they want to include a PSK derived from the `recovery_secret` of some last known good epoch.
- Question: Should we have a proposal for recovery? It would probably be a special case of the **Re-Init** proposal.

Use Case?

Recover from a broken group state.

Branching

- We do not want a proposal here, because the client creating the new (sub-)group might not want any other group to know that the new (sub-)group was created.
- Any party that wants to branch from the main group can create a new group and include one or more `PSKIds` in the initial **Welcome** message.

Use Case?

Create a sub-group while inheriting the security level[1] of one or more existing groups.

[1] "Security Level" is currently used in the draft, but is (as far as we know) not precisely defined yet. This is a ToDo for when we describe security guarantees in the architecture document.

External PSKs

- If a party wants to include an external PSK into an existing group, they have to issue an **External-PSK** proposal to that end.
- The proposal specifies at least one `PSKId` with `psktype = external` and with a `pskid` that corresponds to the id of the external PSK (whatever id scheme it uses).
- Alternatively, if they are creating a new group, they can just add the `PSKIds` to the **Welcome** messages.

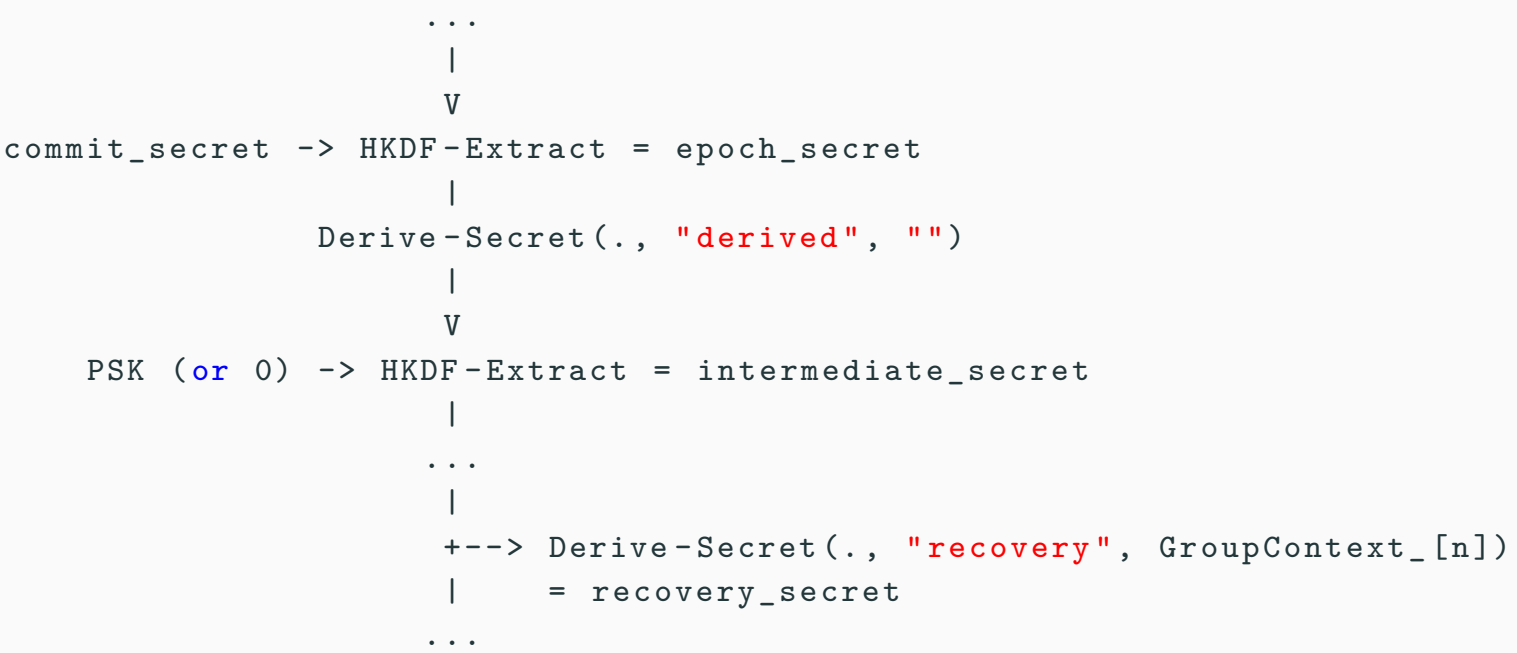
Use Case?

Inject additional shared randomness into the group for (potential) extra security.

Key Schedule

Proposed basic changes:

- Flip order of `PSK` and `commit_secret` injection.
- Add derivation of a `recovery_secret` (distinct from exporter secret)



PSK Injection

We have to avoid collision between *internal* and *external* PSKs. Thus, before injecting a PSK into the Key Schedule, it must be labeled accordingly.

```
enum {
  re-initialization(0),
  recovery(1),
  branch(2),
  external(3),
  (255)
} PSKType;
```

```
PSKLabel(psktype) =
  select (psktype) {
    case re-initialization:
      "re-inizialization";
    case recovery: "recovery";
    case branch: "branch";
    case external: "external";
}
```

We can then derive a PSK as follows:

```
PSK_Inject = Derive-Secret(PSK, PSKLabel(psktype))
```

Inject Multiple PSKs

Especially when starting up a group, a client might want to use multiple internal and/or external PSKs. The idea is to

- derive every individual PSK as shown in the previous slide and then
- `HKDF-Extract` them to a single key, which is then injected into the key schedule.

PSKs in Welcome messages

We indicate which PSK to use when creating a new group by adding an `optional<PSKId> psk<1..2^32-1>;` entry to the `KeyPackage` in the **Welcome** message.

```
struct {
  opaque epoch_secret<1..255>;
  opaque path_secret<1..255>;
  optional<PSKId> psk<1..2^32-1>;
} KeyPackage;
```

PSKId

A `PSKId` details the purpose of the PSK and from which `recovery_secret` it should be derived.

We also include a randomly generated nonce to avoid collisions when using the same `recovery_secret` multiple times, e.g. when multiple branches are made off of the same group.

```
struct {
  PSKType psktype;
  select (psktype) {

    case external:
      opaque psk_id<0..255>;

    default:
      opaque psk_group_id<0..255>;
      uint64 psk_epoch;

  }
  opaque psk_nonce<0..255>;
} PSKId
```

Deriving From Recovery Secrets

Let `recovery_secret(psk_group_id, psk_epoch)` be the `recovery_secret` of the group with group id `psk_group_id` and epoch `psk_epoch`.

```
recovery_key =
  HKDF-Expand(
    recovery_secret(psk_group_id, psk_epoch),
    Hash(PSKLabel(psktype) || recovery_nonce)
  )
```