Messaging Layer Security

Security Analysis

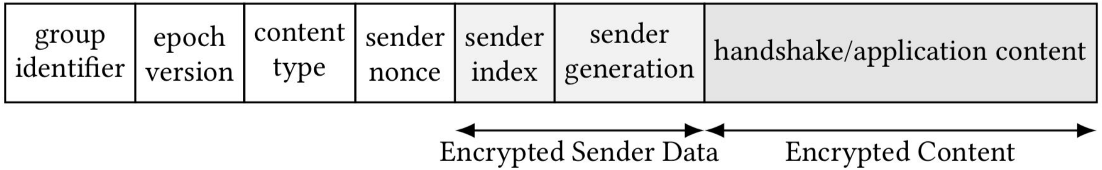
Transcript Hash

```
struct {
    GroupOperationType msg_type;
    select (GroupOperation.msg_type) {
        case init:      Init;
        case add:       Add;
        case update:    Update;
        case remove:    Remove;
    };
    opaque confirmation<0..255>;
} GroupOperation;
```

What does the transcript hash cover?

Only operations, not the sender of the operation!

We need agreement on which member sent which update.

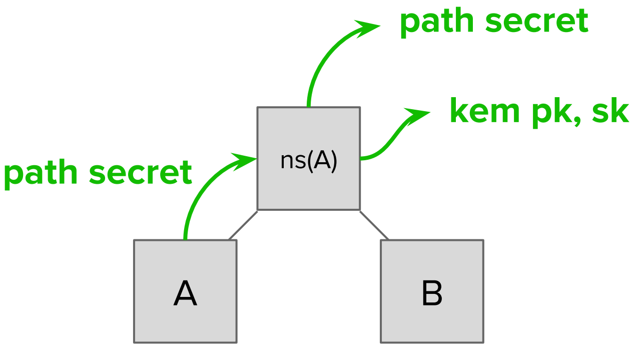


group identifier	epoch version	content type	sender nonce	sender index	sender generation	handshake/application content

Encrypted Sender Data

Encrypted Content

Tree Key Derivation



No context is used in the derivation

Possible for two different subtrees and two different transcripts to result in same path secret

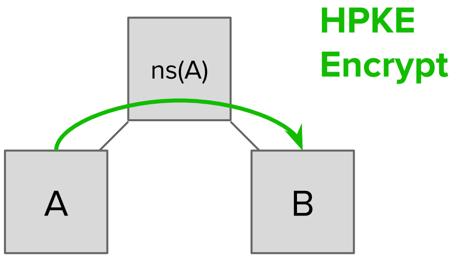
To guarantee independent keys:

Add to context:

previous transcript hash + subtree hash?

```
path_secret[n] = HKDF-Expand-Label(path_secret[n-1], "path", context, Hash.Length)
node_secret[n] = HKDF-Expand-Label(path_secret[n], "node", context, Hash.Length)
node_priv[n], node_pub[n] = Derive-Key-Pair(node_secret[n])
```

HPKE Encrypt context



The key derivation has an empty context while both could agree on it.

Signature Authentication

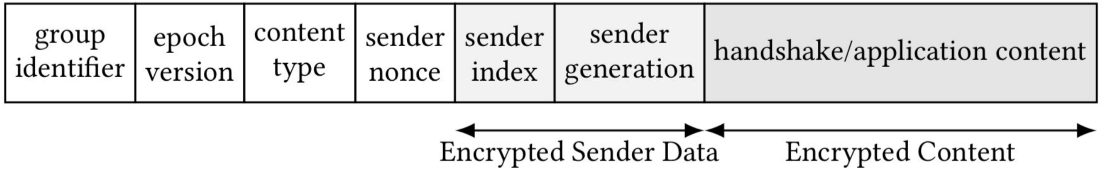
```
struct {
    GroupOperationType msg_type;
    select (GroupOperation.msg_type) {
        case init:       Init;
        case add:        Add;
        case update:     Update;
        case remove:     Remove;
    };
    opaque confirmation<0..255>;
} GroupOperation;
```

What does the signature cover?

In handshake messages: key confirmation, which transtively covers the transcript hash

In application messages, only content, not the transcript hash?

If so, there is a cross-group forwarding attack



group identifier | epoch version | content type | sender nonce | sender index | sender generation | handshake/application content

Encrypted Sender Data

Encrypted Content

Double Join for Newly Added Nodes

```
struct {
    HPKEPublicKey public_key;
    optional<Credential> credential;
} RatchetNode;

struct {
    ProtocolVersion version;
    opaque group_id<0..255>;
    uint32 epoch;
    optional<RatchetNode> tree<1..2^32-1>;
    opaque transcript_hash<0..255>;
    opaque init_secret<0..255>;
} WelcomeInfo;

struct {
    opaque user_init_key_id<0..255>;
    CipherSuite cipher_suite;
    HPKECiphertext encrypted_welcome_info;
} Welcome;
```

A new member must fully trust the adder

Sender may lie about node public keys

Suppose a group has only one malicious member A.

Suppose A adds B and gives it a bogus tree.

Suppose B deletes A and sends a message to the group.

Can this message be read by the attacker?

A Tree of Signatures

```
struct {
    HPKEPublicKey public_key;
    optional<Credential> credential;
} RatchetNode;

struct {
    ProtocolVersion version;
    opaque group_id<0..255>;
    uint32 epoch;
    optional<RatchetNode> tree<1..2^32-1>;
    opaque transcript_hash<0..255>;
    opaque init_secret<0..255>;
} WelcomeInfo;

struct {
    opaque user_init_key_id<0..255>;
    CipherSuite cipher_suite;
    HPKECiphertext encrypted_welcome_info;
} Welcome;
```

What if every subtree were signed by the last member who modified that subtree

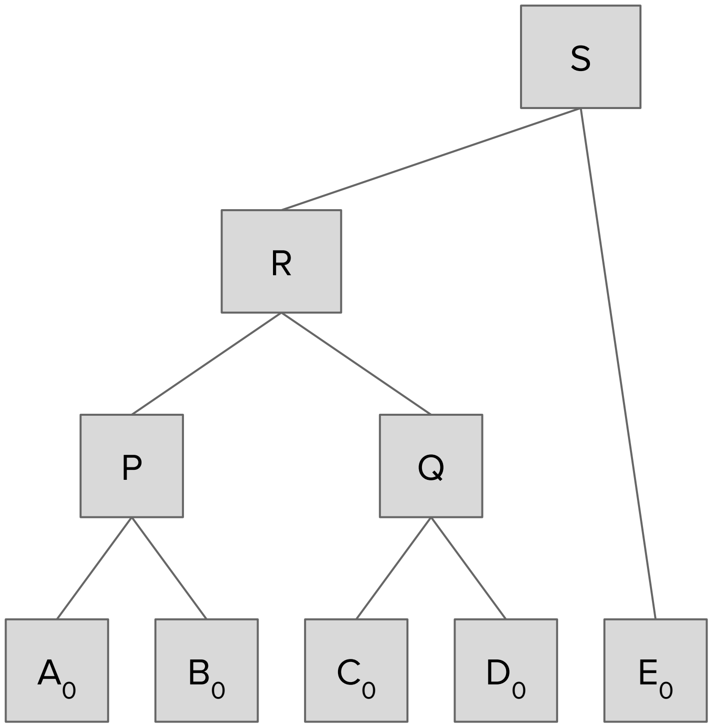
Every send operation requires O(log n) signatures

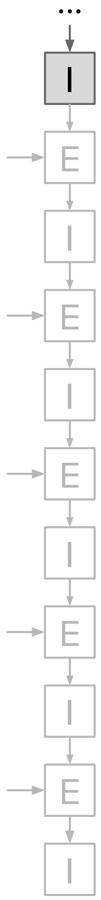
The state of the group now also needs to store one signature per node.

But, we can provide new members with the same guarantees as existing members, without trusting the adding member.

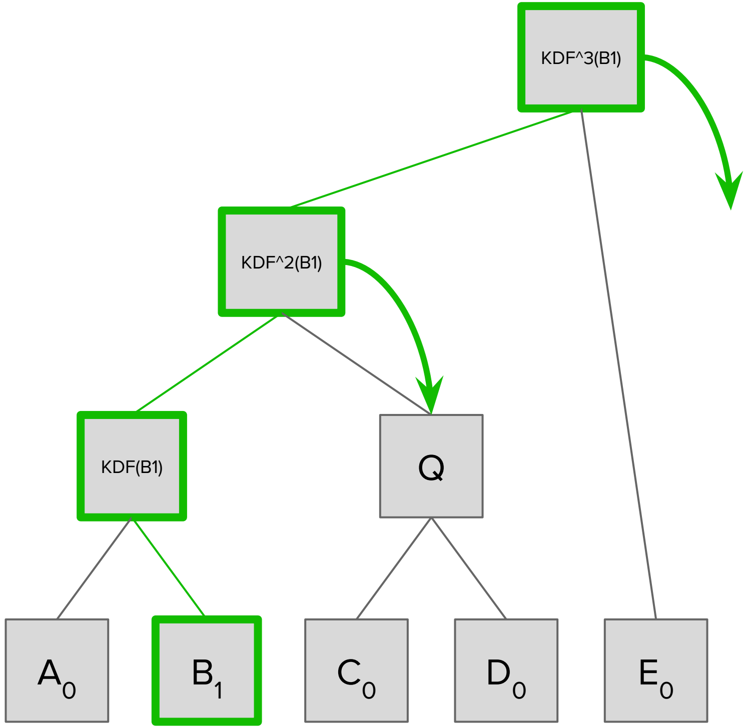
Remove then Update?

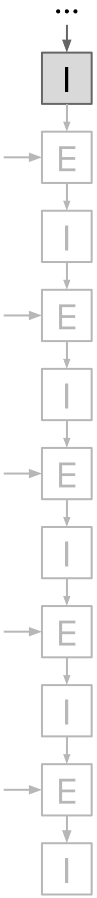
Assume a tree...



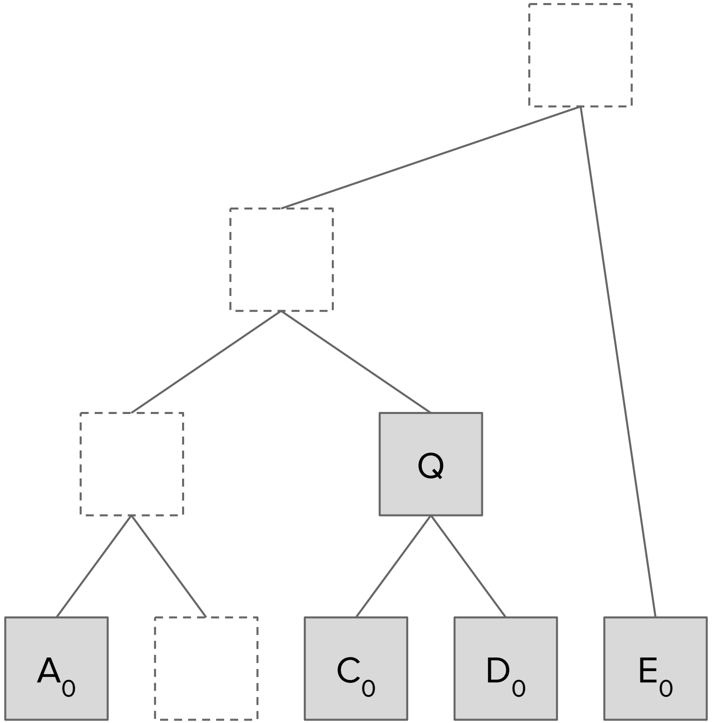


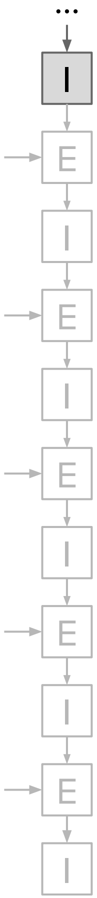
Up to now, we are removing B… by



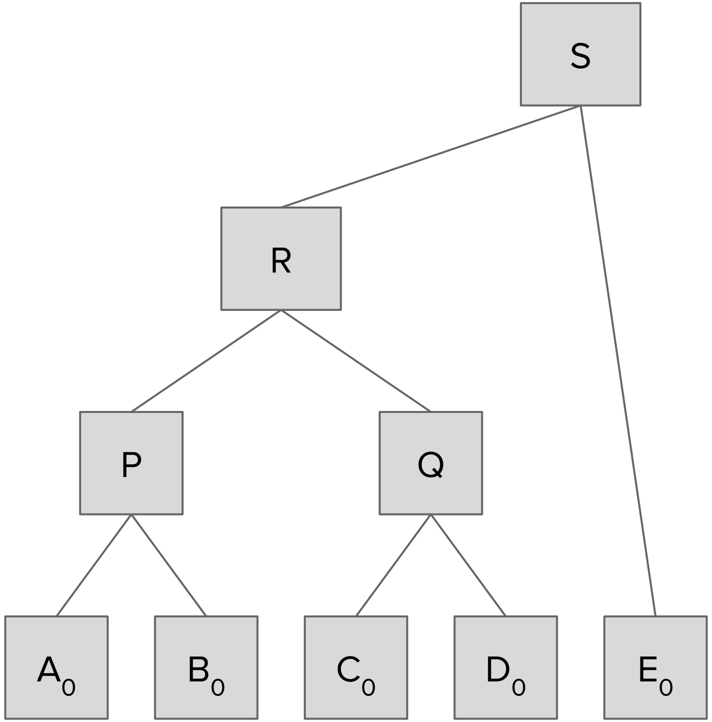


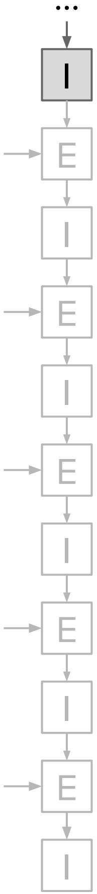
Up to now, we are removing B... by



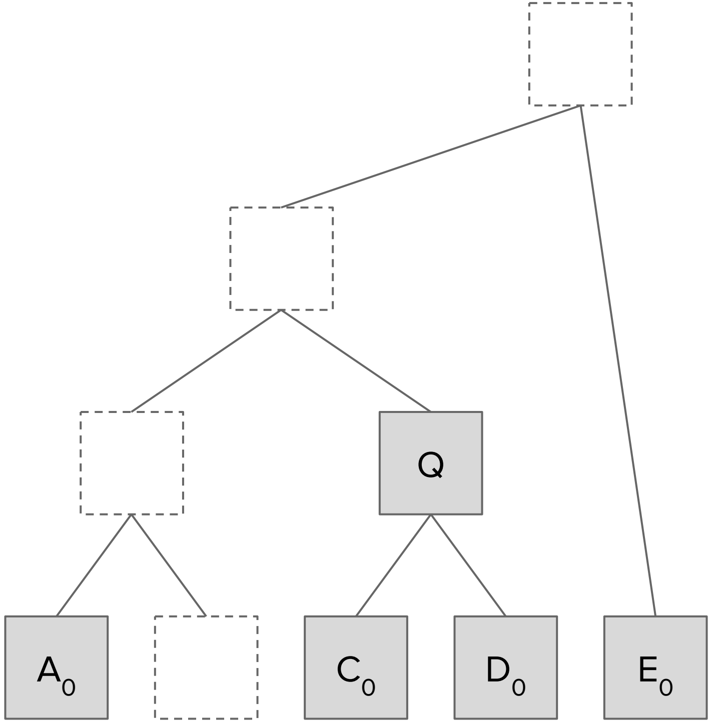


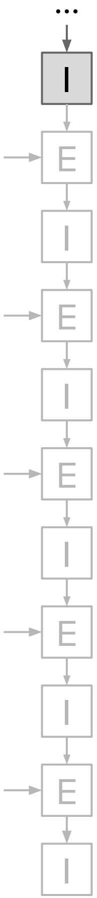
Assume a tree...



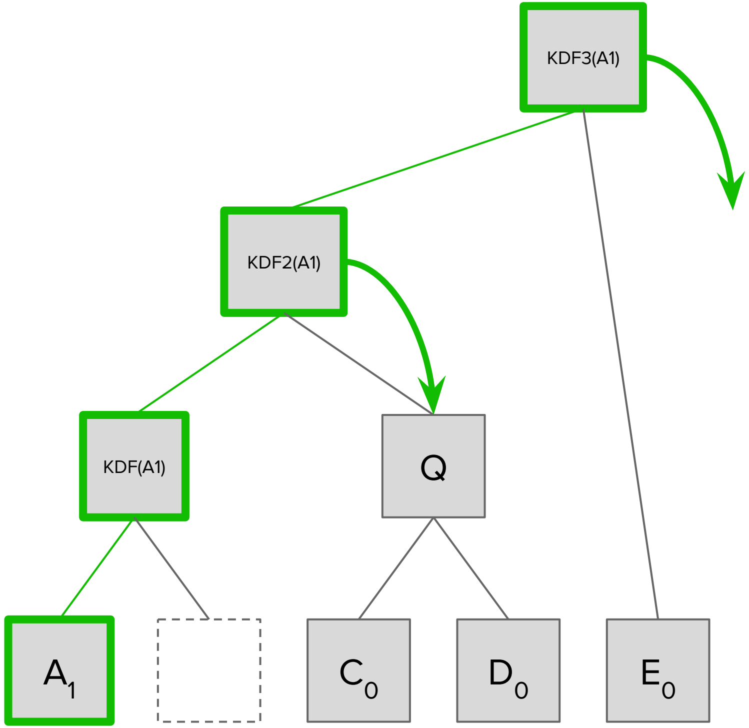


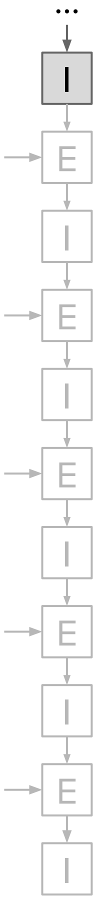
We can blank first...





And the actor can then update.


KDF3(A1)
KDF2(A1)
KDF(A1)
Q
A_1
C_0
D_0
E_0


...
I
E
I
E
I
E
I
E
I
E
I