

Messaging Layer Security (MLS)

RFC 9420

TABLE OF CONTENTS

01

Protocol Overview

02

TreeKEM

03

TreeDEM

04

TreeSync

05

OpenMLS

06

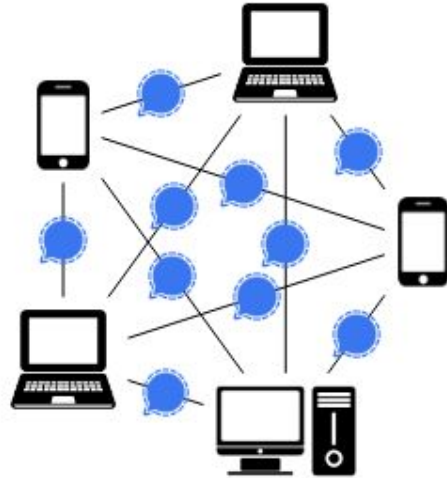
My project



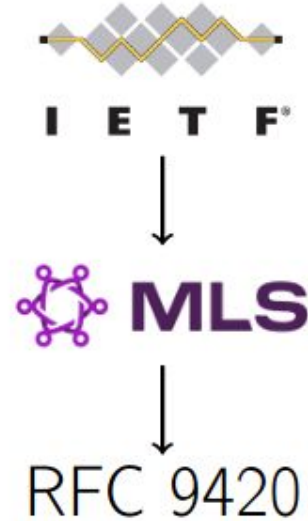
0

Why MLS ?

State of the art before MLS



N devices
 $O(N^2)$ Signal channels!
Slow for large N , e.g. $N \simeq 1000$



Design constraints:
Secure, efficient, asynchronous,
dynamic groups



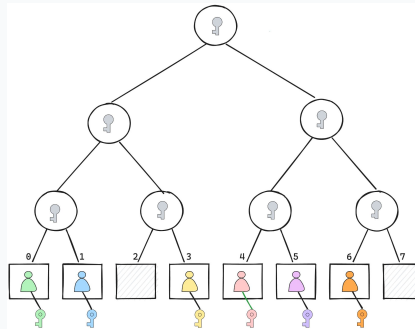
01

PROTOCOL OVERVIEW



Groups

- MLS provides a way for *clients* to form *groups* within which they can communicate **securely**.
- As groups change membership and group or member properties, they advance from one **epoch** to another and the **cryptographic state** of the group evolves
- The group is represented as a tree, which represents the members as the leaves of a tree. It is used to efficiently encrypt to subsets of the members. Each member has a state called a *LeafNode* object holding the client's identity, credentials, and capabilities



Threats

The main threats to messages sent over MLS are:

- from network attackers
- from malicious servers
- and from compromised group members whose cryptographic state and keys have been obtained by the adversary.

Security Goals - 1

- **Message Confidentiality:** If a client C sends a message M in epoch E of group G , and C believes that the membership of G in E is C_0, \dots, C_n , then M is kept secret from the adversary as long as none of these members is compromised.
- **Forward Secrecy:** If a client C sends (or receives) a message M in epoch E of group G , then any compromise of C after this point does not affect the confidentiality of M .
- **Message Authentication:** If a client C accepts a message M in epoch E of group G , and if C believes that the membership of G in E is C_0, \dots, C_n , and if none of these members are compromised at the time of reception, then M must have been sent by one of these group members for the group G in epoch E .

Security Goals - 2

- **Sender Authentication:** If a client C accepts a message M seemingly sent by a client C' in epoch E of group G , and if C' is uncompromised at the time of reception, then M must indeed have been sent by C' in epoch E of group G .
- **Membership Agreement:** If a client C accepts a message M from a client C' in epoch E of group G , then C and C' must agree on the membership of G at E .
- **Post-Remove Security:** If a client C was member of group G in epoch E , and was no longer a member in epoch $E+1$, then even if C was compromised in epochs $\leq E$, it does not affect the confidentiality of messages sent in epochs $\geq E+1$.
- **Post-Update Security:** If a client C was member of group G in epoch E , and has updated its cryptographic keys in epoch $E+1$, then even if the previous state of C in epochs $\leq E$ was compromised, it does not affect the confidentiality of messages sent in epochs $\geq E+1$.



Membership Agreement

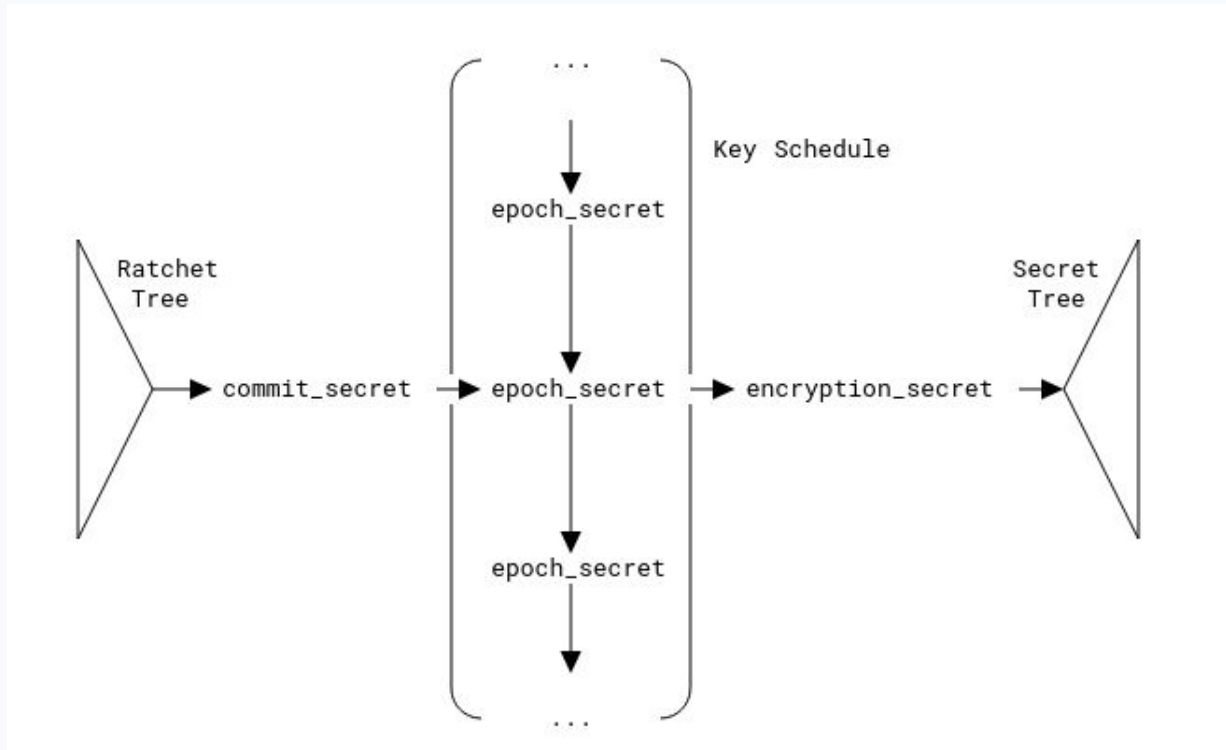
The first truly novel security goal in MLS is membership agreement, which guarantees that group members agree with each other on the current membership. In practice, MLS requires an even stronger agreement on the entire membership history of the group and its cryptographic states. Note that currently deployed group protocols like Signal Sender Keys do not provide membership agreement.

Evolution of an epoch

Various messages are used in the evolution from epoch to epoch:

- A **Proposal** message proposes a change to be made in the next epoch, such as adding or removing a member.
- A **Commit** message initiates a new epoch by instructing members of the group to implement a collection of proposals. Proposals and Commits are collectively called *Handshake messages*.
- A **KeyPackage** provides keys that can be used to add the client to a group, including its LeafNode, and *Signature Key*.
- A **Welcome** message provides a new member to the group with the information to initialize their state for the epoch in which they were added.

Cryptographic State and Evolution



Areas of responsibility

Ratchet tree

represents the membership of the group, providing group members a way to authenticate each other and efficiently encrypt messages to subsets of the group. Each epoch has a distinct ratchet tree. It seeds the key schedule

Key schedule

describes the chain of key derivations used to progress from epoch to epoch (mainly using the `init_secret` and `epoch_secret`), as well as the derivation of a variety of other secrets

Secret tree

tree derived from the key schedule that represents shared secrets used by the members of the group for encrypting and authenticating messages. Each epoch has a distinct secret tree.

Abstract Services

MLS is designed to operate within the context of a messaging service:

- **Authentication Service** (AS) which is responsible for attesting to bindings between application-meaningful identifiers and the public key material used for authentication in the MLS protocol.
- **Delivery Service** (DS) which can receive and distribute messages between group members. The DS is also responsible for storing and delivering initial public key material required by MLS clients in order to proceed with the group secret key establishment that is part of the MLS protocol.

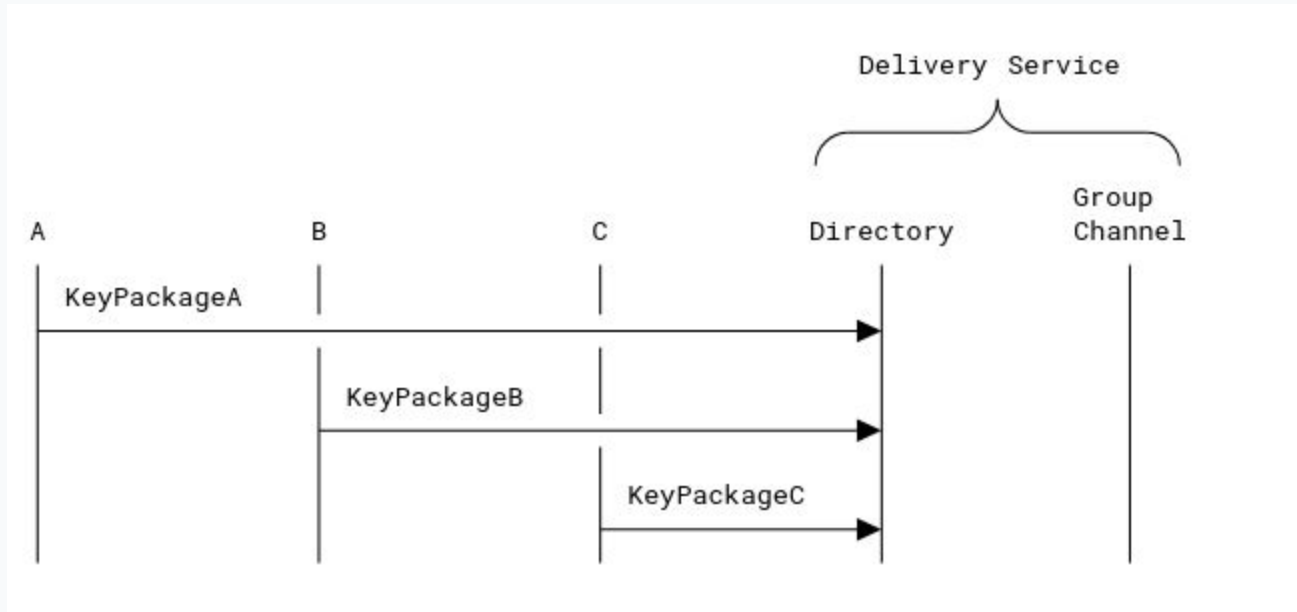


Example Protocol Execution

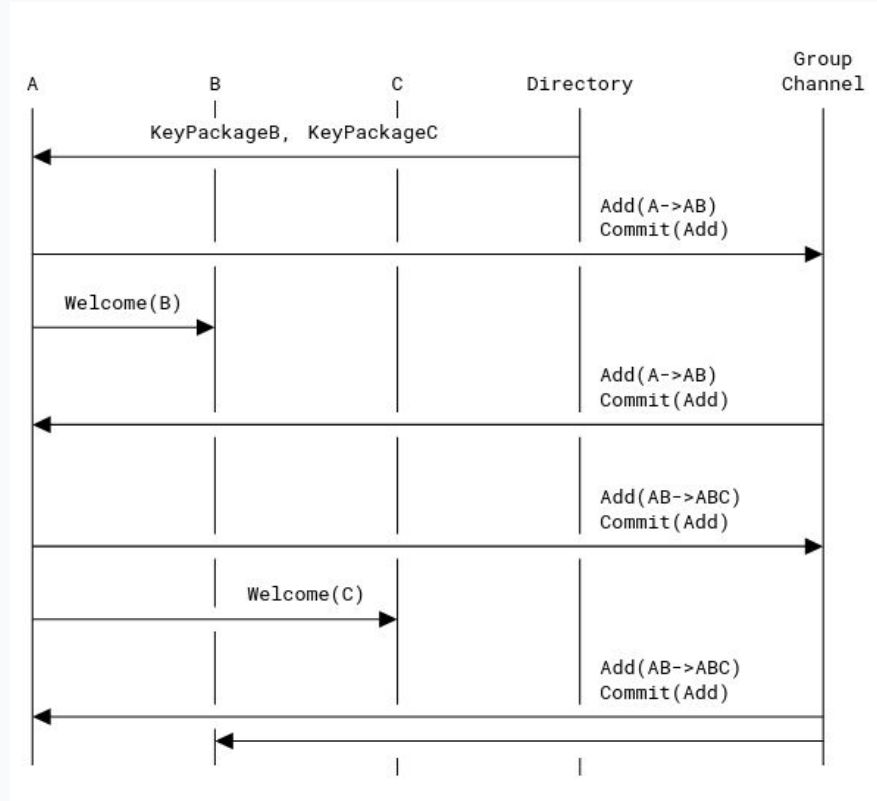
There are three major operations in the life of a group:

- Adding a member
- Updating the keys that represent a member in the tree
- Removing a member

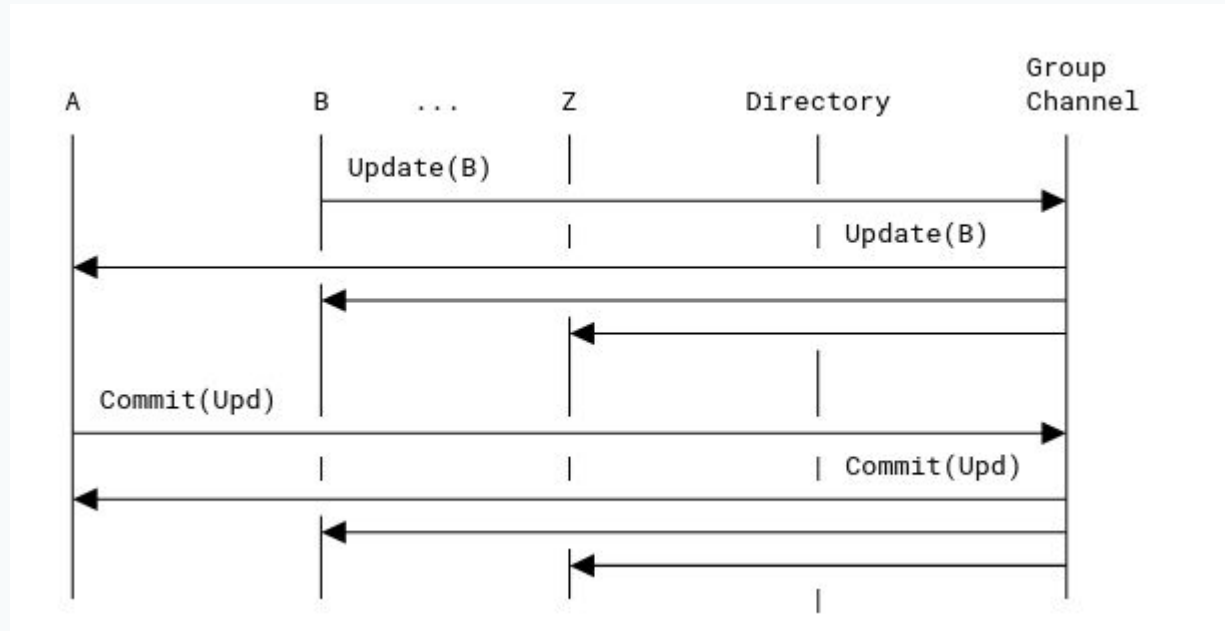
Publish KeyPackages



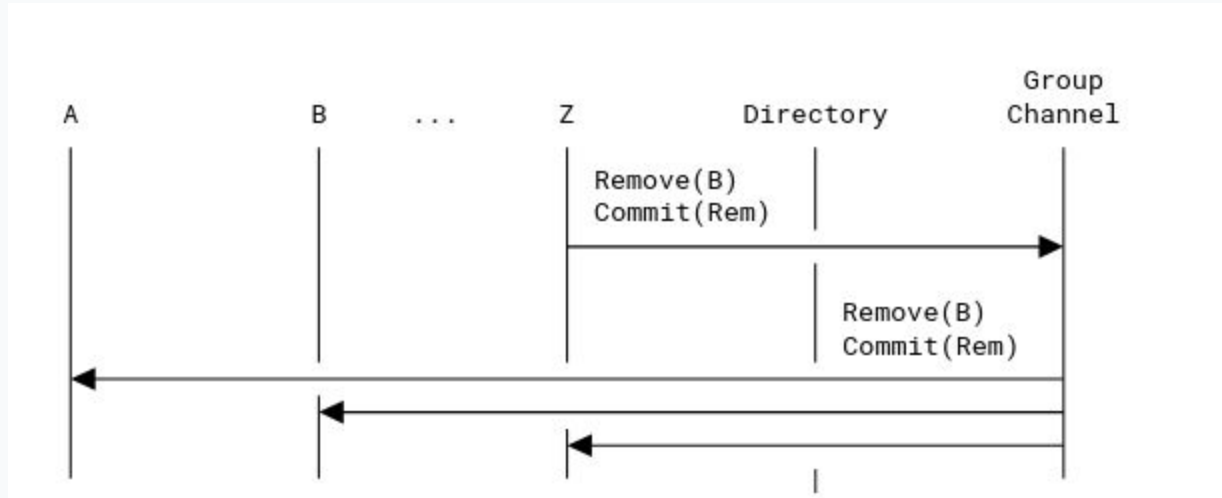
Client A creates a group with clients B and C



Client B proposes to update its key, and client A commits the proposal



Client Z removes client B from the group



The MLS Approach

TreeSync, TreeKEM, TreeDEM

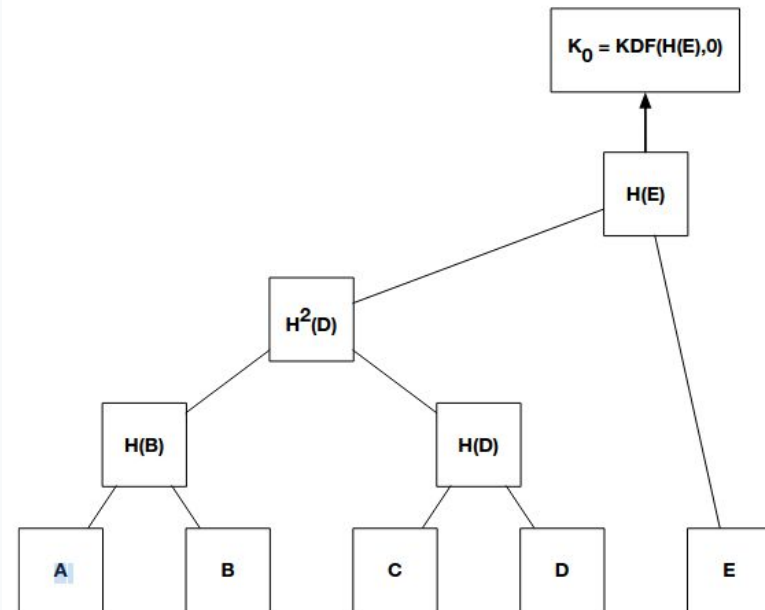


02

TreeKEM

Ratchet Tree Concepts

- The protocol uses "ratchet trees" for deriving shared secrets among a group of clients
- A ratchet tree is an arrangement of secrets and key pairs among the members of a group in a way that allows for secrets to be efficiently updated to reflect changes in the group

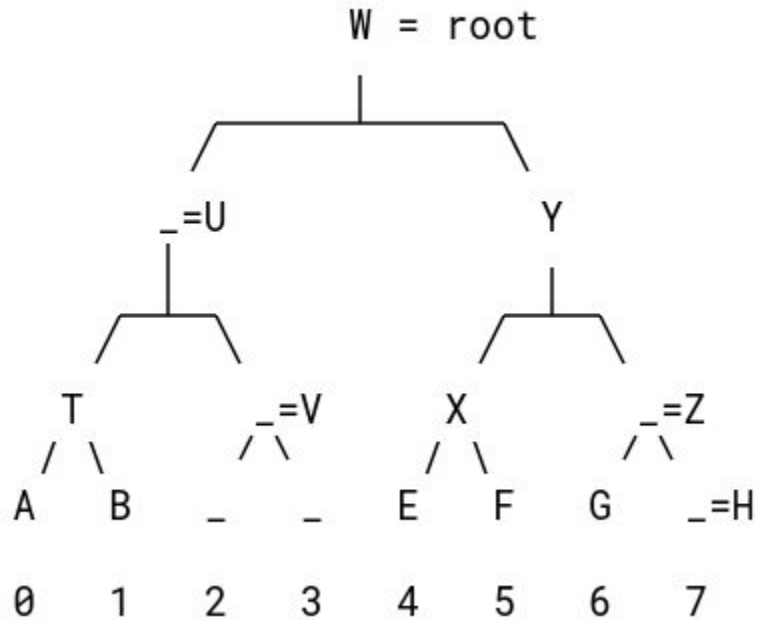


Ratchet Tree Nodes

Each node in a ratchet tree is either blank (containing no value) or it holds an HPKE public key with some associated data:

- A public key
- A credential
- An ordered list of "unmerged" leaves
- A hash of certain information about the node's parent, as of the last time the node was changed

Paths through a Ratchet Tree

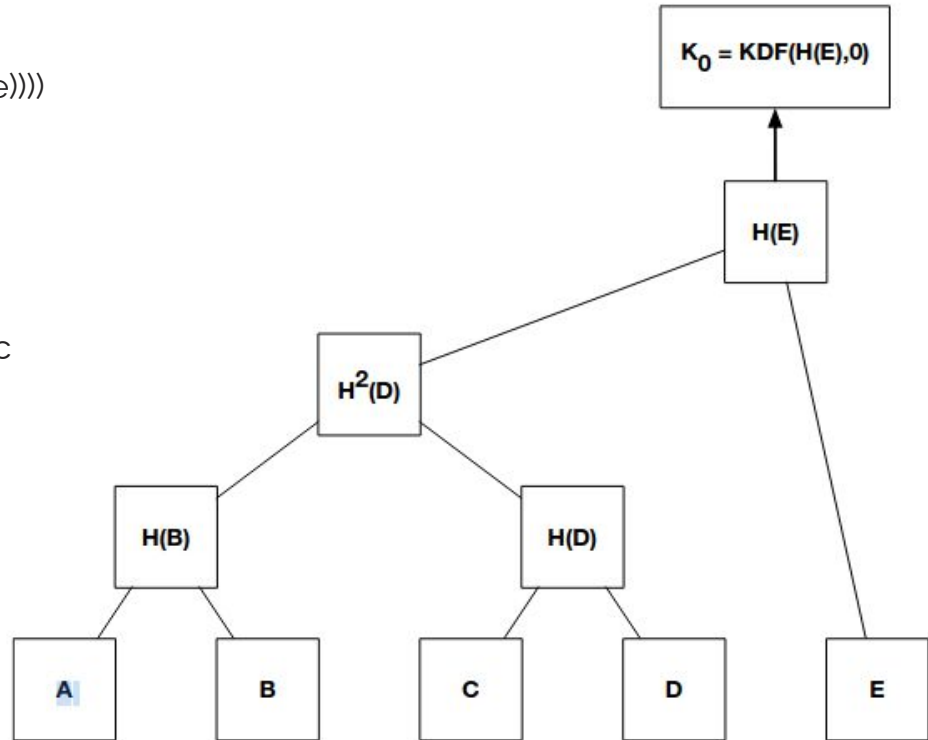


Node	Direct path	Copath	Filtered Direct Path
A	T, U, W	B, V, Y	T, W
B	T, U, W	A, V, Y	T, W
E	X, Y, W	F, Z, U	X, Y, W
F	X, Y, W	E, Z, U	X, Y, W
G	Z, Y, W	H, X, U	Y, W

Computing Tree Keys

sendcreate($d_i, g', \{d_o, \dots, d_n\}$):

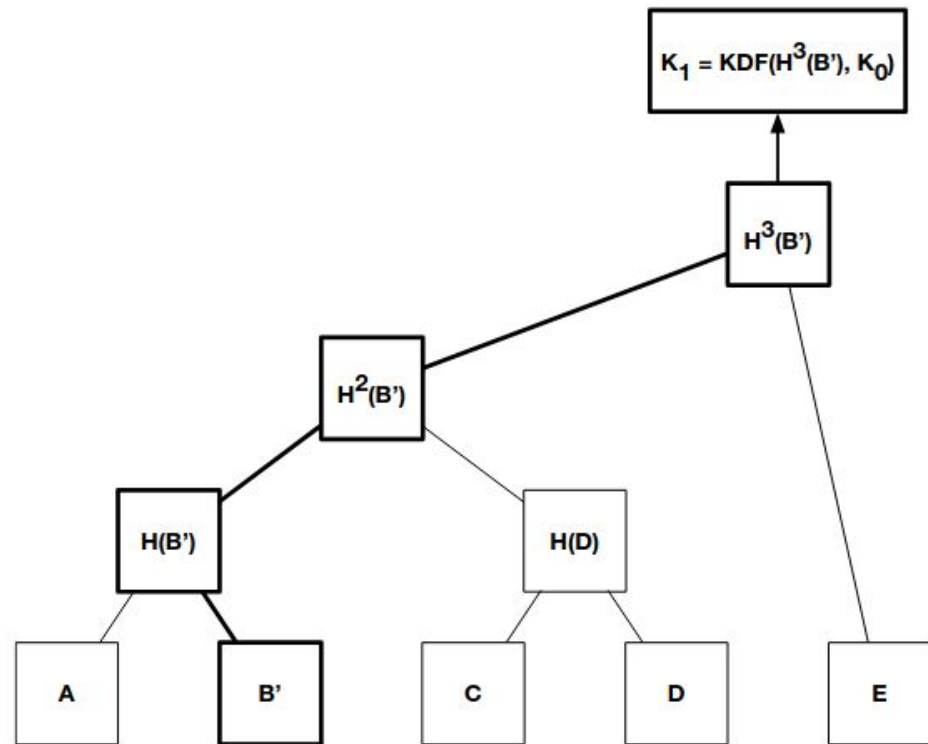
- To a: $\text{penc_a}(K_a, H(K_b), H(H(K_d)), H(H(H(K_e))))$
+ public key(K_b), public key($H(K_d)$), public key($H(H(K_e))$)
- To b: $\text{penc_b}(K_b, H(H(K_d)), H(H(H(K_e))))$ + public key(co-path)
- To c: $\text{penc_c}(K_c, H(K_d), H(H(H(K_e))))$ + public key(co-path)



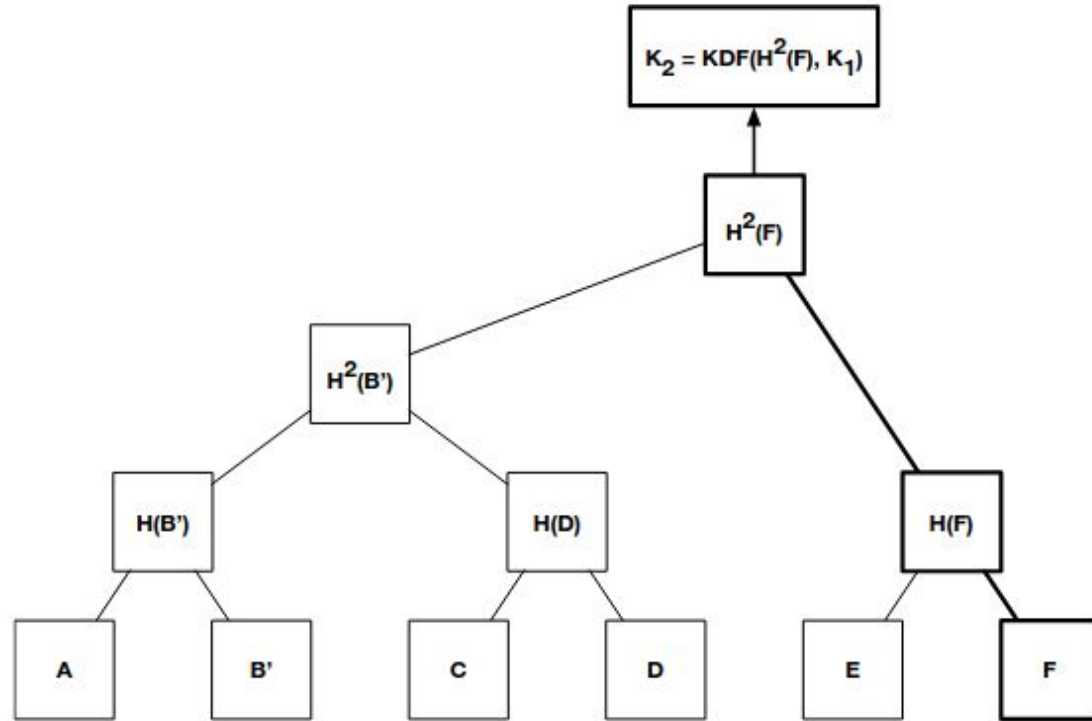
Key update

sendupdate(d_i, g_j):

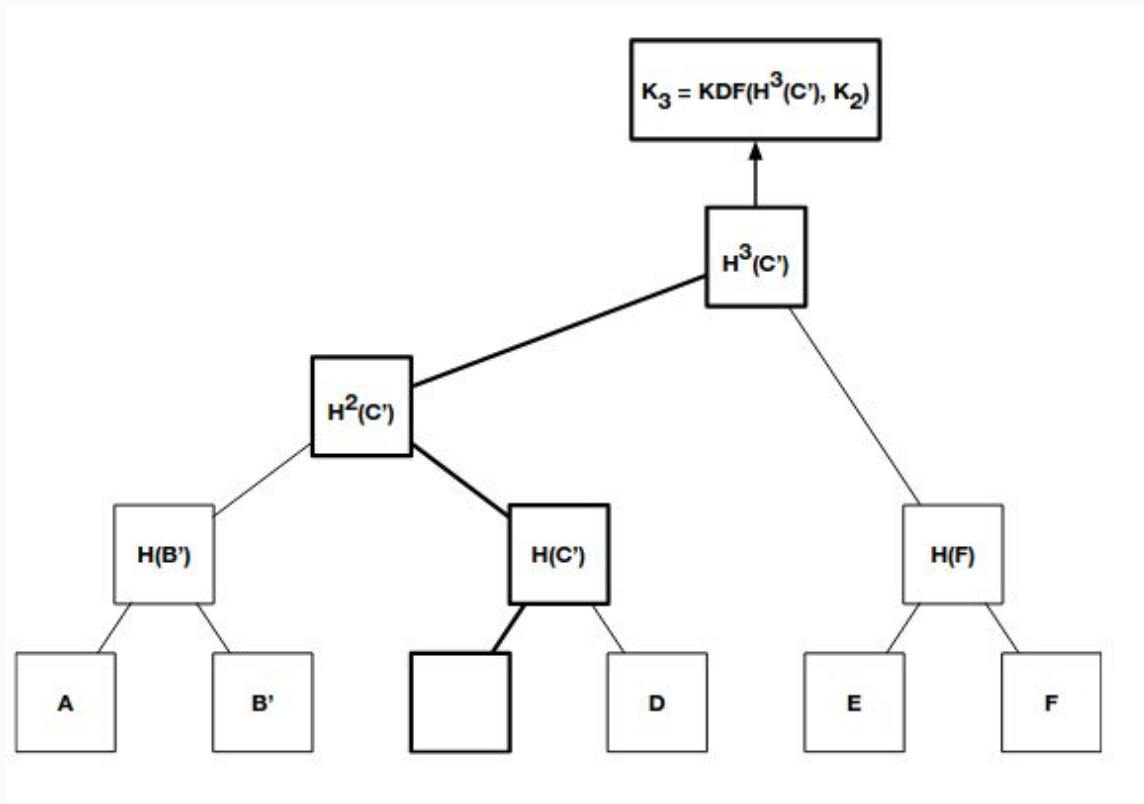
- To a: $\text{penc}_a(H(K'b)) + \text{public key}(K'b)$
- To cd: $\text{penc}_{(H(K_d))(H(H(K'b)))} + \text{public key}(H(K'b))$
- To e: $\text{penc}_{(H(H(K_e))(H(H(H(K'b)))))} + \text{publickey}(H(H(K'b)))$



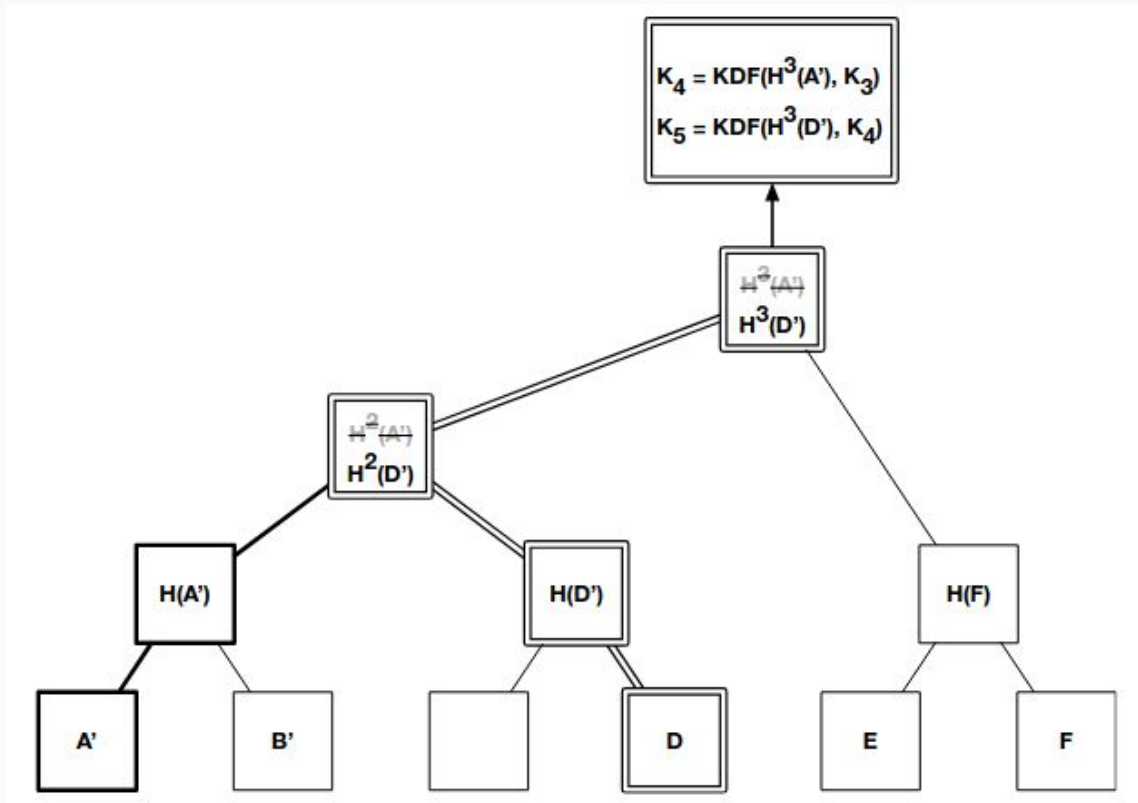
New device has been added



Device removal

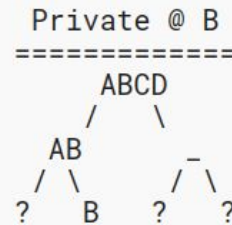
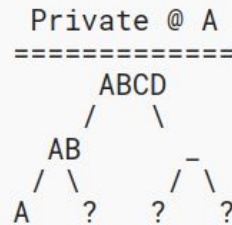
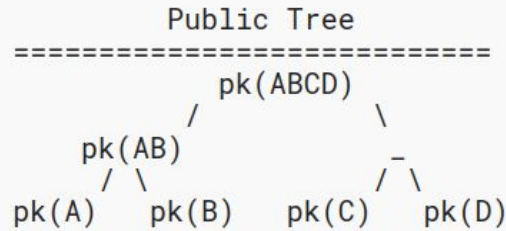


Simultaneous updates



Views of a Ratchet Tree

- The private key for a node in the tree is known to a member of the group only if the node's subtree contains that member's leaf




03

TreeDEM

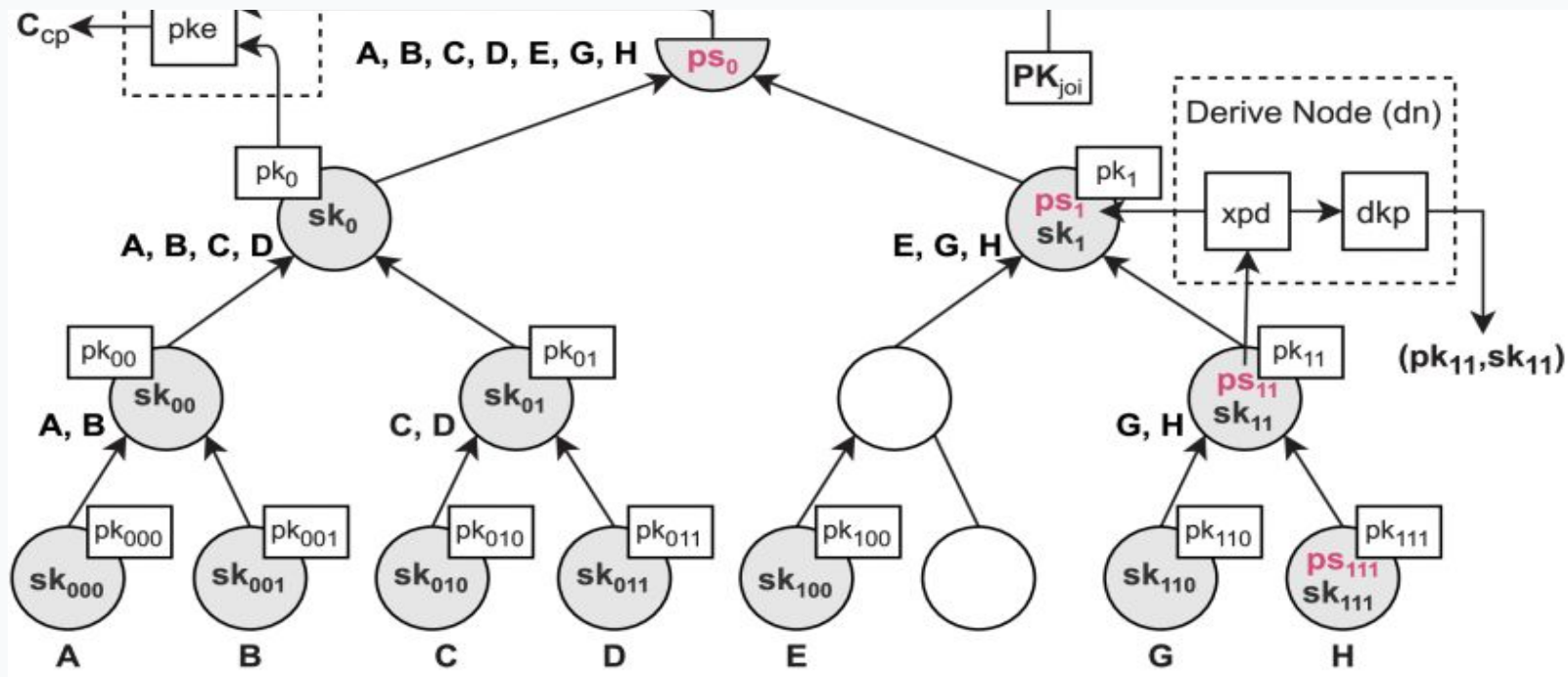
TreeDEM: Group Message Encryption

- Given an epoch secret, the TreeDEM protocol derives message encryption keys for each member in the current group and specifies how they are used to send and receive application messages, handshake messages (containing TreeKEM proposals and commits), and Welcome messages for new members.

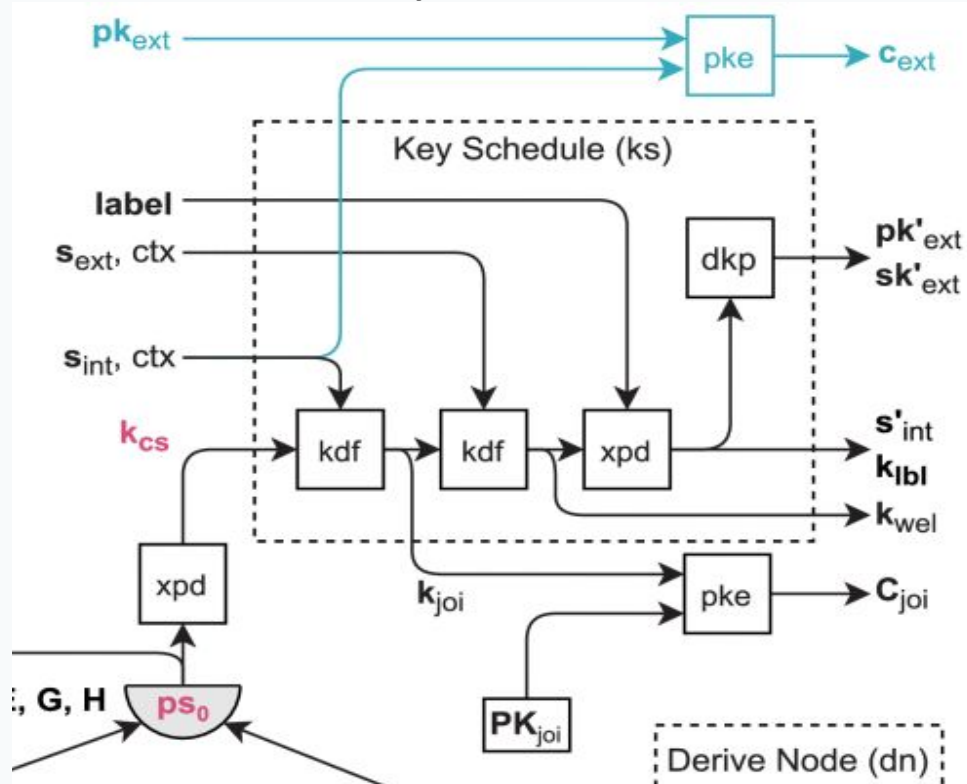


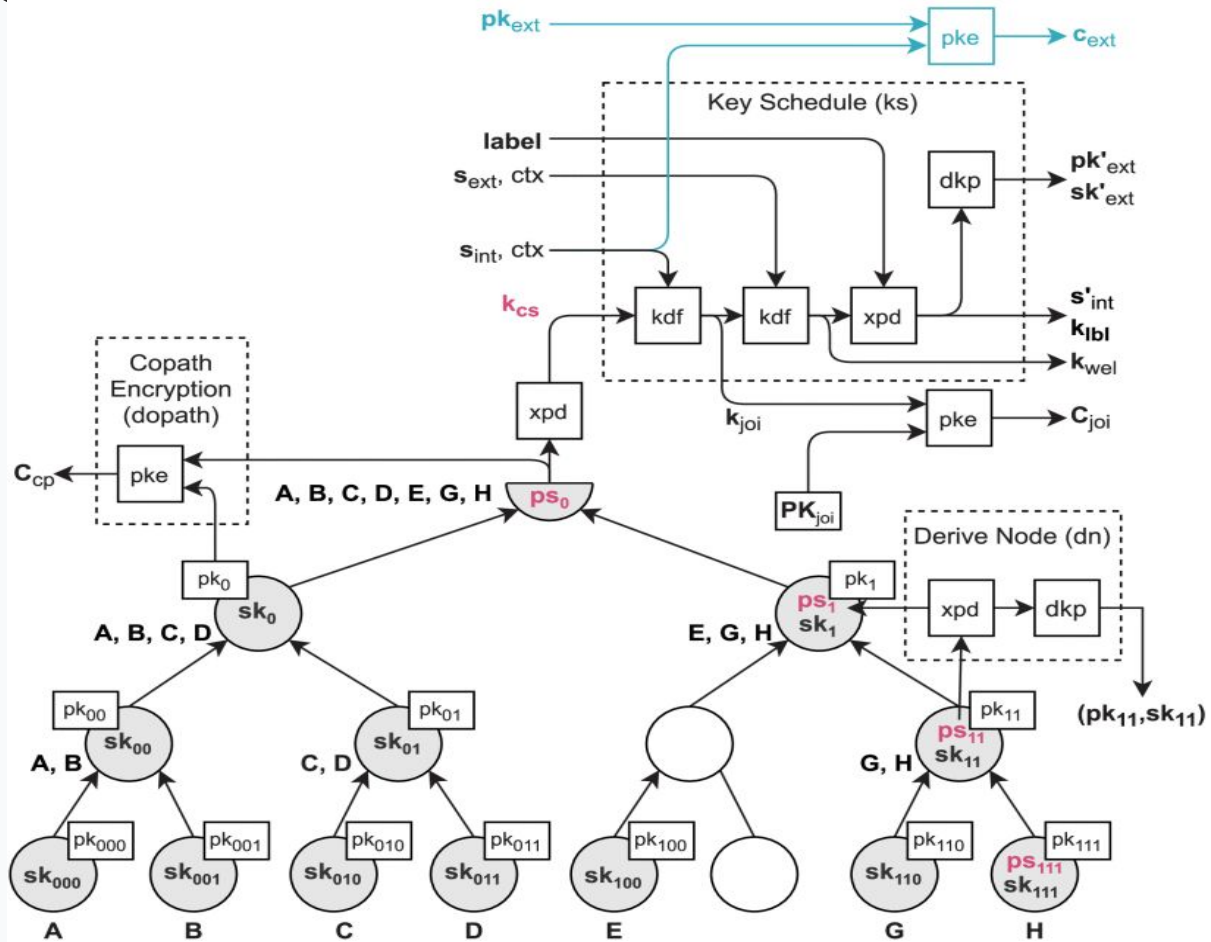
How TreeDEM authenticates and encrypts application messages

- Each application message is serialized into a bitstring along with metadata indicating the group, epoch, and sender
- This bitstring is then signed with the sender's signing key
- and then MACed with a key derived from current epoch secret
- The serialized message, its signature, and MAC are then encrypted using an authenticated encryption (AEAD) scheme
- The recipient performs the reverse set of operations to decrypt the message, verify the signature and MAC to ensure that the message was sent by a sender who is a member of the group



Key Schedule





04

TreeSync



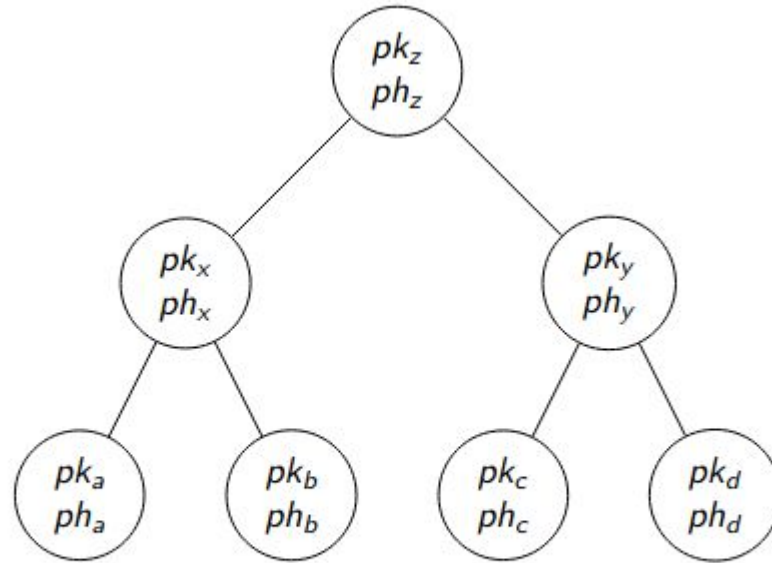
Why?

- Assuming Alice enters a secure group and receives the public key tree, how can she be sure that these keys are not controlled by an attacker?
- How can she be sure about who the participants in the group are? Can an attacker be in the group without her knowing? Is Bob really Bob or is he someone else?

TreeSync aims to solve these problems by going to authenticate TreeKEM states. Specifically:

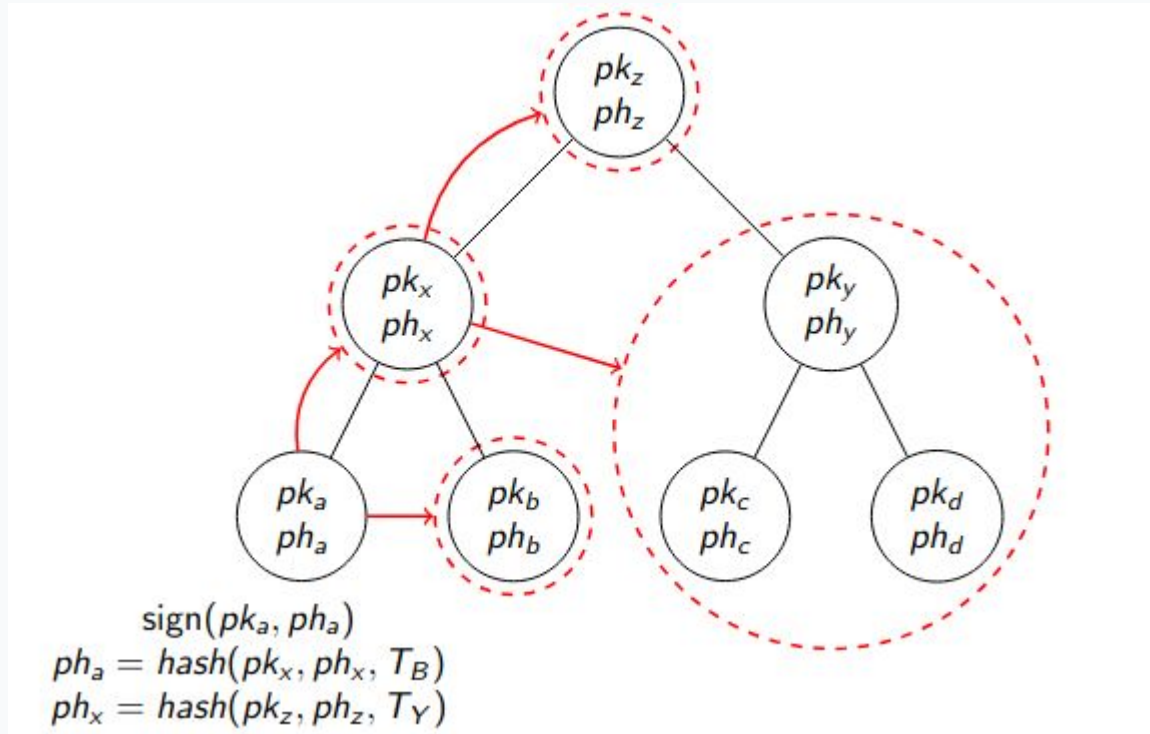
- authenticates all public keys
- authenticate group membership agreements

TreeSync - Final Attempt



$\text{sign}(pk_a, ph_a)$
 $ph_a = \text{hash}(pk_x, ph_x, T_B)$
 $ph_x = \text{hash}(pk_z, ph_z, T_Y)$

TreeSync - Final Attempt



05

OpenMLS

OpenMLS

- A open source implementation of the Messaging layer security protocol
- It has a safe and easy-to-use interface that hides the complexity of the underlying cryptographic operations.
- The OpenMLS library is written in pure Rust. It supports different cryptographic backend libraries. It ships with a backend based on RustCrypto and an optional backend based on Evercrypt.
- <https://openmls.tech/>

Supported ciphersuites

- MLS_128_HPKEX25519_AES128GCM_SHA256_Ed25519 (MTI)
- MLS_128_DHKEMP256_AES128GCM_SHA256_P256
- MLS_128_HPKEX25519_CHACHA20POLY1305_SHA256_Ed25519

Cipher suite names follow the naming convention:

- CipherSuite MLS_LVL_KEM_AEAD_HASH_SIG = VALUE;

Component	Contents
LVL	The security level (in bits)
KEM	The KEM algorithm used for HPKE in ratchet tree operations
AEAD	The AEAD algorithm used for HPKE and message protection
HASH	The hash algorithm used for HPKE and the MLS transcript hash
SIG	The signature algorithm used for message authentication

06

My project

My Project

My project simulates the following operations:

- Alice creates a group
- Alice adds Bob
- Alice sends a message to Bob
- Bob updates and commits
- Alice updates and commits
- Bob adds Charlie
- Charlie sends a message to the group
- Charlie updates and commits
- Charlie removes Bob

RESOURCES

- [RFC 9420](#)
- <https://www.ietf.org/archive/id/draft-ietf-mls-architecture-15.html>
- [TreeKEM](#)
- [TreeSync](#)
- [OpenMLS](#)
- [Security Analysis of the MLS Key Derivation](#)
- [End-to-End Encrypted Group Chats with MLS: Design, Implementation and Verification](#)
- [Three \(thousand\) may keep a secret](#)