End-to-End Encrypted Group Chats with MLS: Design, Implementation and Verification

MLS

TODO: insert here an easy to understand yet impactful figure representing MLS (don't forget to fill this in before the final presentation!)

Théophile Wallez, *Inria Paris*

FRANCE 2030

Inria

Disclaimer

This talk is about my research journey during my PhD, with two papers.

TreeSync:
Authenticated Group Management for
Messaging Layer Security

https://www.usenix.org/conference/usenixsecurity23/presentation/wallez
(USENIX Security '23, Internet defense prize and distinguished paper award!)

Comparse:
Provably Secure Formats
for Cryptographic Protocols

https://eprint.iacr.org/2023/1390
(ACM CCS 2023)

TreeSync: Authenticated Group Management for Messaging Layer Security

TODO: insert here an easy to understand yet impactful figure representing MLS (don't forget to fill this in before the final presentation!)

Théophile Wallez, *Inria Paris*
Jonathan Protzenko, *Microsoft Research*
Benjamin Beurdouche, *Inria Paris, Mozilla*
Karthikeyan Bhargavan, *Inria Paris, Cryspen*

What is Messaging Layer Security (MLS)

Secure group messaging

Secure group messaging

https://www.nytimes.com/2020/06/11/style/signal-messaging-app-encryption-protests.html

The New York Times

Signal Downloads Are Way Up Since the Protests Began

Organizers and demonstrators say they feel safer communicating with end-to-end encryption.

Secure group messaging

https://www.nytimes.com/2020/06/11/style/signal-messaging-app-encryption-protests.html

The New York Times

Signal Downloads Are Way Up Since the Protests Began

Organizers and demonstrators say they feel safer communicating with end-to-end encryption.

time

Secure group messaging

https://www.nytimes.com/2020/06/11/style/signal-messaging-app-encryption-protests.html

The New York Times

Signal Downloads Are Way Up Since the Protests Began

Organizers and demonstrators say they feel safer communicating with end-to-end encryption.

Forward secrecy

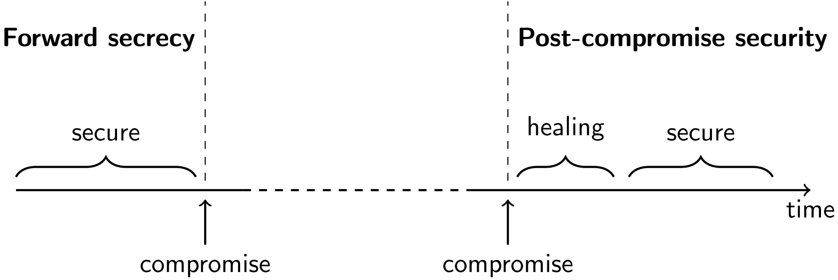
Secure group messaging

https://www.nytimes.com/2020/06/11/style/signal-messaging-app-encryption-protests.html


The New York Times

Signal Downloads Are Way Up Since the Protests Began

Organizers and demonstrators say they feel safer communicating with end-to-end encryption.



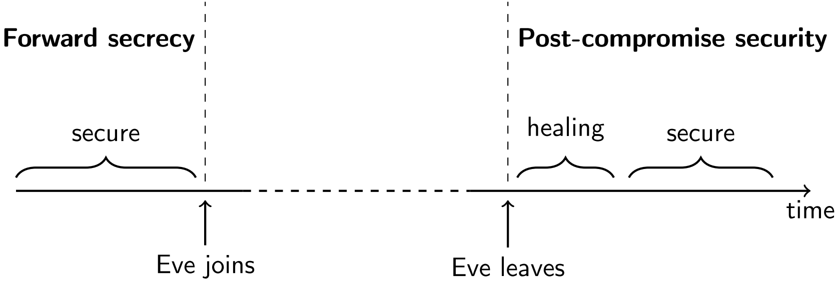
Secure group messaging

https://www.nytimes.com/2020/06/11/style/signal-messaging-app-encryption-protests.html

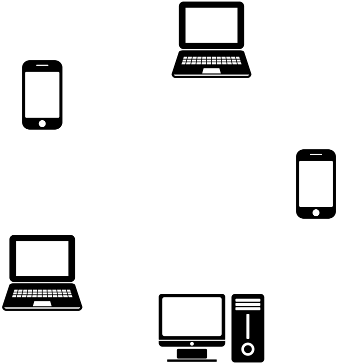
The New York Times

Signal Downloads Are Way Up Since the Protests Began

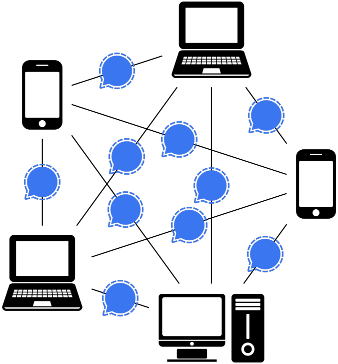
Organizers and demonstrators say they feel safer communicating with end-to-end encryption.



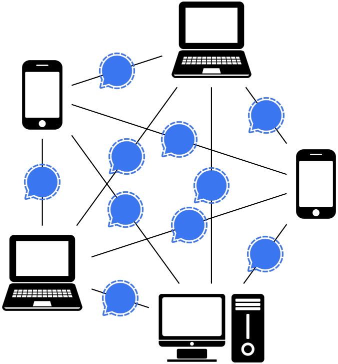
State of the art, before MLS



State of the art, before MLS

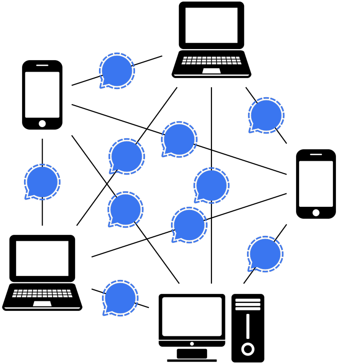


State of the art, before MLS



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

State of the art, before MLS



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

IETF

MLS

RFC 9420

Design constraints:
Secure, <u>efficient</u>, asynchronous, dynamic groups

A complex problem

A complex problem

https://nebuchadnezzar-megolm.github.io/

[matrix]

Upgrade now to address E2EE vulnerabilities in matrix-js-sdk, matrix-ios-sdk and matrix-android-sdk2

28.09.2022 17:41 — Security — Matthew Hodgson, Denis Kasak, Matrix Cryptography Team, Matrix Security Team

A complex problem

https://nebuchadnezzar-megolm.github.io/

matrix

Upgrade now to address E2EE vulnerabilities in matrix-js-sdk, matrix-ios-sdk and matrix-android-sdk2

28.09.2022 17:41 — Security — Matthew Hodgson, Denis Kasak, Matrix Cryptography Team, Matrix Security Team

Many performance / security tradeoffs

(https://inria.hal.science/hal-02425229/)

Protocol	Create		Add			Remove		Update		Group	Update	Remove
	Send	Recv	Send	Recv	New	Send	Recv	Send	Recv	Agreement	PPCS	PACS
Sender Keys [18]	N^2	N	1	1	N	-	-	-	-	No	No	No
Chained mKEM+	N	1	1	1	1	N	1	N	1	Yes	Yes	Yes
2-KEM Trees+	N	$log(N)$	$log(N)$	$log(N)$	$log(N)$	$log(N)$	$log(N)$	$log(N)$	$log(N)$	Yes	Yes	No
ART [7]	N	$log(N)$	$log(N)$	$log(N)$	$log(N)$	-	-	$log(N)$	$log(N)$	Yes	Yes	No
TreeKEM+	N	$log(N)$	$log(N)$	1	1	$log(N)$	1	$log(N)$	1	Yes	Yes	No
TreeKEM_B+	N	1	1	1	1	$log(N)..N$	1	$log(N)..N$	1	Yes	Yes	No*
TreeKEM_{B+S}+	N	1	1	1	N	$log(N)..N$	1	$log(N)..N$	1	Yes	Yes	Yes

Protocol — Performance — Security

A complex RFC

Table of Contents

[Page 164]

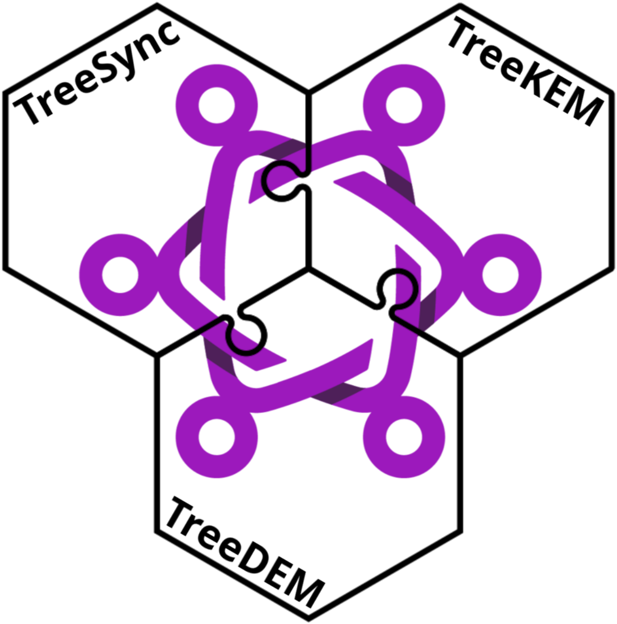
1,233 commits

0 Open ✓ 582 Closed

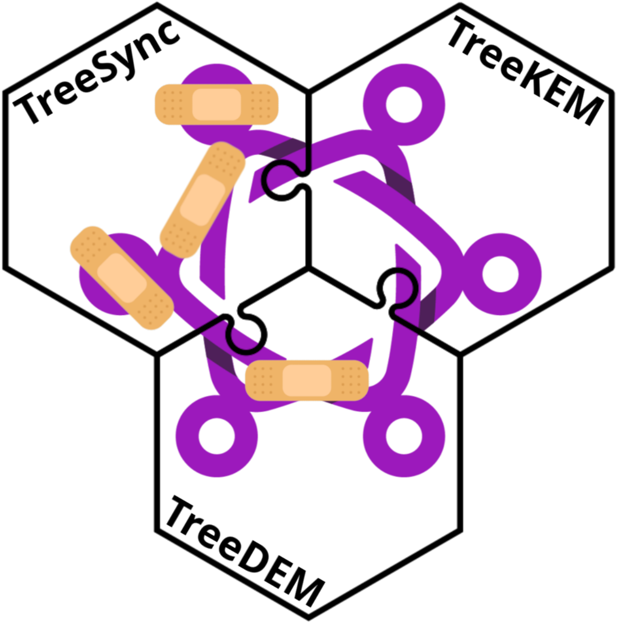
Quick interlude: our contributions

Contributions TL;DR

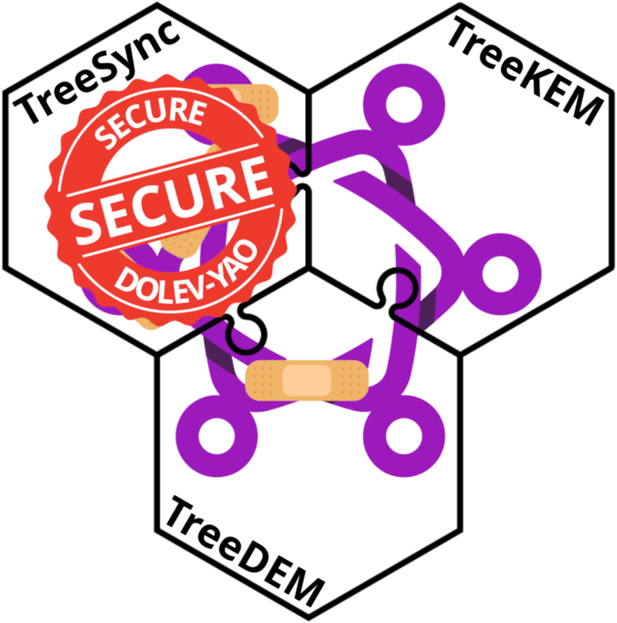
Contributions TL;DR



Contributions TL;DR



Contributions TL;DR



Contribution: Methodology

MLS

↓

F* specification

Contribution: Methodology

MLS

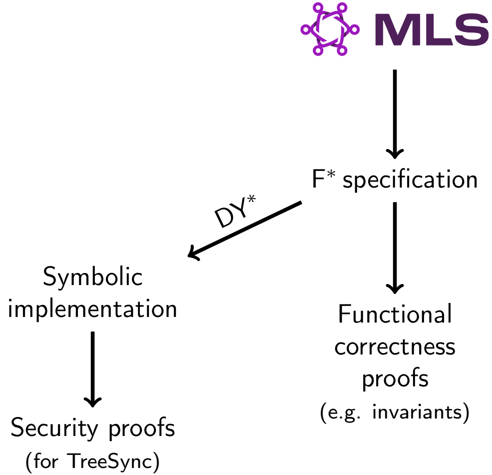
↓

F* specification

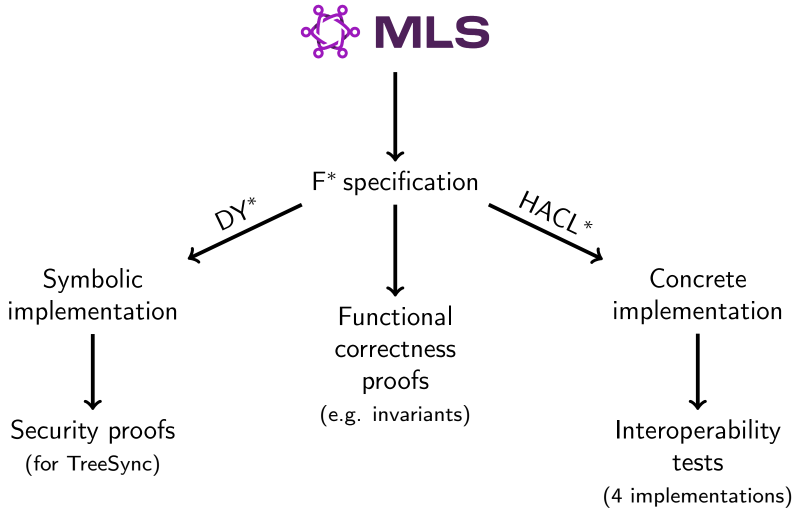
↓

Functional
correctness
proofs
(e.g. invariants)

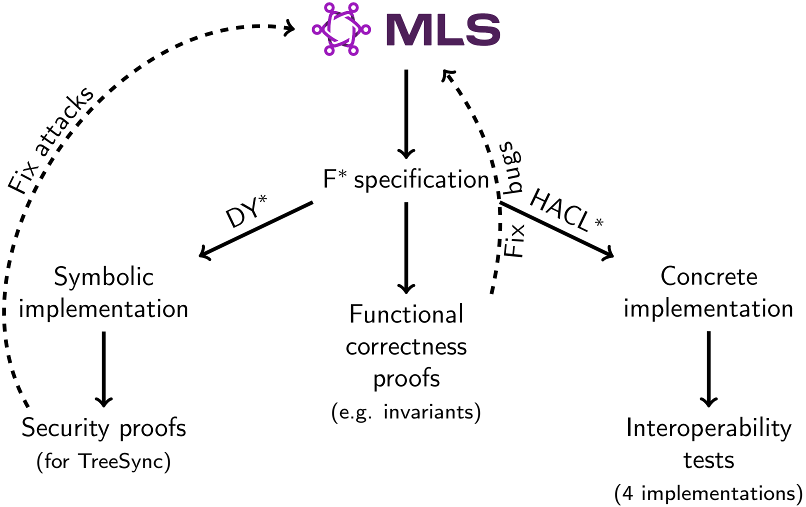
Contribution: Methodology



Contribution: Methodology

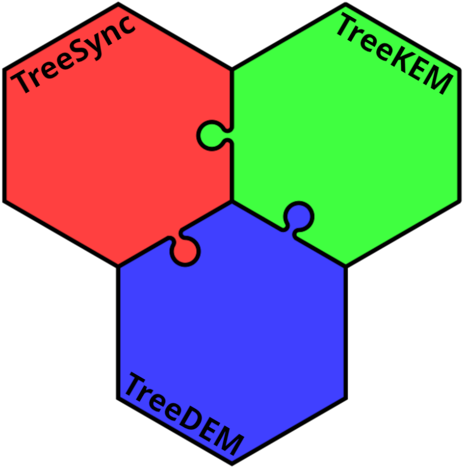


Contribution: Methodology



A tour of MLS

MLS decomposition



TreeSync: authenticated group synchronization
TreeKEM: efficient continuous group key establishment
TreeDEM: forward secure group messaging

Disclaimer

The following explanations do the following assumption:

- there are 2^n participants in the group.

In particular, no dynamic groups (i.e. no add / remove).

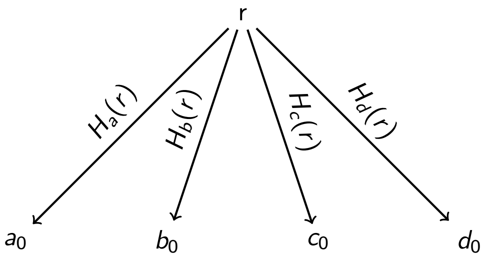
Why:

- avoid consuming too much brainpower budget :)
- still give the core ideas behind MLS

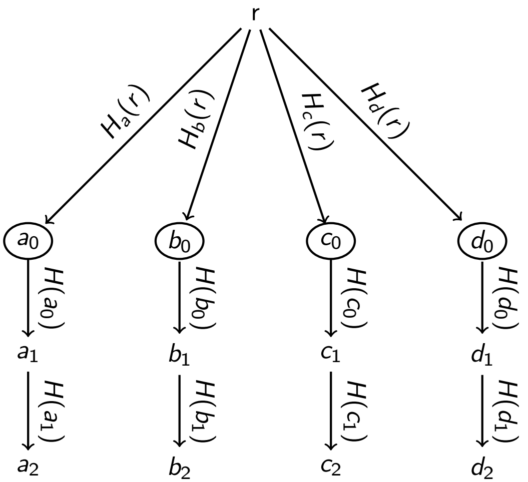
TreeDEM

r

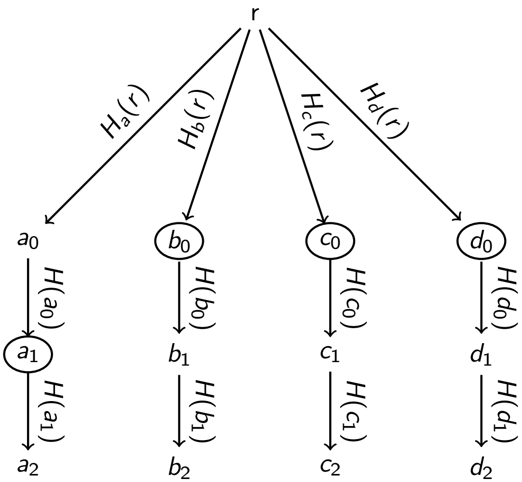
TreeDEM



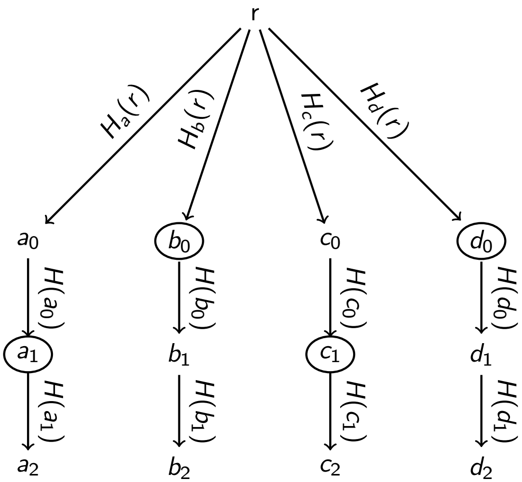
TreeDEM


r
$H_a(r)$
$H_b(r)$
$H_c(r)$
$H_d(r)$
a_0
b_0
c_0
d_0
$H(a_0)$
$H(b_0)$
$H(c_0)$
$H(d_0)$
a_1
b_1
c_1
d_1
$H(a_1)$
$H(b_1)$
$H(c_1)$
$H(d_1)$
a_2
b_2
c_2
d_2

TreeDEM

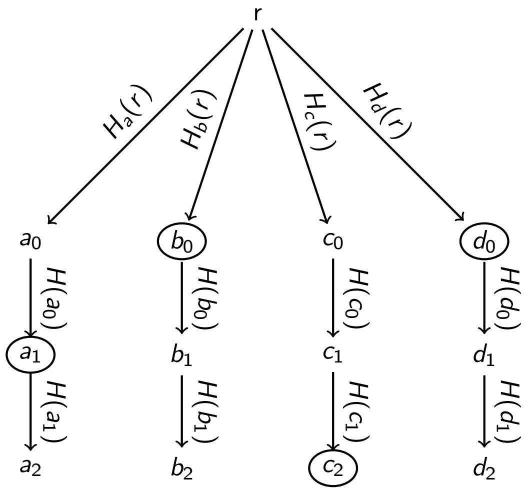

r
$H_a(r)$
$H_b(r)$
$H_c(r)$
$H_d(r)$
a_0
b_0
c_0
d_0
$H(a_0)$
$H(b_0)$
$H(c_0)$
$H(d_0)$
a_1
b_1
c_1
d_1
$H(a_1)$
$H(b_1)$
$H(c_1)$
$H(d_1)$
a_2
b_2
c_2
d_2

TreeDEM

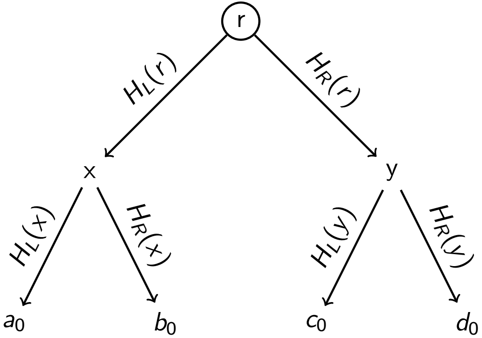


r
$H_a(r)$
$H_b(r)$
$H_c(r)$
$H_d(r)$
a_0
b_0
c_0
d_0
$H(a_0)$
$H(b_0)$
$H(c_0)$
$H(d_0)$
a_1
b_1
c_1
d_1
$H(a_1)$
$H(b_1)$
$H(c_1)$
$H(d_1)$
a_2
b_2
c_2
d_2

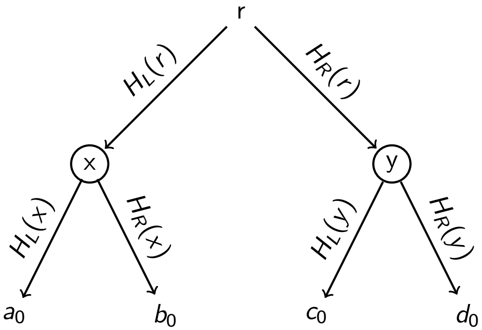
TreeDEM


r
$H_a(r)$
$H_b(r)$
$H_c(r)$
$H_d(r)$
a_0
b_0
c_0
d_0
$H(a_0)$
$H(b_0)$
$H(c_0)$
$H(d_0)$
a_1
b_1
c_1
d_1
$H(a_1)$
$H(b_1)$
$H(c_1)$
$H(d_1)$
a_2
b_2
c_2
d_2

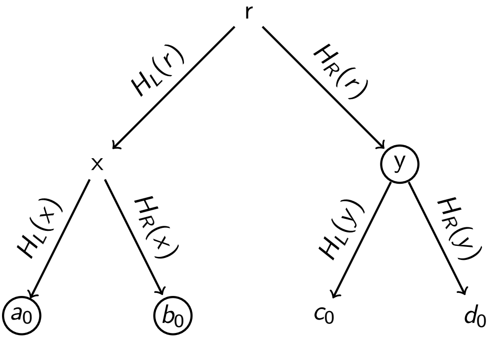
TreeDEM. . . with a tree



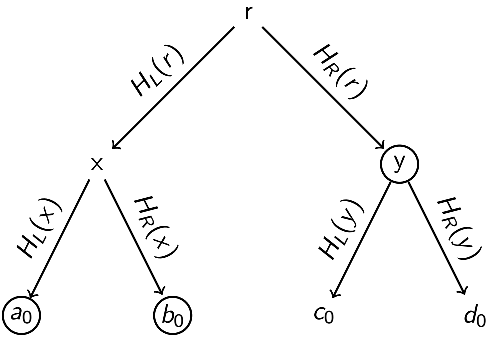
TreeDEM. . . with a tree



TreeDEM. . . with a tree

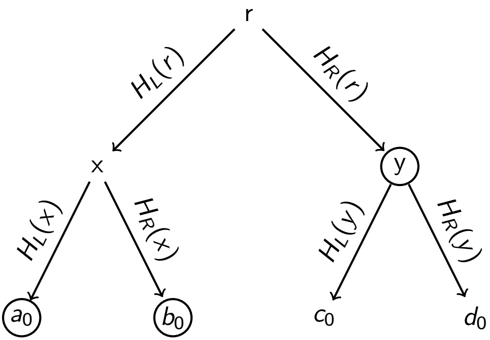


TreeDEM. . . with a tree



Root key to participant key (worst case): $O(\log(n))$

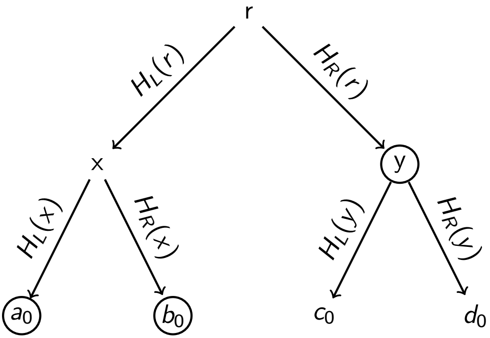
TreeDEM...with a tree



Root key to participant key (worst case): $O(\log(n))$

But:
Root key to all participant keys (worst case): $O(n)$

TreeDEM. . . with a tree



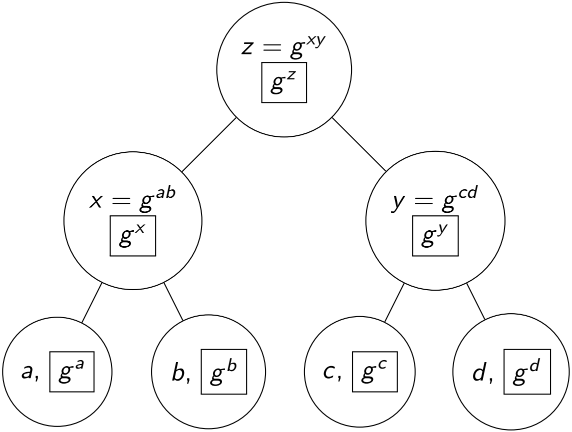
Root key to participant key (worst case): $O(\log(n))$

But:
Root key to all participant keys (worst case): $O(n)$

Hence:
Root key to participant key (amortized): $O(1)$

TreeKEM, the initial idea (ART)

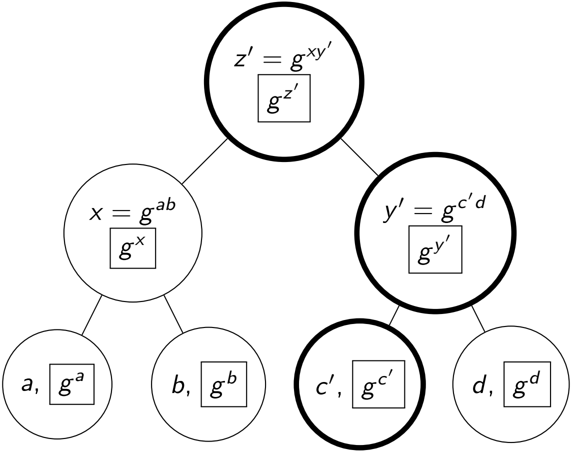
Idea: do a tree of Diffie-Hellman.
Invariant: private key of a node known exactly by its subtree.



TreeKEM, the initial idea (ART)

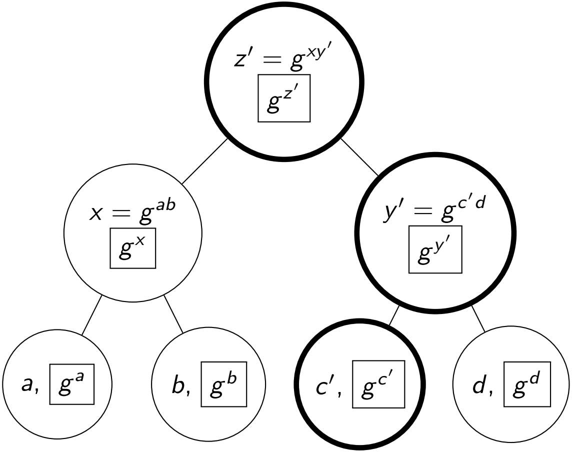
Idea: do a tree of Diffie-Hellman.

Invariant: private key of a node known exactly by its subtree.



TreeKEM, the initial idea (ART)

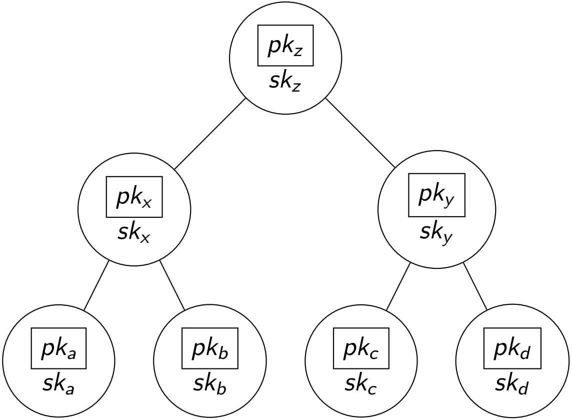
Idea: do a tree of Diffie-Hellman.
Invariant: private key of a node known exactly by its subtree.



Send complexity: $O(\log(n))$ asymetric operations
Receive complexity: $O(\log(n))$ asymetric operations

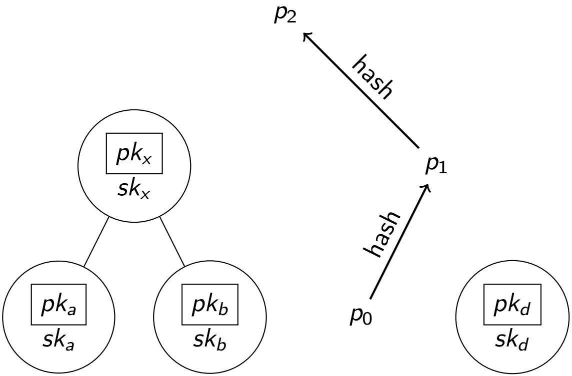
TreeKEM, toward the final design

Idea: rely on asymetric encryption (HPKE) and hashes (HKDF).
Invariant: private key of a node known exactly by its subtree.
Three steps: generate, encrypt, publish.



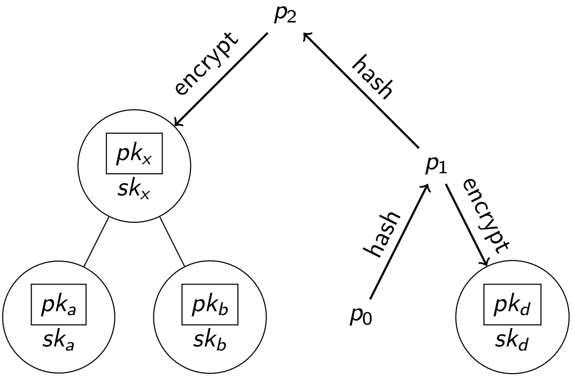
TreeKEM, toward the final design

Idea: rely on asymetric encryption (HPKE) and hashes (HKDF).
Invariant: private key of a node known exactly by its subtree.
Three steps: **generate**, encrypt, publish.



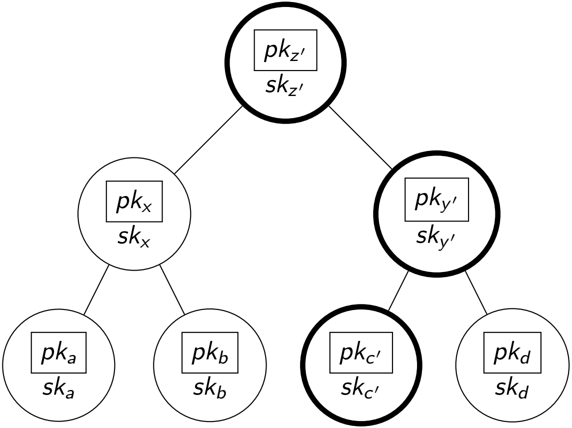
TreeKEM, toward the final design

Idea: rely on asymetric encryption (HPKE) and hashes (HKDF).
Invariant: private key of a node known exactly by its subtree.
Three steps: generate, **encrypt**, publish.



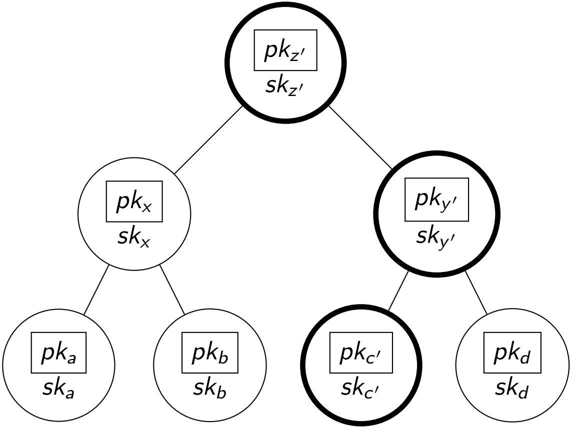
TreeKEM, toward the final design

Idea: rely on asymetric encryption (HPKE) and hashes (HKDF).
Invariant: private key of a node known exactly by its subtree.
Three steps: generate, encrypt, **publish**.



TreeKEM, toward the final design

Idea: rely on asymetric encryption (HPKE) and hashes (HKDF).
Invariant: private key of a node known exactly by its subtree.
Three steps: generate, encrypt, publish.

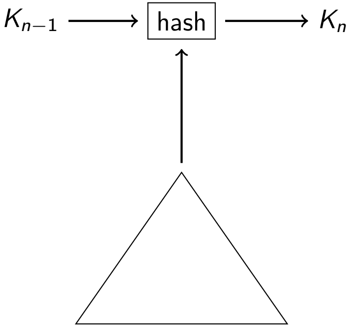


Send complexity: $O(\log(n))$ asymetric operations
Receive complexity: only 1 asymetric operation!

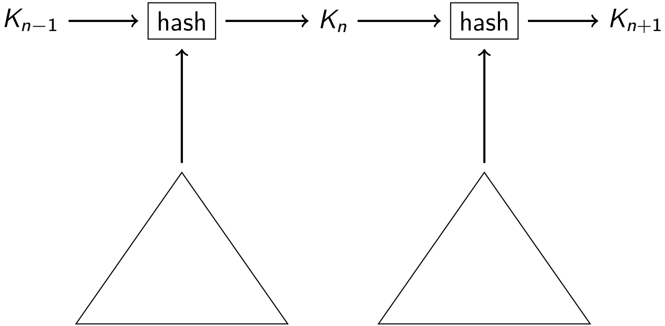
TreeKEM, with a key schedule for forward secrecy

K_{n-1}

TreeKEM, with a key schedule for forward secrecy



TreeKEM, with a key schedule for forward secrecy



TreeSync: why?

Alice joins a secure group, and receive a tree of public keys.
How does she makes sure those keys are not attacker-controlled?

TreeSync: why?

Alice joins a secure group, and receive a tree of public keys.
How does she makes sure those keys are not attacker-controlled?

How does she makes sure who is in the group?
Can the attacker be in the group without her knowledge?
Is Bob really Bob, or is it the attacker somehow?

TreeSync: why?

Alice joins a secure group, and receive a tree of public keys.
How does she makes sure those keys are not attacker-controlled?

How does she makes sure who is in the group?
Can the attacker be in the group without her knowledge?
Is Bob really Bob, or is it the attacker somehow?

TreeSync solves these problems by authenticating TreeKEM's state.
In particular:

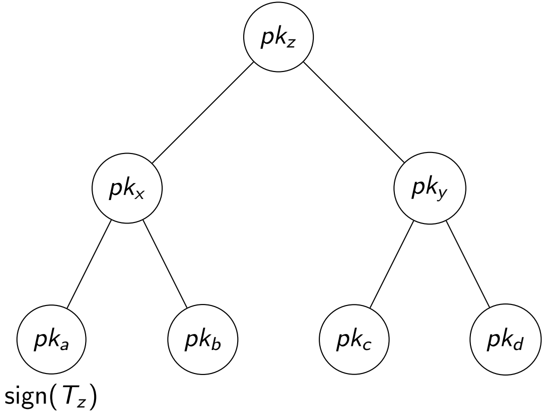
- authenticates all public keys, along with their recipients
- authenticates the roster, ensuring **group membership agreement**

Before the integration of TreeSync in MLS,
several man-in-the-middle-like attacks were found in MLS.
With TreeSync, this class of attacks are not possible anymore.

TreeSync: (naive) attempt 1

When a participant update keys, it signs the new tree.

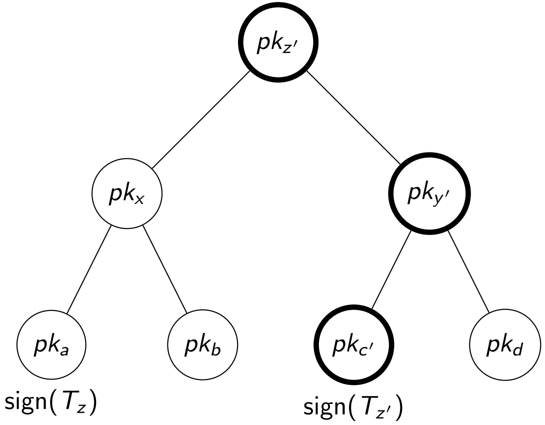
$T_z =$



TreeSync: (naive) attempt 1

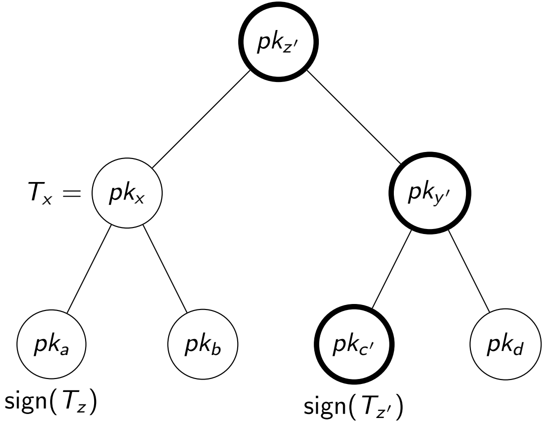
When a participant update keys, it signs the new tree.

$T_{z'} =$



TreeSync: (naive) attempt 1

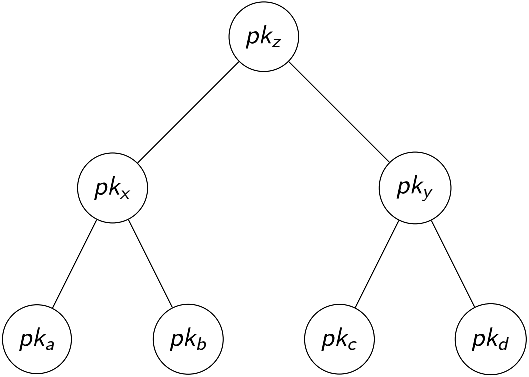
When a participant update keys, it signs the new tree.
$T_{z'} =$



Now, Alice's signature is unintelligible!
As a result, T_x not authenticated by Alice anymore.

TreeSync: attempt 2

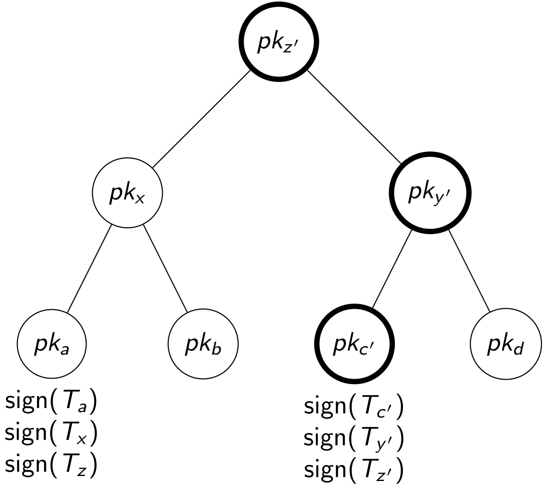
When a participant update keys, it signs the every modified subtree.



sign(T_a)
sign(T_x)
sign(T_z)

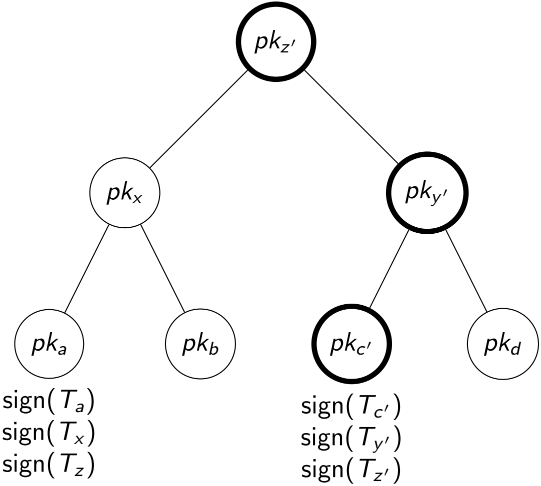
TreeSync: attempt 2

When a participant update keys, it signs the every modified subtree.



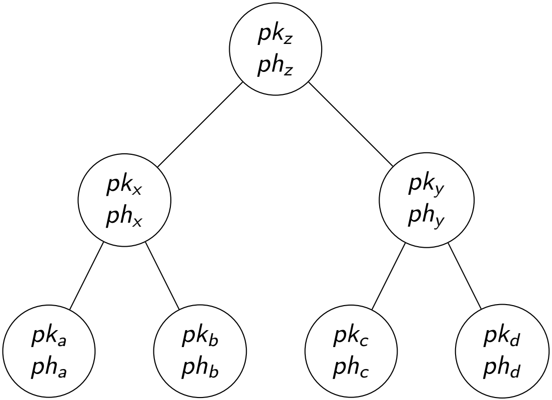
TreeSync: attempt 2

When a participant update keys, it signs the every modified subtree.



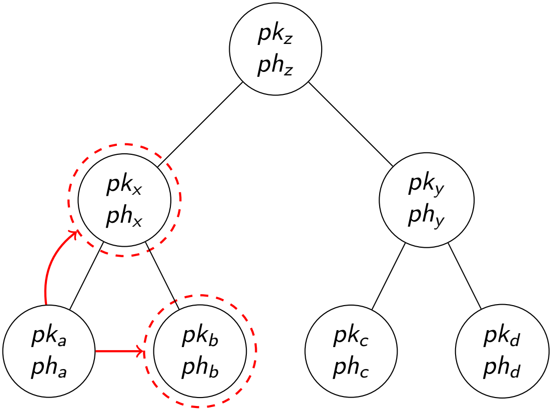
Invariant: every subtree is signed by one of the leaves under it.
Complexity: requires log(n) signatures in each leaf :(

TreeSync: final attempt



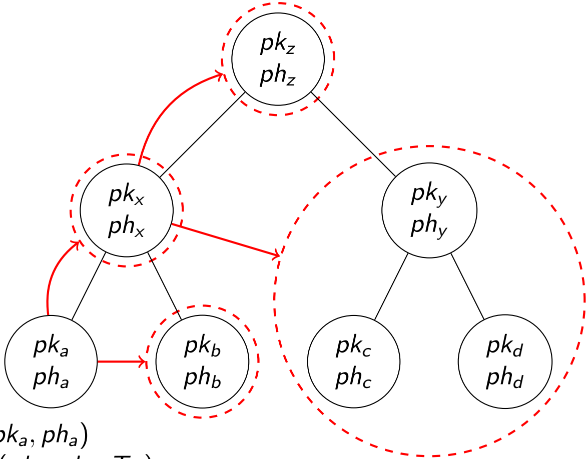
$$\text{sign}(pk_a, ph_a)$$
$$ph_a = hash(pk_x, ph_x, T_B)$$
$$ph_x = hash(pk_z, ph_z, T_Y)$$

TreeSync: final attempt



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

TreeSync: final attempt

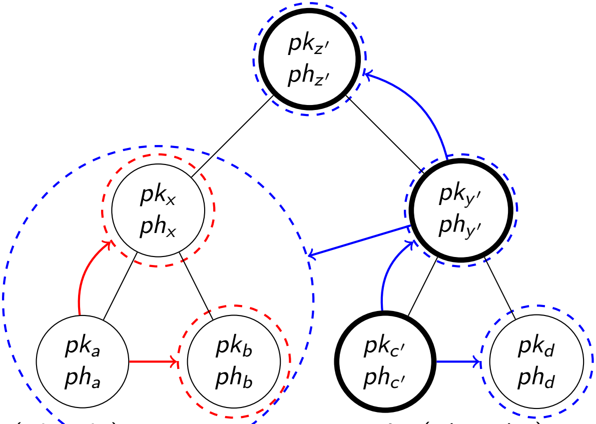


$\mathrm{sign}(pk_a, ph_a)$

$ph_a = hash(pk_x, ph_x, T_B)$

$ph_x = hash(pk_z, ph_z, T_Y)$

TreeSync: final attempt



$\text{sign}(pk_a, ph_a)$

$ph_a = hash(pk_x, ph_x, T_B)$

$ph_x = hash(pk_z, ph_z, T_Y)$

$\text{sign}(pk_{c'}, ph_{c'})$

$ph_{c'} = hash(pk_{y'}, ph_{y'}, T_D)$

$ph_{y'} = hash(pk_{z'}, ph_{z'}, T_X)$

Invariant: every subtree is linked by parent-hash to one of its leaves.
Complexity: requires only 1 signature in each leaf!

2^n participants: what did we miss?

Blank leaves: for non-power-of-two number of participants

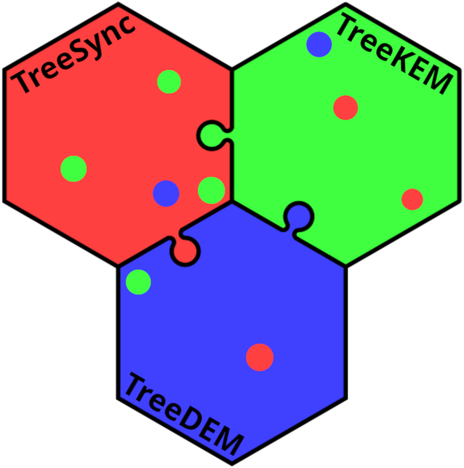
Blank nodes: remove participants and erase secrets they know

Unmerged leaves: add new participants efficiently

Filtered nodes: optimize away nodes that are redundant

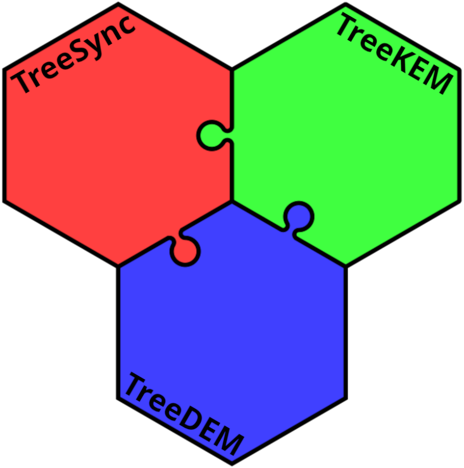
Contributions on TreeSync

Contribution: Modularizing MLS



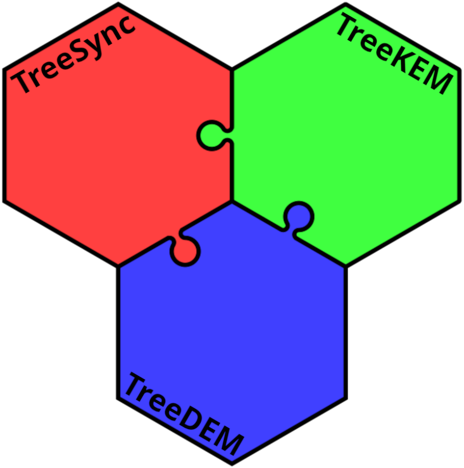
TreeSync: authenticated group synchronization
TreeKEM: efficient continuous group key establishment
TreeDEM: forward secure group messaging

Contribution: Modularizing MLS



TreeSync: authenticated group synchronization
TreeKEM: efficient continuous group key establishment
TreeDEM: forward secure group messaging

Contribution: Modularizing MLS



TreeSync: authenticated group synchronization
TreeKEM: efficient continuous group key establishment
TreeDEM: forward secure group messaging

Contribution: Fixing TreeSync's invariants

```
def join_group(group):
    if well_formed(group):
        # ...
    else:
        raise MalformedGroupException
```

Desirable property: well_formed is an invariant under group modifications.

Contribution: Fixing TreeSync's invariants

```
def join_group(group):
    if well_formed(group):
        # ...
    else:
        raise MalformedGroupException
```

Desirable property: well_formed is an invariant under group modifications.

Actually, a well-formed group could become malformed!

Contribution: Fixing TreeSync's invariants

```
def join_group(group):
    if well_formed(group):
        # ...
    else:
        raise MalformedGroupException
```

Desirable property: well_formed is an invariant under group modifications.

Actually, a well-formed group could become malformed!

Contribution: Fixing TreeSync's guarantees

7.9. Parent Hashes

While tree hashes summarize the state of a tree at point in time, parent hashes capture information about how keys in the tree were populated.

path. When a client computes an UpdatePath (as defined in Section 7.5), it computes and signs a parent hash that summarizes the state of the tree after the UpdatePath has been applied. These summaries are constructed in a chain from the root to the member's

As a result, the signature over the parent hash in each member's leaf effectively signs the subtree of the tree that hasn't been changed since that leaf was last changed in an UpdatePath. A new member joining the group uses these parent hashes to verify that the parent

Contribution: Fixing TreeSync's guarantees

7.9. Parent Hashes

While tree hashes summarize the state of a tree at point in time, parent hashes capture information about how keys in the tree were populated.

path. When a client computes an UpdatePath (as defined in Section 7.5), it computes and signs a parent hash that summarizes the state of the tree after the UpdatePath has been applied. These summaries are constructed in a chain from the root to the member's

As a result, the signature over the parent hash in each member's leaf effectively signs the subtree of the tree that hasn't been changed since that leaf was last changed in an UpdatePath. A new member joining the group uses these parent hashes to verify that the parent

Problem 1: Guarantees described in imprecise prose.

Contribution: Fixing TreeSync's guarantees

7.9. Parent Hashes

While tree hashes summarize the state of a tree at point in time, parent hashes capture information about how keys in the tree were populated.

path. When a client computes an UpdatePath (as defined in Section 7.5), it computes and signs a parent hash that summarizes the state of the tree after the UpdatePath has been applied. These summaries are constructed in a chain from the root to the member's

As a result, the signature over the parent hash in each member's leaf effectively signs the subtree of the tree that hasn't been changed since that leaf was last changed in an UpdatePath. A new member joining the group uses these parent hashes to verify that the parent

Problem 1: Guarantees described in imprecise prose.

Problem 2: Guarantees not actually met by parent hash!

Contribution: Fixing TreeSync's guarantees

7.9. Parent Hashes

While tree hashes summarize the state of a tree at point in time, parent hashes capture information about how keys in the tree were populated.

---

path. When a client computes an UpdatePath (as defined in Section 7.5), it computes and signs a parent hash that summarizes the state of the tree after the UpdatePath has been applied. These summaries are constructed in a chain from the root to the member's

---

As a result, the signature over the parent hash in each member's leaf effectively signs the subtree of the tree that hasn't been changed since that leaf was last changed in an UpdatePath. A new member joining the group uses these parent hashes to verify that the parent

Problem 1: Guarantees described in imprecise prose.

Problem 2: Guarantees not actually met by parent hash!

Contribution: Fixing a signature ambiguity attack

TreeSync

$$\text{sig} = \text{sign}(\text{sk}, \text{serialize}_{T1}(\text{msg}_1))$$

Contribution: Fixing a signature ambiguity attack

TreeSync

$\text{sig} = \text{sign}(\text{sk}, \underline{\text{serialize}}_{T1}(\text{msg}_1))$

Contribution: Fixing a signature ambiguity attack

TreeSync

$\text{sig} = \text{sign}(\text{sk}, \text{serialize}_{T1}(\text{msg}_1))$
$\text{verify}(\text{pk}, \text{sig}, \text{serialize}_{T1}(\text{msg}_1))$

Contribution: Fixing a signature ambiguity attack

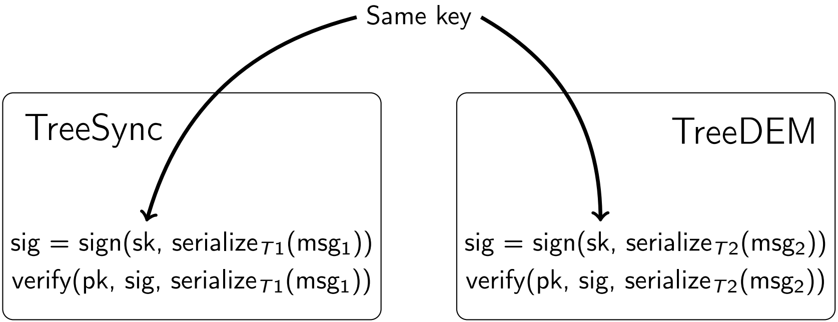
TreeSync

$\text{sig} = \text{sign}(\text{sk}, \text{serialize}_{T1}(\text{msg}_1))$
$\text{verify}(\text{pk}, \text{sig}, \text{serialize}_{T1}(\text{msg}_1))$

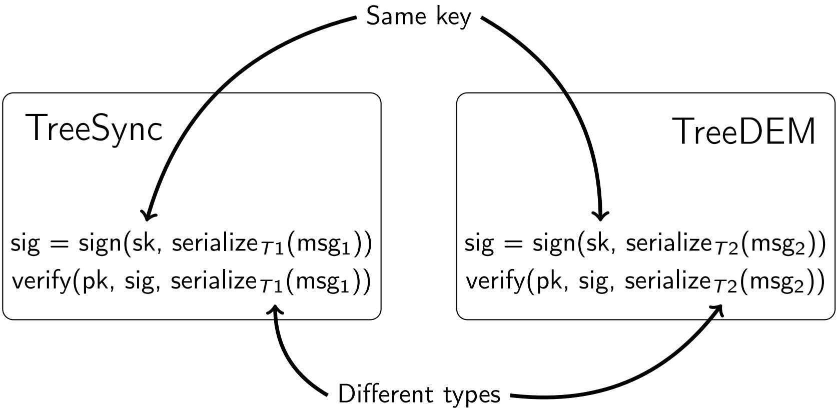
TreeDEM

$\text{sig} = \text{sign}(\text{sk}, \text{serialize}_{T2}(\text{msg}_2))$
$\text{verify}(\text{pk}, \text{sig}, \text{serialize}_{T2}(\text{msg}_2))$

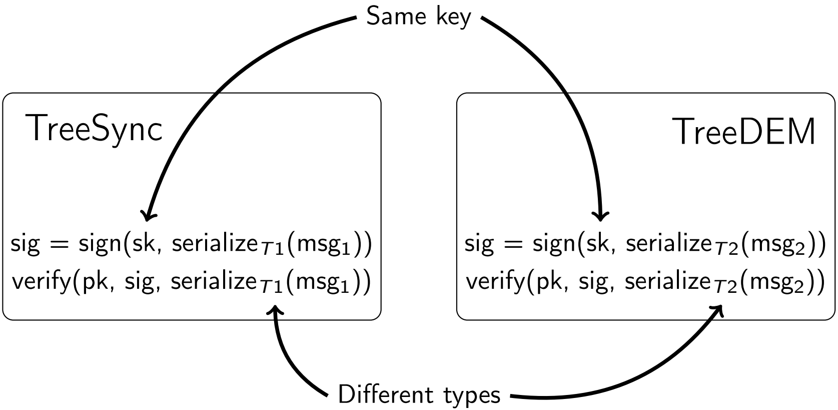
Contribution: Fixing a signature ambiguity attack



Contribution: Fixing a signature ambiguity attack

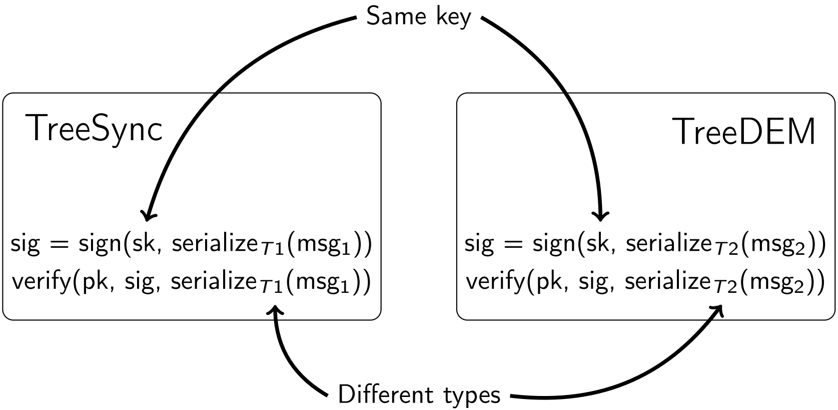


Contribution: Fixing a signature ambiguity attack



What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T1}(\text{msg}_1) = \text{serialize}_{T2}(\text{msg}_2)$?

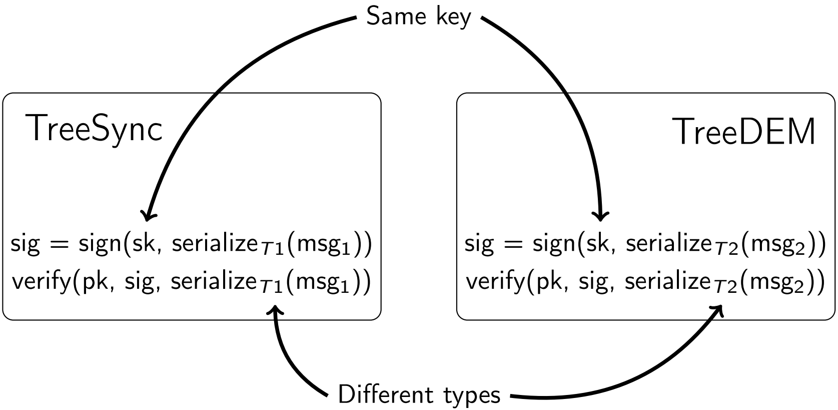
Contribution: Fixing a signature ambiguity attack



What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T1}(\text{msg}_1) = \text{serialize}_{T2}(\text{msg}_2)$?

Possible attack:
TreeDEM signature could be used to forge a signature in TreeSync!

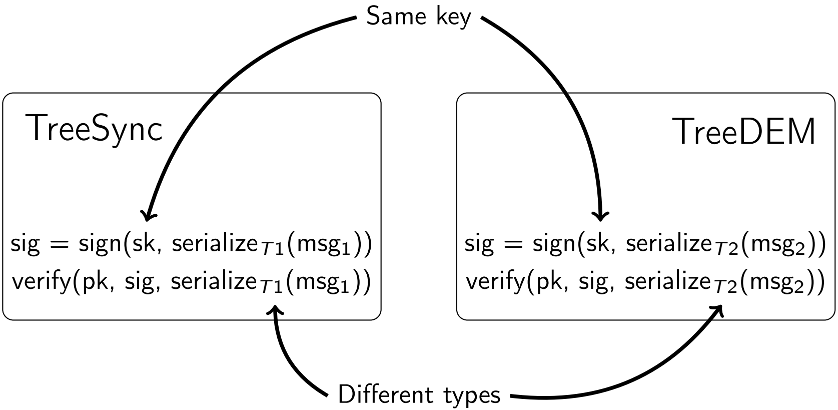
Contribution: Fixing a signature ambiguity attack



What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T1}(\text{msg}_1) = \text{serialize}_{T2}(\text{msg}_2)$?

Possible attack:
TreeDEM signature could be used to forge a signature in TreeSync!

Contribution: Fixing a signature ambiguity attack

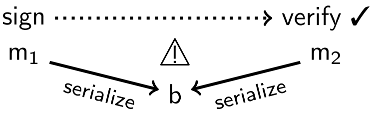


What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T1}(\text{msg}_1) = \text{serialize}_{T2}(\text{msg}_2)$?

Possible attack:
TreeDEM signature could be used to forge a signature in TreeSync!

Attack found by doing proofs on a bit-precise specification, thanks to executability and interoperability tests.

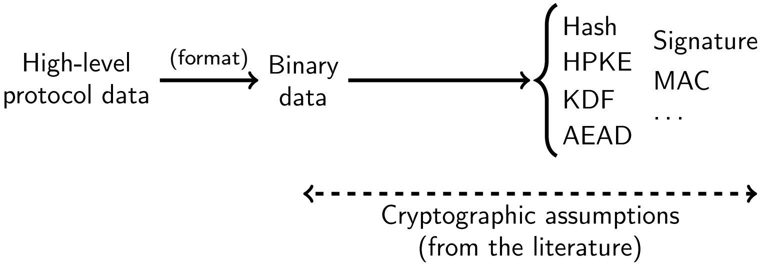
Comparse:
Provably Secure Formats for Cryptographic Protocols



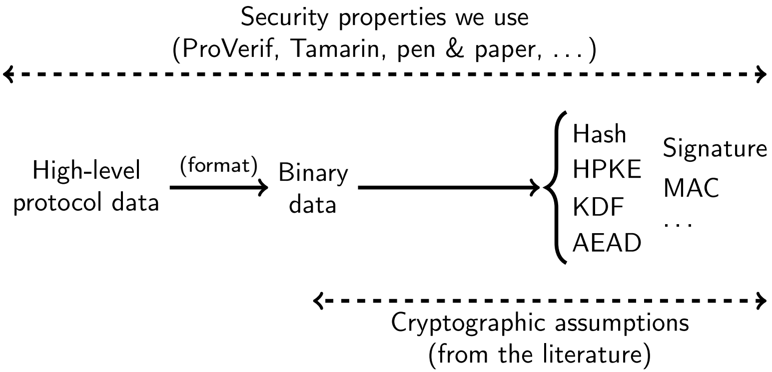
Théophile Wallez, *Inria Paris*
Jonathan Protzenko, *Microsoft Research*
Karthikeyan Bhargavan, *Inria Paris, Cryspen*

Security critical formats are omnipresent

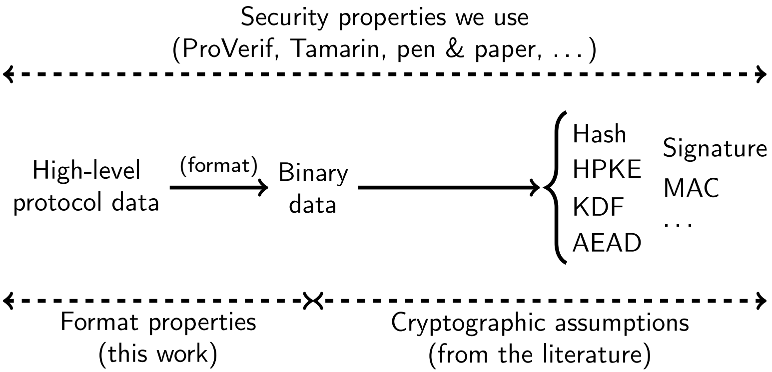
Security critical formats are omnipresent



Security critical formats are omnipresent



Security critical formats are omnipresent



Messages formats play a crucial role in cryptographic protocols security.

We study their impact in two steps:

1. study properties of message formats
2. show how format properties compose with cryptographic assumptions to obtain the security properties we use

Running example: signatures.

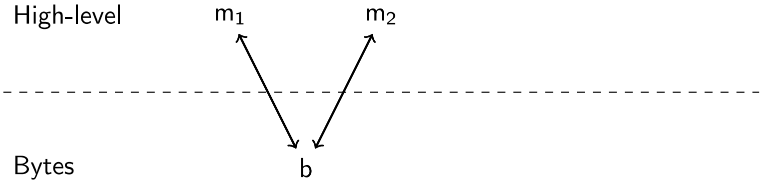
Message formats properties

High-level

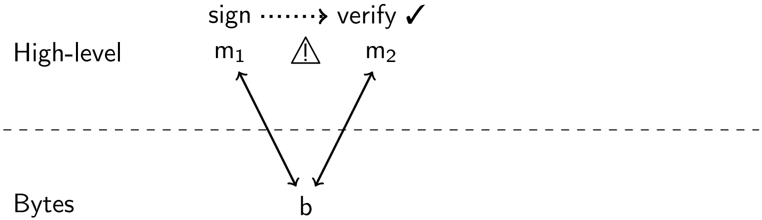
- -

Bytes

Message formats properties

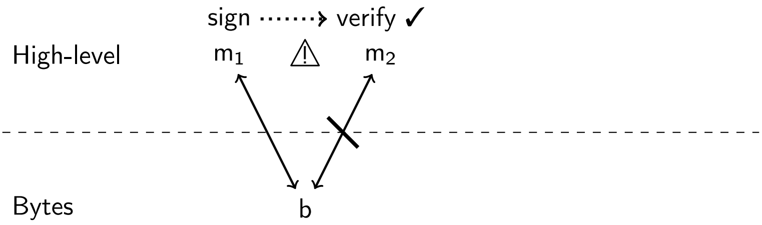


Message formats properties

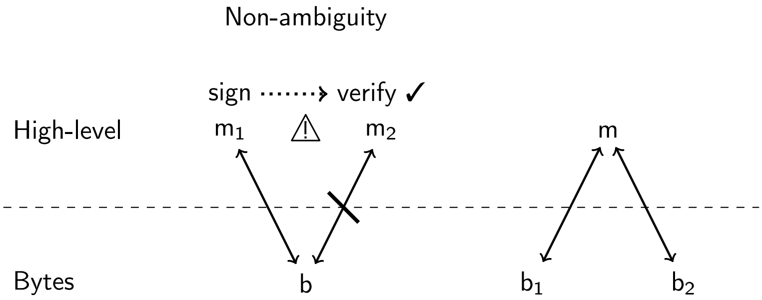


Message formats properties

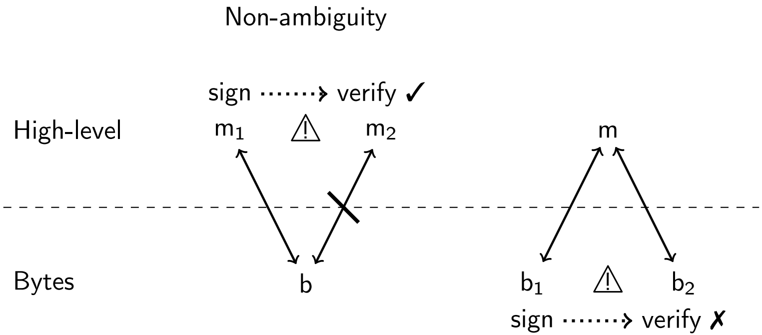
Non-ambiguity



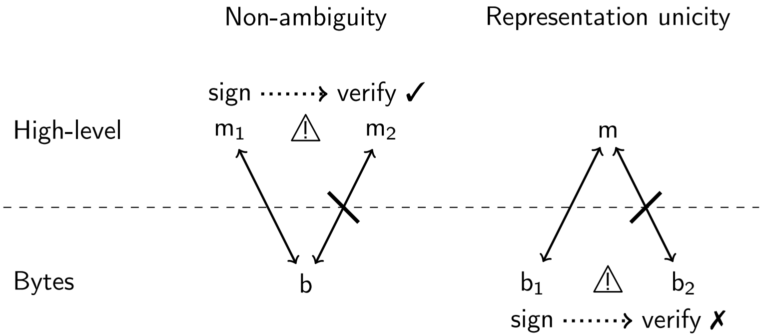
Message formats properties



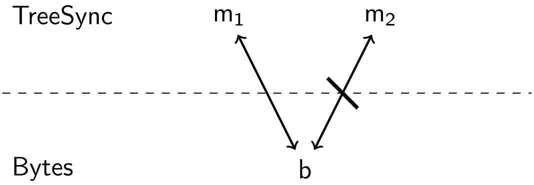
Message formats properties



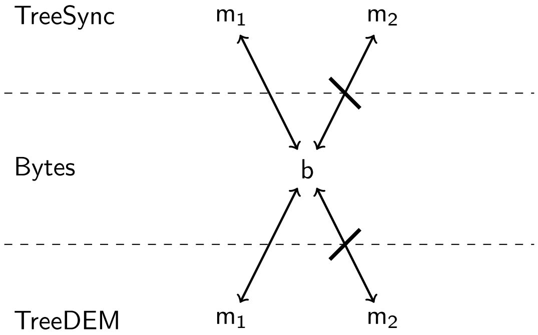
Message formats properties



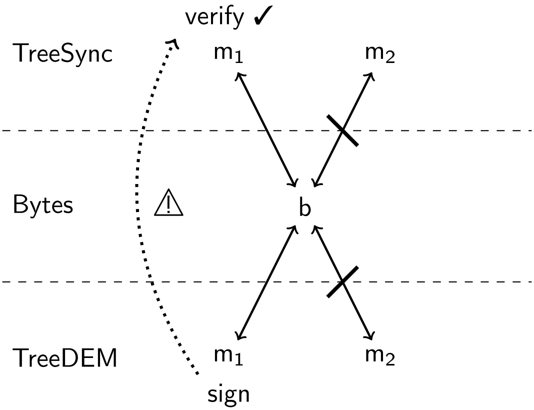
Message formats properties across protocols



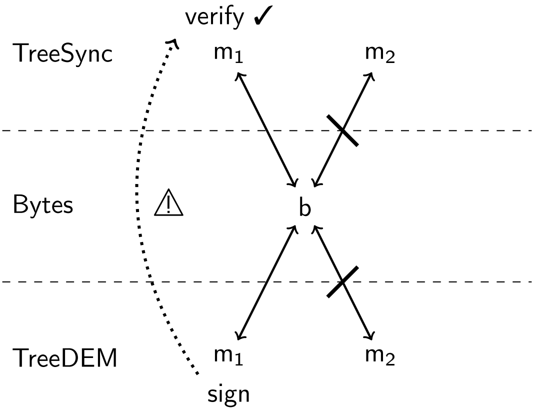
Message formats properties across protocols



Message formats properties across protocols

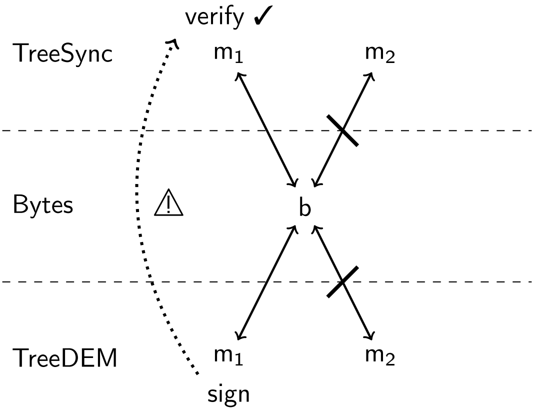


Message formats properties across protocols



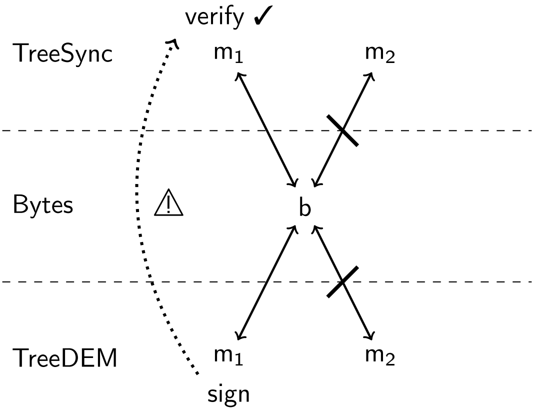
The problem: the meaning of b **depends** on the sub-protocol.

Message formats properties across protocols



The problem: the meaning of b **depends** on the sub-protocol.
One solution: add a tag in b to disambiguate the sub-protocol in use.

Message formats properties across protocols

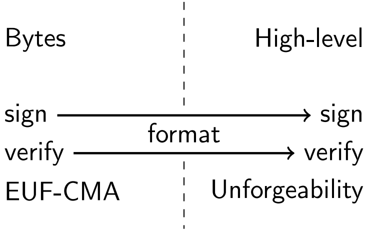


The problem: the meaning of b **depends** on the sub-protocol.
One solution: add a tag in b to disambiguate the sub-protocol in use.
The result: the meaning of b becomes **self-contained**.

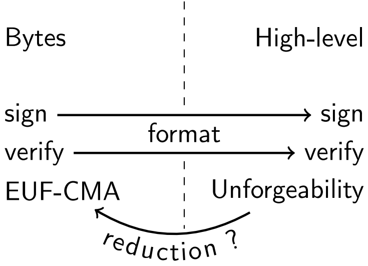
A rigorous approach to domain separation

Bytes	High-level
sign	
verify	
EUF-CMA	

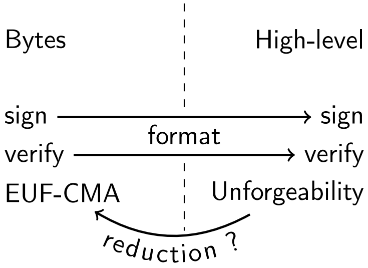
A rigorous approach to domain separation



A rigorous approach to domain separation

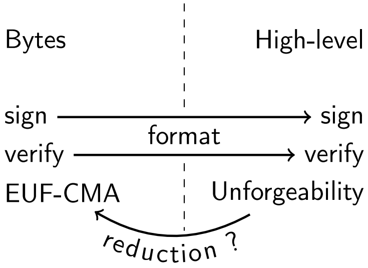


A rigorous approach to domain separation



Reduction if: this format is self-contained and non-ambiguous.

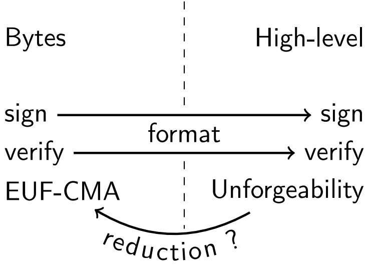
A rigorous approach to domain separation



Design discipline: Each signature key is used with a single format, and

Reduction if: this format is self-contained and non-ambiguous.

A rigorous approach to domain separation

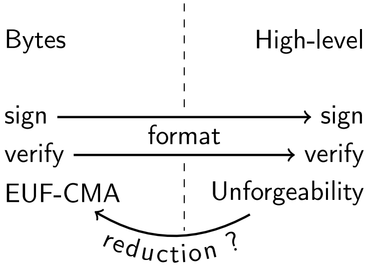


Design discipline: Each signature key is used with a single format, and

Reduction if: this format is self-contained and non-ambiguous.

Note 1: MLS draft 12 failed to obey this design discipline!
This weakness can be used in an attack.

A rigorous approach to domain separation



Design discipline: Each signature key is used with a single format, and

Reduction if: this format is self-contained and non-ambiguous.

Note 1: MLS draft 12 failed to obey this design discipline!
This weakness can be used in an attack.

Note 2: similar design discipline for MAC, AEAD, KDF, . . .

Final notes

Proof effort

Component	F* LoC	Verification time
Library code	836	1min30s
TreeSync	1274	4min30s
TreeKEM	396	1min
TreeDEM	1384	2min45s
High level API	1024	1min30s
Library proofs	1170	1min45s
TreeSync proofs	4018	13min30s
Tests	2782	2min45s
Total specification	4914	11min15s
Total proofs	5188	15min15s

Roughly two man-years of work, because many by-products to work on:

- Develop the methodology to treat such large protocols
- How to obtain a bit-precise specification
- Developed a framework for verified message formatting, both concrete and symbolic (Comparse, submitted at CCS 2023)
- A protocol during its standardization is a moving target

Conclusion

Our contributions:

- formally specify MLS decomposed into three sub-protocols: TreeSync, TreeKEM, and TreeDEM
- prove the security of TreeSync in the Dolev-Yao model
- do proofs on an executable, interoperable specification
- found design flaws and submitted fixes to the MLS Working Group
- (Comparse) shed light on the importance of formatting in cryptographic protocols

Future work: security proofs for TreeKEM and TreeDEM ; prove efficient implementations.

The MLS Working Group gladly welcomed these contributions, resulting in a fruitful collaboration.

`https://github.com/Inria-Prosecco/treesync`

`theophile.wallez@inria.fr`
`https://www.twal.org/`
@twallez

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE | ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED | ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

Proof sketch of TreeSync

Security proof, step 1: invariants

We prove many invariants on TreeSync (the well-formedness checks):

- Leaf signatures are valid
- Every node is linked by parent-hash to a node under it
- Things with unmerged leaves

Security proof, step 2: the parent-hash guarantee theorem

We define an equivalence relation on trees $\simeq$.

We prove the theorem:

$$\begin{array}{ccc} P_1 & & P_2 \\ \uparrow & & \uparrow \\ C_1 & \simeq & C_2 \end{array}$$

Security proof, step 2: the parent-hash guarantee theorem

We define an equivalence relation on trees $\simeq$.

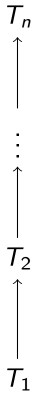
We prove the theorem:

$$\begin{array}{ccc} P_1 & \simeq & P_2 \\ \uparrow & & \uparrow \\ C_1 & \simeq & C_2 \end{array}$$

Security proof, step 3: signature invariant

We want to prove : every subtree is authenticated by one of its leaves.

Proof sketch:



Security proof, step 3: signature invariant

We want to prove : every subtree is authenticated by one of its leaves.

Proof sketch:

$$
\begin{array}{ccc}
T_n & & T'_n \\
\uparrow & & \uparrow \\
\vdots & & \vdots \\
\uparrow & & \uparrow \\
T_2 & & T'_2 \\
\uparrow & & \uparrow \\
T_1 & \simeq & T'_1
\end{array}
$$

Security proof, step 3: signature invariant

We want to prove : every subtree is authenticated by one of its leaves.

Proof sketch:

$$\begin{array}{ccc}
T_n & & T'_n \\
\uparrow & & \uparrow \\
\vdots & & \vdots \\
\uparrow & & \uparrow \\
T_2 & \simeq & T'_2 \\
\uparrow & & \uparrow \\
T_1 & \simeq & T'_1
\end{array}$$

Security proof, step 3: signature invariant

We want to prove : every subtree is authenticated by one of its leaves.

Proof sketch:

$$
\begin{array}{ccc}
T_n & \simeq & T'_n \\
\uparrow & & \uparrow \\
\vdots & & \vdots \\
\uparrow & & \uparrow \\
T_2 & \simeq & T'_2 \\
\uparrow & & \uparrow \\
T_1 & \simeq & T'_1
\end{array}
$$