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

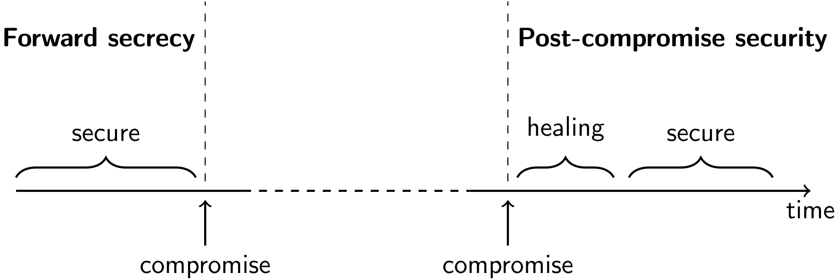
Secure group messaging

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.



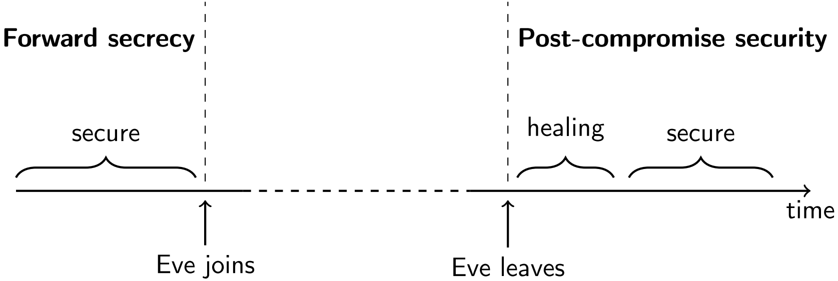
Secure group messaging

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

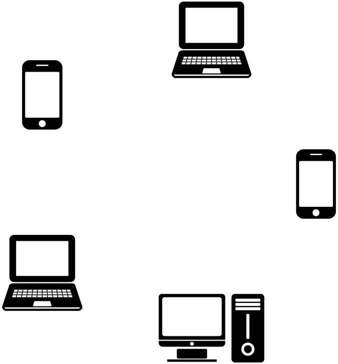
The New York Times

Signal Downloads Are Way Up Since the Protests Began

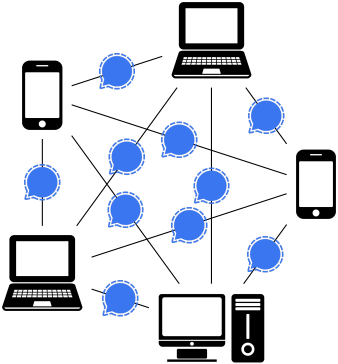
Organizers and demonstrators say they feel safer communicating with end-to-end encryption.



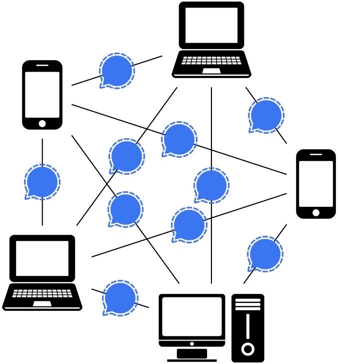
State of the art, before MLS



State of the art, before MLS

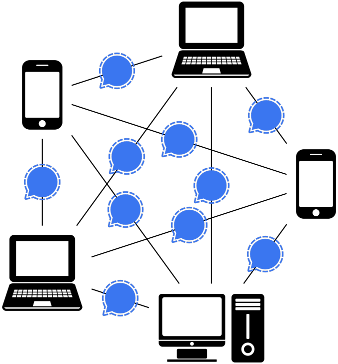


State of the art, before MLS



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/

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 Agreement	Update PPCS	Remove PACS
	Send	Recv	Send	Recv	New	Send	Recv	Send	Recv			
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
$\text{TreeKEM}_{\text{B}}^{+}$	N	1	1	1	1	$log(N)..N$	1	$log(N)..N$	1	Yes	Yes	No*
$\text{TreeKEM}_{\text{B+S}}^{+}$	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]

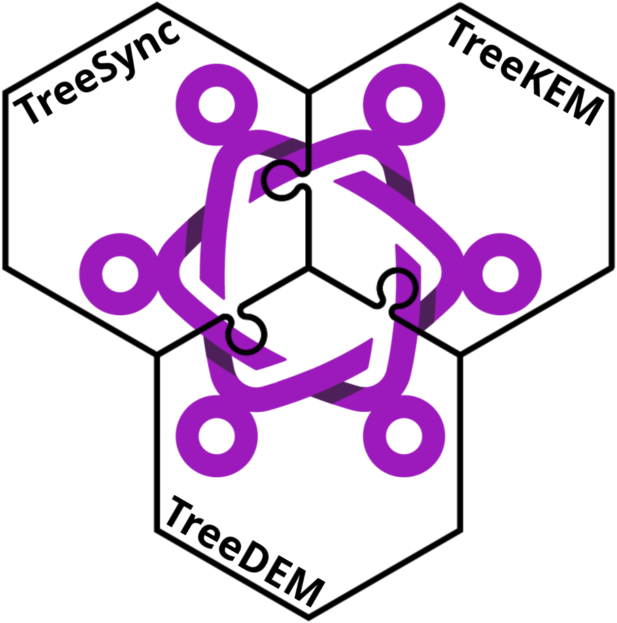
1,233 commits

0 Open ✓ 582 Closed

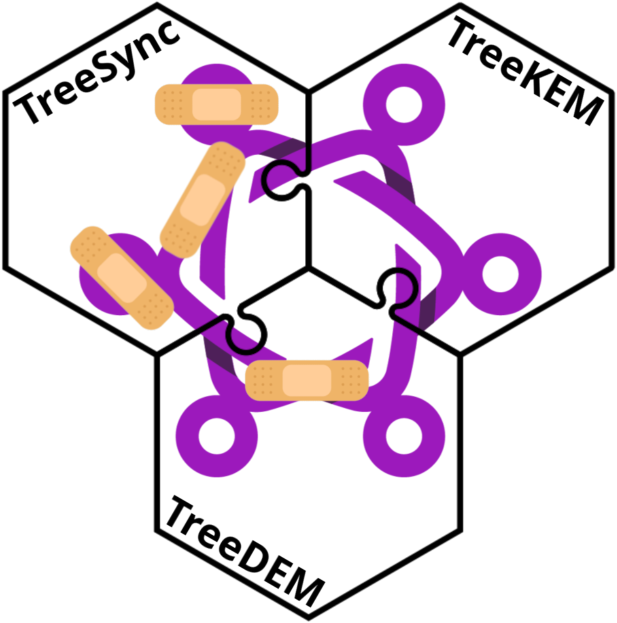
Our contributions

Contributions TL;DR

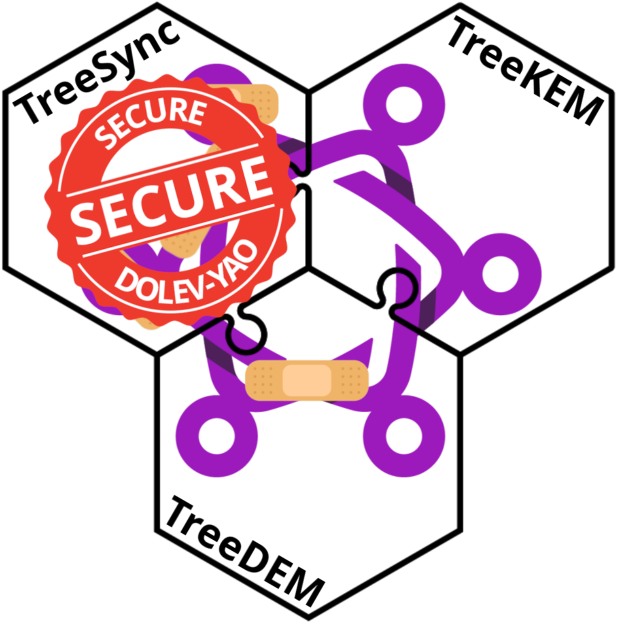
Contributions TL;DR



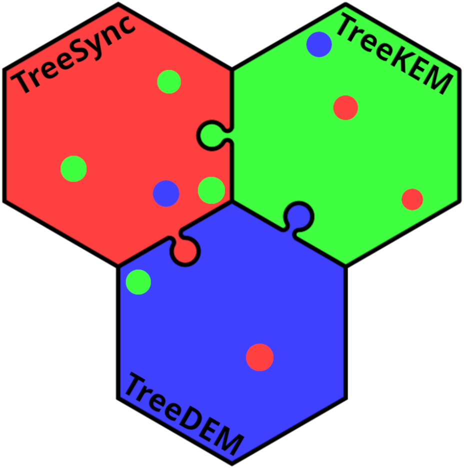
Contributions TL;DR



Contributions TL;DR

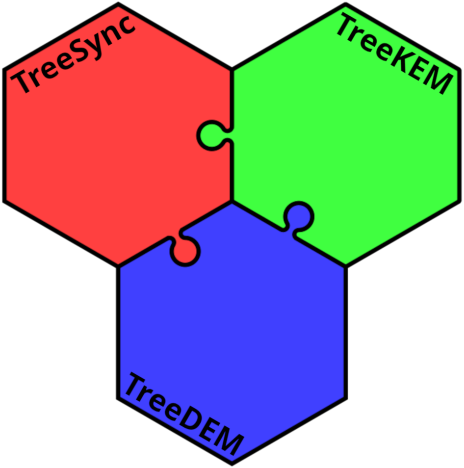


Contribution: Modularizing MLS



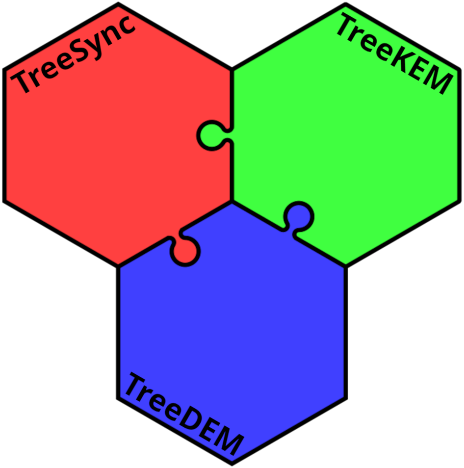
TreeSync: authenticated group synchronization
TreeKEM: efficient continuous group key establishment
TreeDEM: forward secure group messaging

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: Formal proof of TreeSync

MLS

$\downarrow$

F* specification

Contribution: Formal proof of TreeSync

MLS

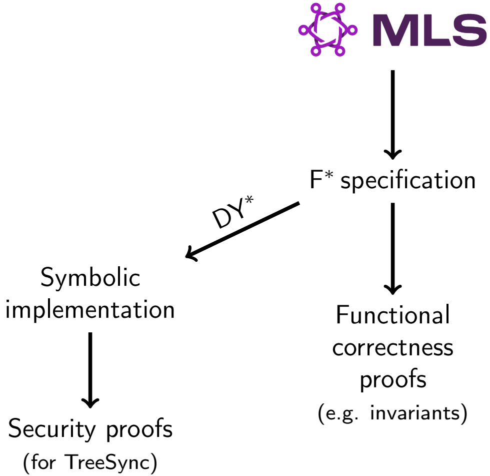
↓

F* specification

↓

Functional
correctness
proofs
(e.g. invariants)

Contribution: Formal proof of TreeSync



Contribution: Formal proof of TreeSync

MLS

F* specification

DY*

HACL*

Symbolic implementation

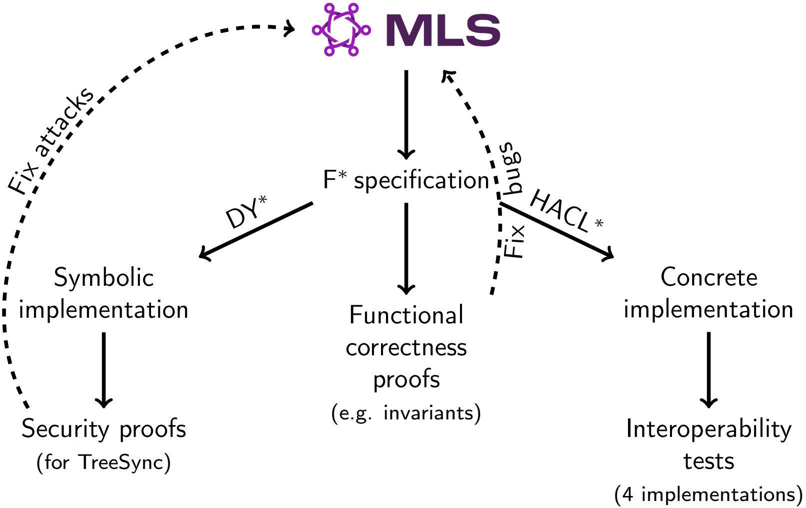
Security proofs
(for TreeSync)

Functional correctness proofs
(e.g. invariants)

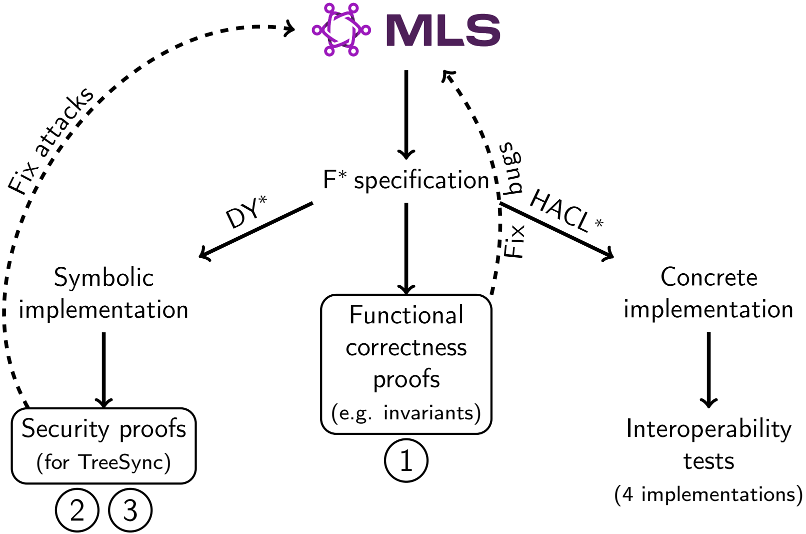
Concrete implementation

Interoperability tests
(4 implementations)

Contribution: Formal proof of TreeSync



Contribution: Formal proof of TreeSync



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

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

TreeDEM

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

Contribution: Fixing a signature ambiguity attack

Same key

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

Same key

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

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

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

Different types

Contribution: Fixing a signature ambiguity attack

Same key

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

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

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

Different types

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

Same key

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

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

Different types

What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T1}(\text{msg}_1) = \text{serialize}_{T2}(\text{msg}_2)$?
Bad interaction between TreeSync and TreeDEM!

Contribution: Fixing a signature ambiguity attack

Same key

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

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

Different types

What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T1}(\text{msg}_1) = \text{serialize}_{T2}(\text{msg}_2)$?
Bad interaction between TreeSync and TreeDEM!

Contribution: Fixing a signature ambiguity attack

Same key

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

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

Different types

What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T1}(\text{msg}_1) = \text{serialize}_{T2}(\text{msg}_2)$?
Bad interaction between TreeSync and TreeDEM!

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

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

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