

TreeSync: Authenticated Group Management for Messaging Layer Security



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*

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.



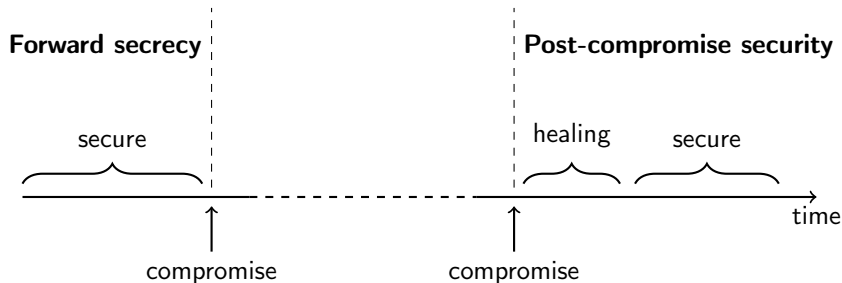
Secure group messaging

<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.



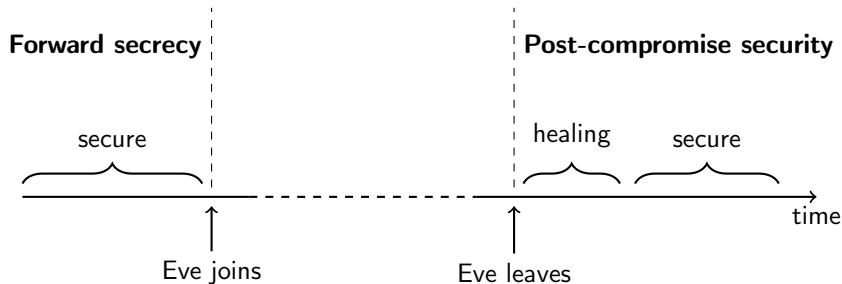
Secure group messaging

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

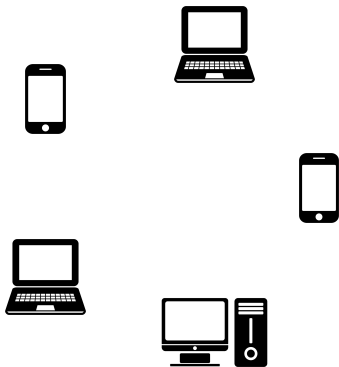
The New York Times

Signal Downloads Are Way Up Since the Protests Began

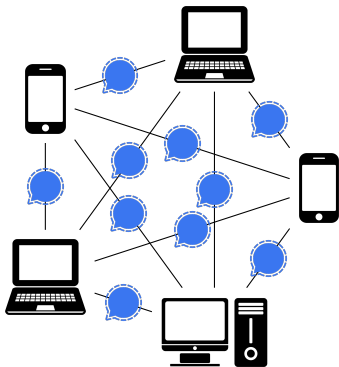
Organizers and demonstrators say they feel safer communicating with end-to-end encryption.



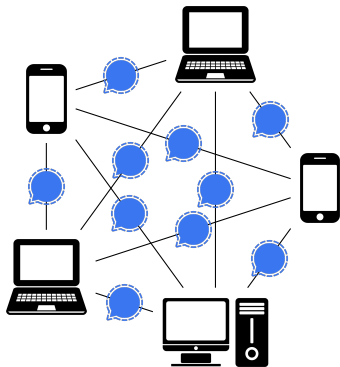
State of the art, before MLS



State of the art, before MLS



State of the art, before MLS

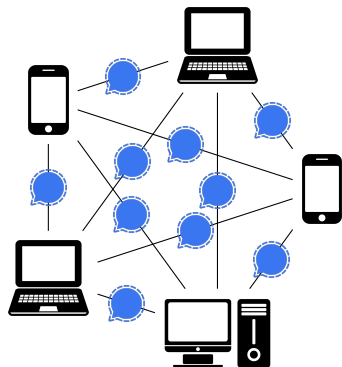


N devices

$O(N^2)$ Signal channels!

Slow for large N, e.g. $N \simeq 1000$

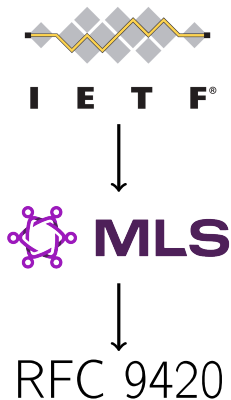
State of the art, before MLS



N devices

$O(N^2)$ Signal channels!

Slow for large N, e.g. $N \simeq 1000$



Design constraints:
Secure, efficient, 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 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
TreeKEM _B ⁺	N	1	1	1	1	$\log(N) \dots N$	1	$\log(N) \dots N$	1	Yes	Yes	No*
TreeKEM _{B+S} ⁺	N	1	1	1	N	$\log(N) \dots N$	1	$\log(N) \dots N$	1	Yes	Yes	Yes

Protocol

Performance

Security

A complex RFC

Table of Contents

1. Introduction	12. Group Evolution
2. Terminology	12.1. Proposals
2.1. Presentation Language	12.1.1. Add
2.1.1. Optional Value	12.1.2. Update
2.1.2. Variable-Size Vector Length Headers	12.1.3. Remove
3. Protocol Overview	12.1.4. PreSharedKey
3.1. Cryptographic State and Evolution	12.1.5. ReInit
3.2. Example Protocol Execution	12.1.6. ExternalInit
3.3. External Joins	12.1.7. GroupContextExtensions
3.4. Relationships between Epochs	12.1.8. External Proposals
4. Ratchet Tree Concepts	12.2. Proposal List Validation
4.1. Ratchet Tree Terminology	12.3. Applying a Proposal List
4.1.1. Ratchet Tree Nodes	12.4. Commit
4.1.2. Paths through a Ratchet Tree	12.4.1. Creating a Commit
4.2. Views of a Ratchet Tree	12.4.2. Processing a Commit
5. Cryptographic Objects	12.4.3. Adding Members to the Group
5.1. Cipher Suites	13. Extensibility
5.1.1. Public Keys	13.1. Additional Cipher Suites
5.1.2. Signing	13.2. Proposals
5.1.3. Public Key Encryption	13.3. Credential Extensibility
5.2. Hash-Based Identifiers	13.4. Extensions
5.3. Credentials	13.5. GREASE
5.3.1. Credential Validation	14. Sequencing of State Changes
5.3.2. Credential Expiry and Revocation	15. Application Messages
5.3.3. Uniquely Identifying Clients	15.1. Padding
6. Message Framing	15.2. Restrictions
6.1. Content Authentication	15.3. Delayed and Reordered Application Messages
6.2. Encoding and Decoding a Public Message	16. Security Considerations
6.3. Encoding and Decoding a Private Message	16.1. Transport Security
6.3.1. Content Encryption	16.2. Confidentiality of Group Secrets
6.3.2. Sender Data Encryption	16.3. Confidentiality of Sender Data
7. Ratchet Tree Operations	16.4. Confidentiality of Group Metadata
7.1. Parent Node Contents	16.4.1. GroupID, Epoch, and Message Frequency
7.2. Leaf Node Contents	16.4.2. Group Extensions
7.3. Leaf Node Validation	16.4.3. Group Membership
7.4. Ratchet Tree Evolution	16.5. Authentication
7.5. Synchronizing Views of the Tree	16.6. Forward Secrecy and Post-Compromise Security
7.6. Update Paths	16.7. Uniqueness of Ratchet Tree Key Pairs
7.7. Adding and Removing Leaves	16.8. KeyPackage Reuse
7.8. Tree Hashes	16.9. Delivery Service Compromise
7.9. Parent Hashes	16.10. Authentication Service Compromise
7.9.1. Using Parent Hashes	16.11. Additional Policy Enforcement
7.9.2. Verifying Parent Hashes	16.12. Group Fragmentation by Malicious Insiders
8. Key Schedule	17. IANA Considerations
8.1. Group Context	17.1. MLS Cipher Suites
8.2. Transcript Hashes	17.2. MLS Wire Formats
8.3. External Initialization	17.3. MLS Extension Types
8.4. Pre-Shared Keys	17.4. MLS Proposal Types
8.5. Exporters	17.5. MLS Credential Types
8.6. Resumption PSK	17.6. MLS Signature Labels
8.7. Epoch Authenticators	17.7. MLS Public Key Encryption Labels
9. Secret Tree	17.8. MLS Exporter Labels
9.1. Encryption Keys	17.9. MLS Designated Expert Pool
9.2. Deletion Schedule	17.10. The "message/mls" Media Type
10. Key Packages	18. References
10.1. KeyPackage Validation	18.1. Normative References
11. Group Creation	18.2. Informative References
11.1. Required Capabilities	Appendix A. Protocol Origins of Example Trees
11.2. Reinitialization	Appendix B. Evolution of Parent Hashes
11.3. Subgroup Branching	Appendix C. Array-Based Trees
	Appendix D. Link-Based Trees
	Contributors
	Authors' Addresses

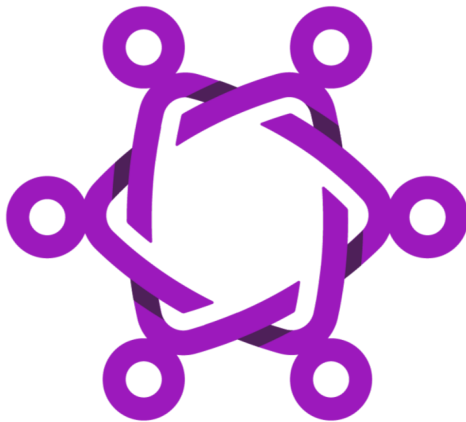
[Page 164]

 1,233 commits

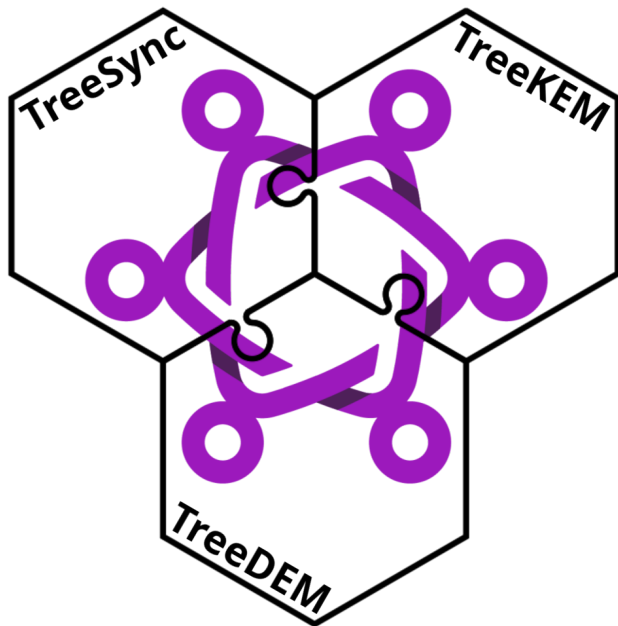
 0 Open  582 Closed

Our contributions

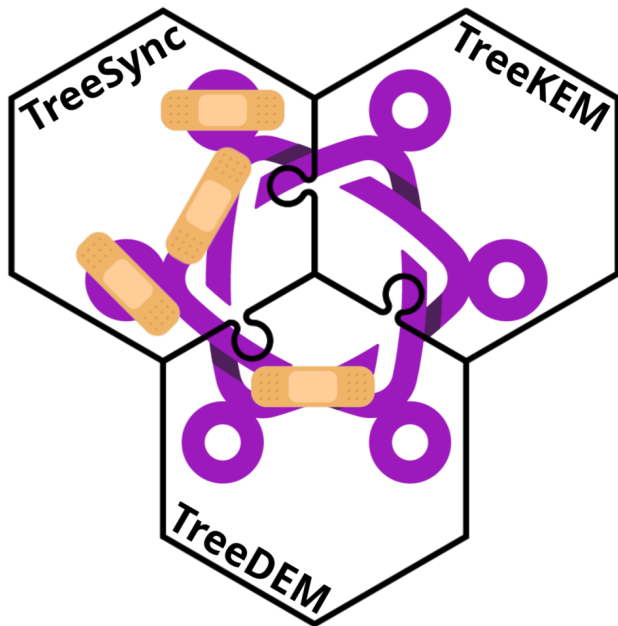
Contributions TL;DR



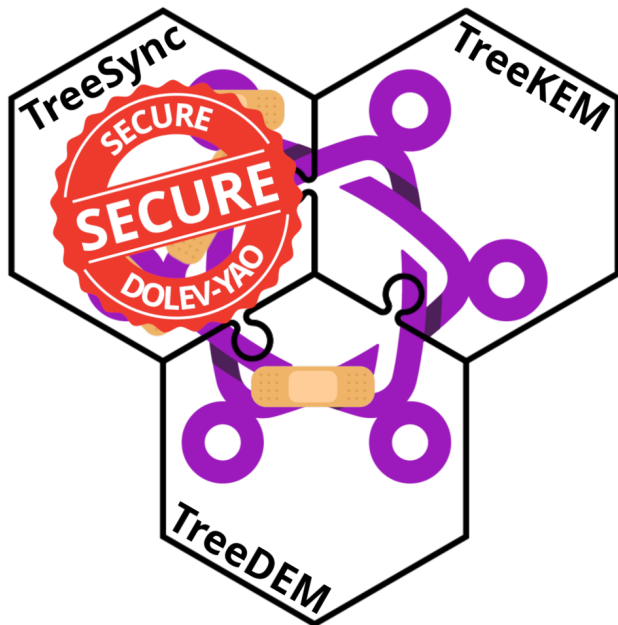
Contributions TL;DR



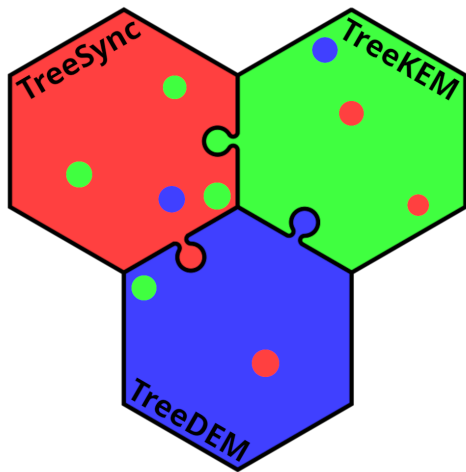
Contributions TL;DR



Contributions TL;DR



Contribution: Modularizing MLS

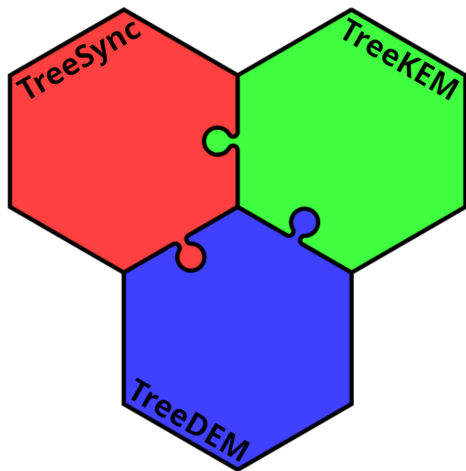


TreeSync: authenticated group synchronization

TreeKEM: efficient continuous group key establishment

TreeDEM: forward secure group messaging

Contribution: Modularizing MLS

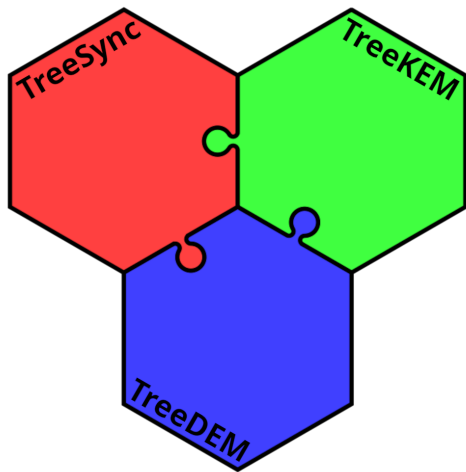


TreeSync: authenticated group synchronization

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



F* specification

Contribution: Formal proof of TreeSync

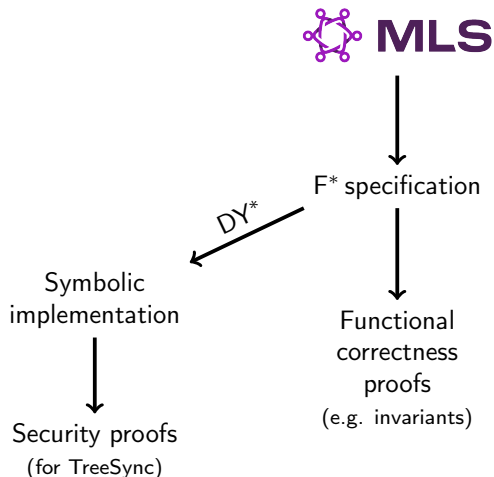


F* specification

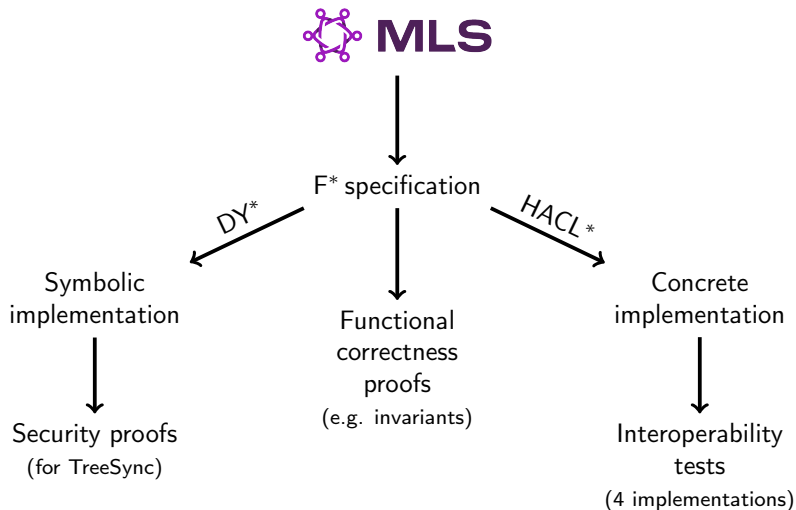


Functional
correctness
proofs
(e.g. invariants)

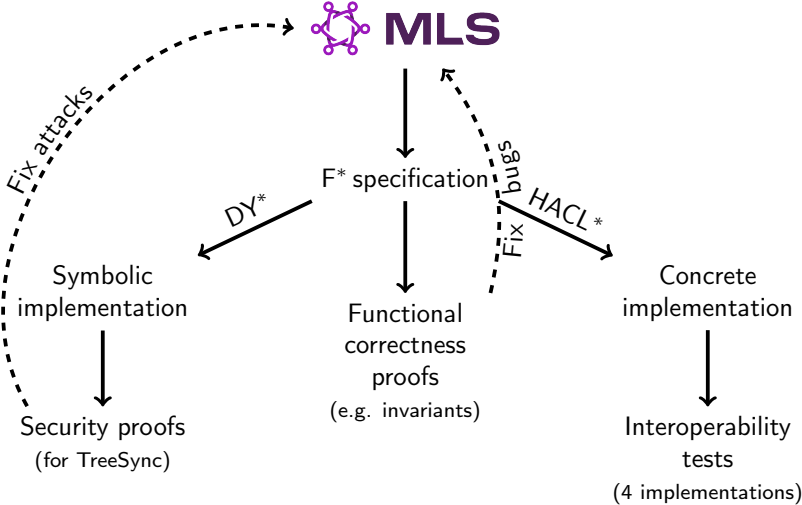
Contribution: Formal proof of TreeSync



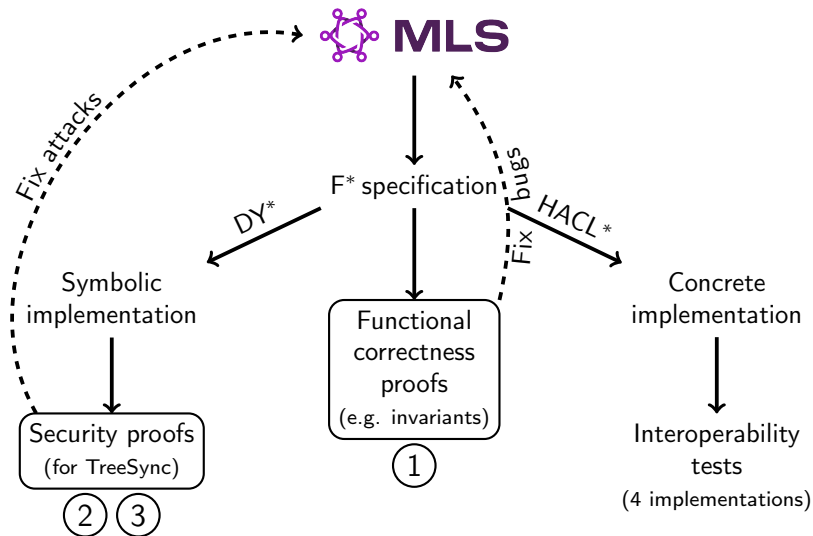
Contribution: Formal proof of TreeSync



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

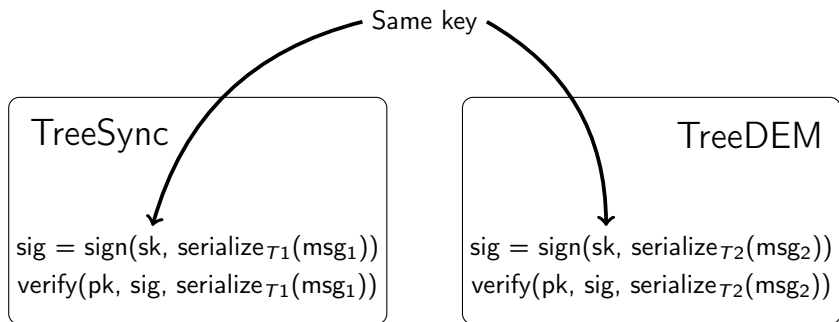
TreeSync

$\text{sig} = \text{sign}(\text{sk}, \text{serialize}_{T_1}(\text{msg}_1))$
 $\text{verify}(\text{pk}, \text{sig}, \text{serialize}_{T_1}(\text{msg}_1))$

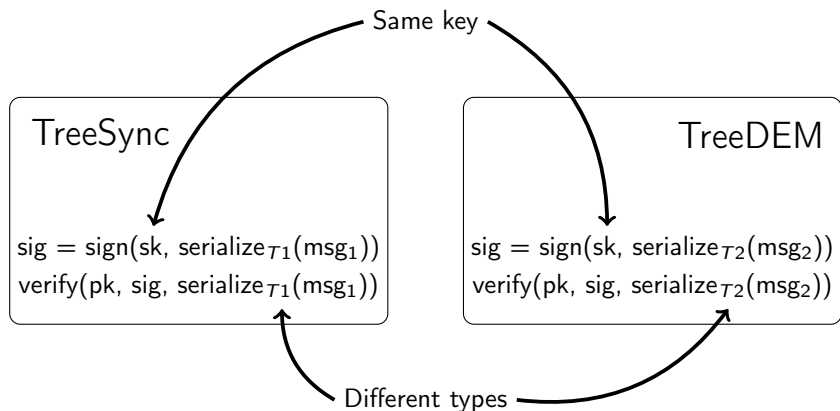
TreeDEM

$\text{sig} = \text{sign}(\text{sk}, \text{serialize}_{T_2}(\text{msg}_2))$
 $\text{verify}(\text{pk}, \text{sig}, \text{serialize}_{T_2}(\text{msg}_2))$

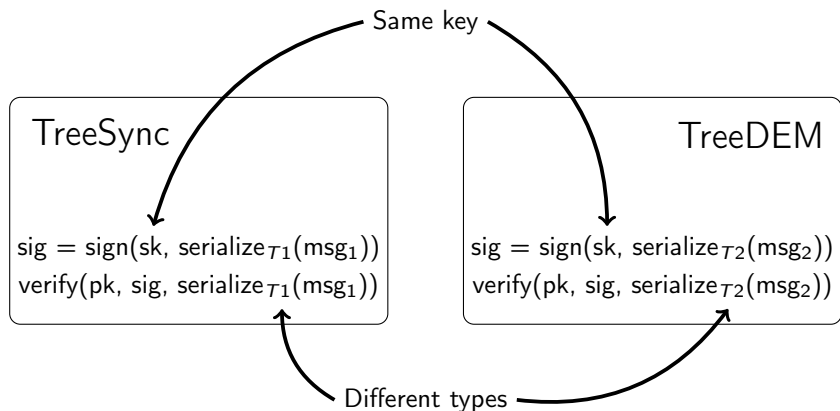
Contribution: Fixing a signature ambiguity attack



Contribution: Fixing a signature ambiguity attack

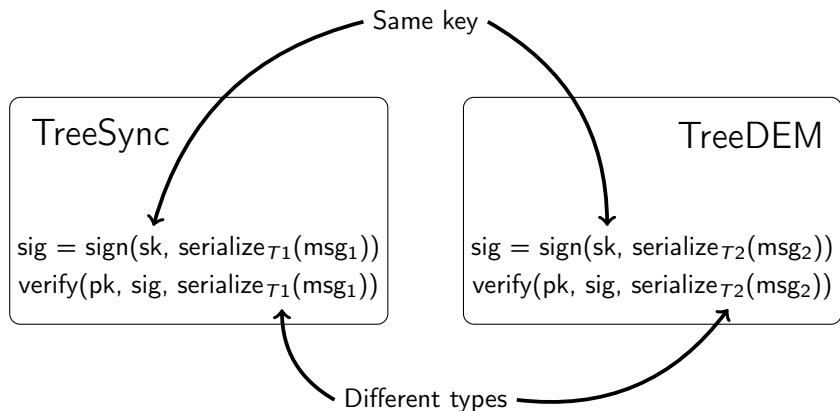


Contribution: Fixing a signature ambiguity attack



What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T_1}(\text{msg}_1) = \text{serialize}_{T_2}(\text{msg}_2)$?

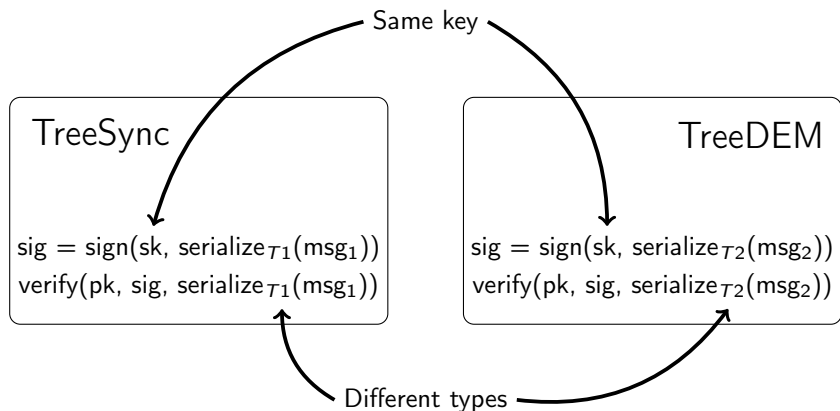
Contribution: Fixing a signature ambiguity attack



What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T_1}(\text{msg}_1) = \text{serialize}_{T_2}(\text{msg}_2)$?

Bad interaction between TreeSync and TreeDEM!

Contribution: Fixing a signature ambiguity attack

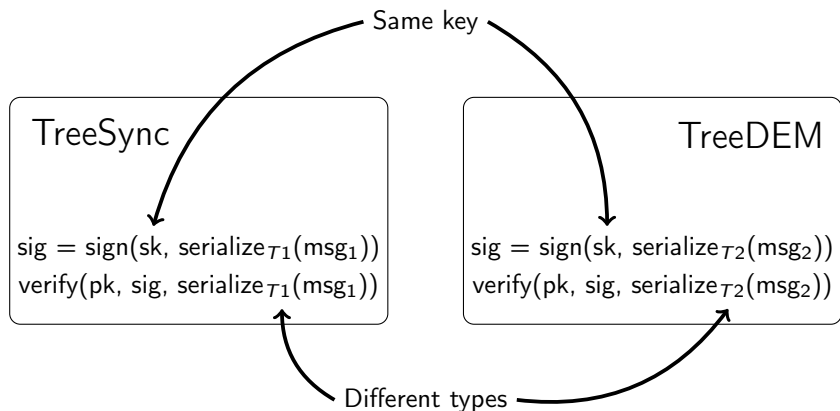


What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T_1}(\text{msg}_1) = \text{serialize}_{T_2}(\text{msg}_2)$?

Bad interaction between TreeSync and TreeDEM!



Contribution: Fixing a signature ambiguity attack



What if $\exists \text{msg}_1 \text{msg}_2, \text{serialize}_{T_1}(\text{msg}_1) = \text{serialize}_{T_2}(\text{msg}_2)$?

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

