Formally analyzing a cryptographic protocol standard

(or: how MLS kept this PhD student busy for three years)

Théophile Wallez, *Inria Paris*
+ work of co-authors

Introduction

The destination of my PhD:

- become a cryptographic protocol analyst
- produce a machine-checked security proof of MLS (secure group messaging protocol)

Introduction

The destination of my PhD:

- become a cryptographic protocol analyst
- produce a machine-checked security proof of MLS (secure group messaging protocol)

The journey:

- help to fix flaws in MLS before its standardization
- identify and fill gaps in formal security proofs (Comparse)
- improve tools to conduct symbolic security analysis at scale (DY^*)
- ...

Introduction

The destination of my PhD:

- become a cryptographic protocol analyst
- produce a machine-checked security proof of MLS (secure group messaging protocol)

The journey:

- help to fix flaws in MLS before its standardization
- identify and fill gaps in formal security proofs (Comparse)
- improve tools to conduct symbolic security analysis at scale (DY*)
- ...

Goal of this talk: share lessons I've learned

- for protocol analysts
- for protocol designers

Analyzing cryptographic protocols

Traditional pen & paper proofs:

- 👍 several proof techniques (game-hop, UC, SSP, . . .)
- ☹️ requires expert humans to check the proof

Analyzing cryptographic protocols

Traditional pen & paper proofs:
- 👍 several proof techniques (game-hop, UC, SSP, ...)
- ☹️ requires expert humans to check the proof

Machine-checked computational proofs:
- 👍 several tools (CryptoVerif, EasyCrypt, Squirrel, Owl, ProofFrog, ...)
- 👍 same guarantees as pen & paper proofs
- ☹️ limited automation

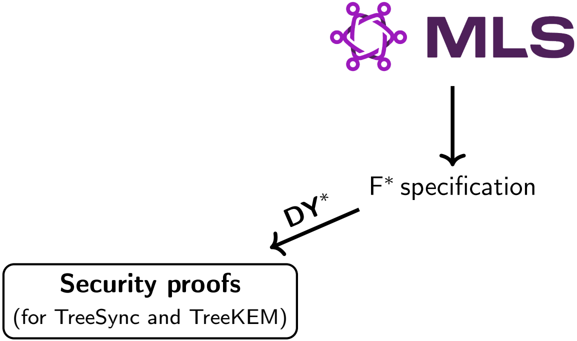
Analyzing cryptographic protocols

Traditional pen & paper proofs:
👍 several proof techniques (game-hop, UC, SSP, ...)
☹️ requires expert humans to check the proof

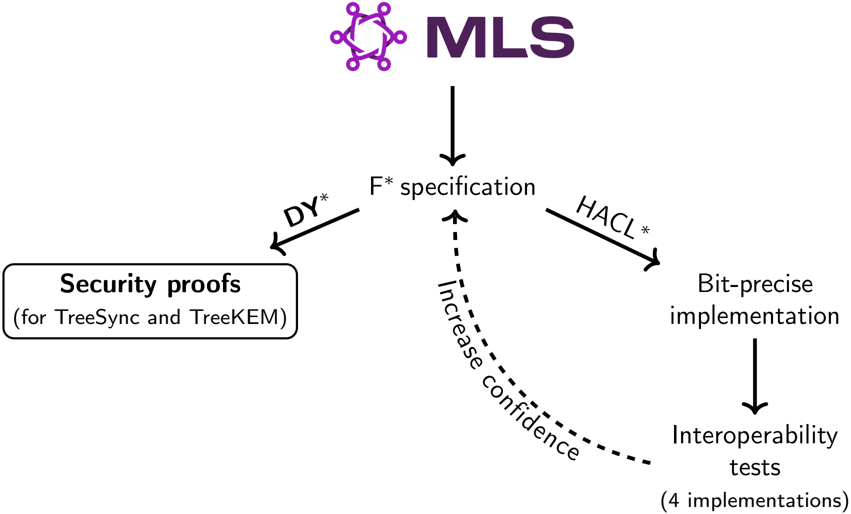
Machine-checked computational proofs:
👍 several tools (CryptoVerif, EasyCrypt, Squirrel, Owl, ProofFrog, ...)
👍 same guarantees as pen & paper proofs
☹️ limited automation

Machine-checked symbolic proofs:
👍 several tools (ProVerif, Tamarin, DY*, ...)
👍 good automation
😕 symbolic model is less precise than computational model
👍 many successes during the last decade (TLS 1.3, Signal, ...)

Our approach for protocol analysis



Our approach for protocol analysis

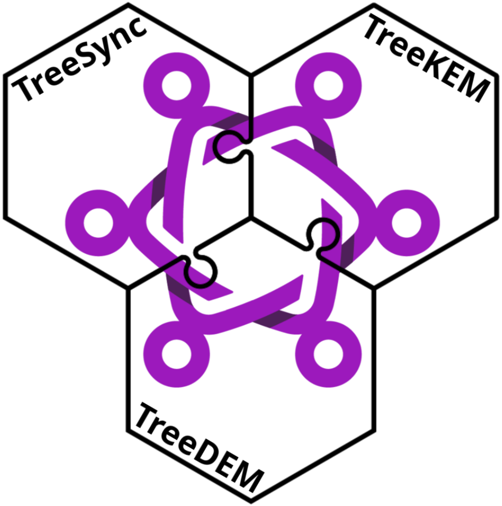


Symbolic security analysis of MLS

Towards a modular analysis of MLS

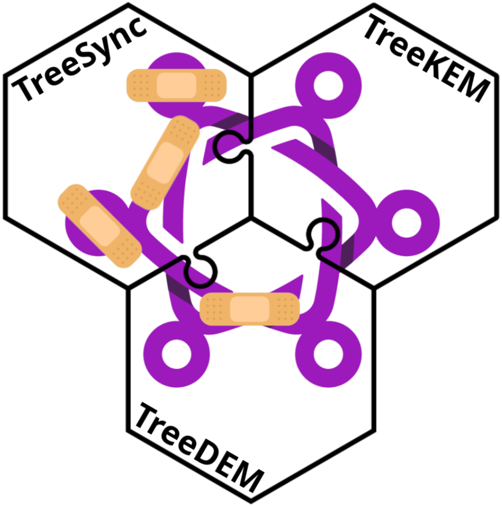
MLS specification (RFC 9420): 120 pages

Towards a modular analysis of MLS



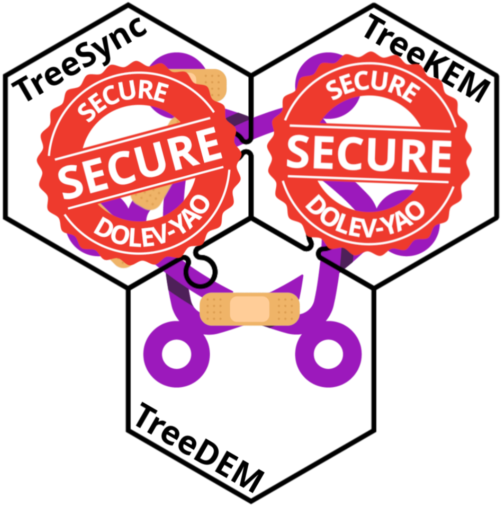
MLS specification (RFC 9420): 120 pages

Towards a modular analysis of MLS



MLS specification (RFC 9420): 120 pages

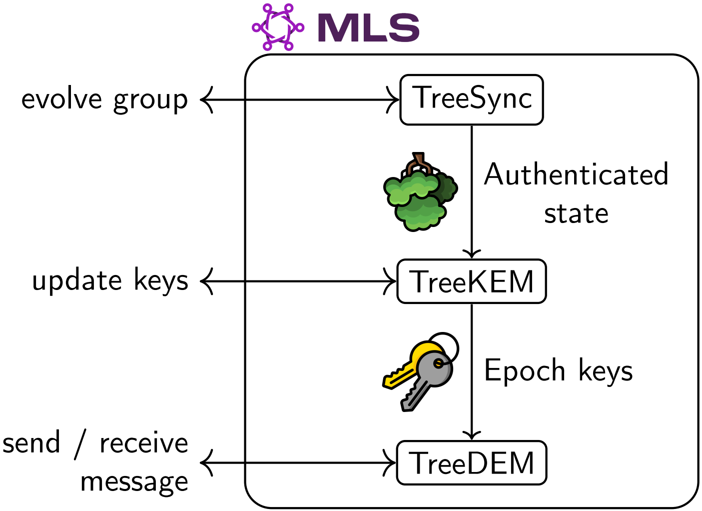
Towards a modular analysis of MLS



MLS specification (RFC 9420): 120 pages

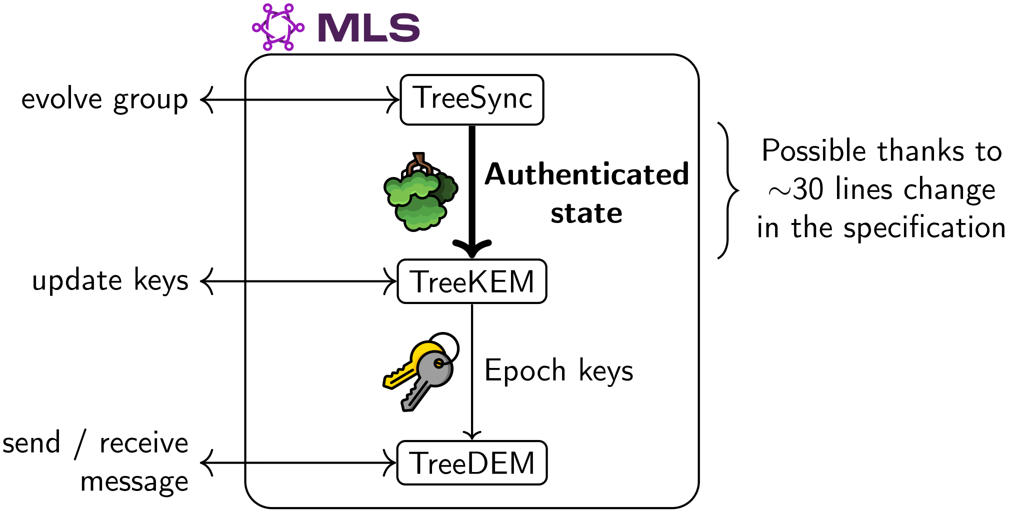
Modularizing MLS

("TreeSync: ...", USENIX Security '23, https://ia.cr/2022/1732)



Modularizing MLS

("TreeSync: . . .", USENIX Security '23, https://ia.cr/2022/1732)

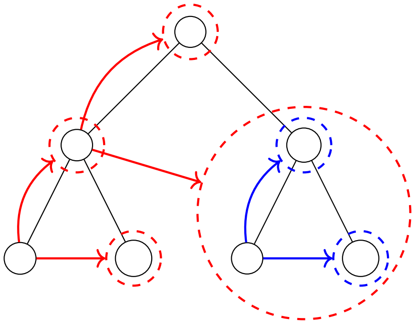


Lesson for protocol designers: modularize protocols

- Collaborate with protocol analysts
- Bonus: protocol is easier to understand
- Bonus: help implementers

Proving security of TreeSync

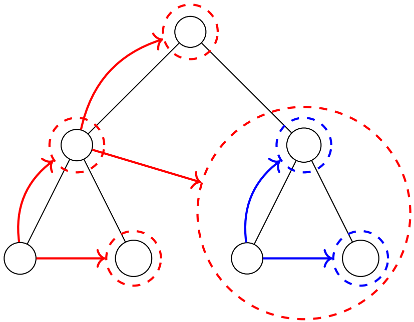
("TreeSync: . . .", USENIX Security '23, `https://ia.cr/2022/1732`)



- prove agreement theorem (incl. membership agreement)
- relies on minimal assumptions on TreeKEM and TreeDEM

Proving security of TreeSync

("TreeSync: ...", USENIX Security '23, https://ia.cr/2022/1732)



- prove agreement theorem (incl. membership agreement)
- relies on minimal assumptions on TreeKEM and TreeDEM

...however these assumption were initially not true

Signature ambiguity in MLS draft 12

("TreeSync: ...", USENIX Security '23, https://ia.cr/2022/1732)

TreeSync

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

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

Signature ambiguity in MLS draft 12

("TreeSync: ...", USENIX Security '23, https://ia.cr/2022/1732)

TreeSync

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

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

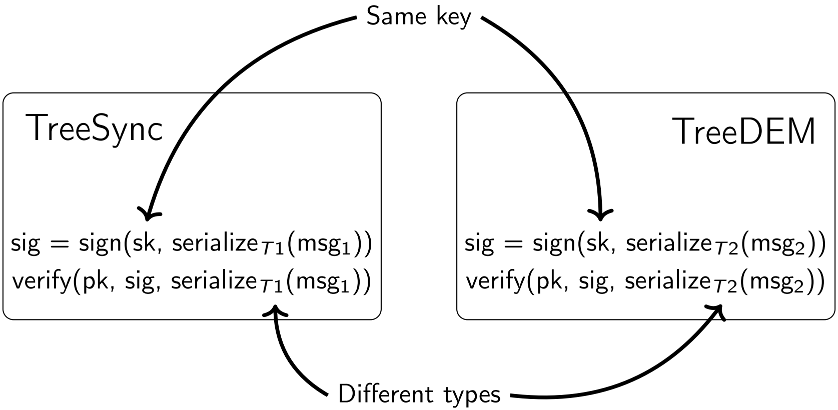
TreeDEM

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

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

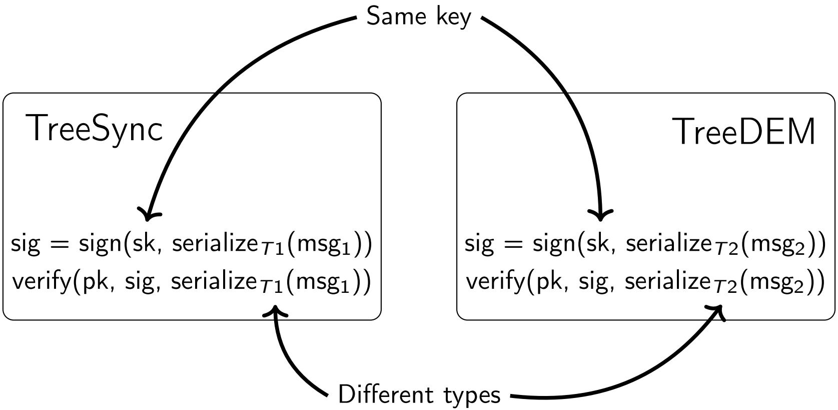
Signature ambiguity in MLS draft 12

("TreeSync: . . .", USENIX Security '23, `https://ia.cr/2022/1732`)



Signature ambiguity in MLS draft 12

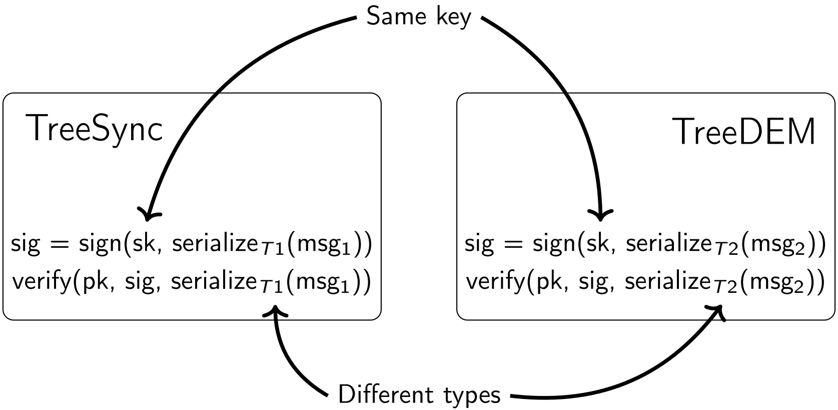
("TreeSync: . . .", USENIX Security '23, `https://ia.cr/2022/1732`)



What if $\text{serialize}_{T1}(\text{msg}_1) = \text{serialize}_{T2}(\text{msg}_2)$?
First step for an attack:
TreeDEM signature on msg_2 is a signature forgery on msg_1 in TreeSync!

Signature ambiguity in MLS draft 12

("TreeSync: ...", USENIX Security '23, `https://ia.cr/2022/1732`)



What if serialize$_{T1}$(msg$_1$) = serialize$_{T2}$(msg$_2$)?
First step for an attack:
TreeDEM signature on msg$_2$ is a signature forgery on msg$_1$ in TreeSync!

Two questions

From a protocol designer perspective:

- How did this attack survive 4 years and 12 drafts of the MLS standard, although this is a classic issue known as “lack of domain-separation”?

Our answer:

- there is no rigorous definition for “domain-separation”
- it is hard to enforce in a large standard

Two questions

From a protocol designer perspective:

- How did this attack survive 4 years and 12 drafts of the MLS standard, although this is a classic issue known as “lack of domain-separation”?

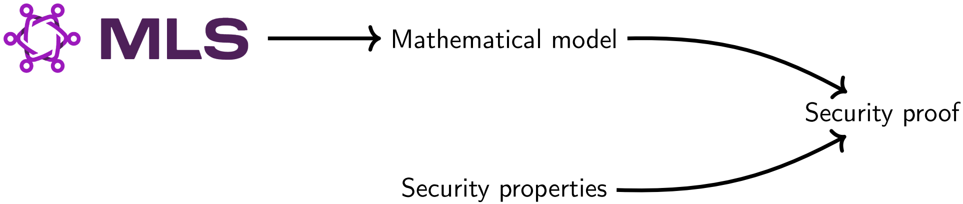
Our answer:

- there is no rigorous definition for “domain-separation”
- it is hard to enforce in a large standard

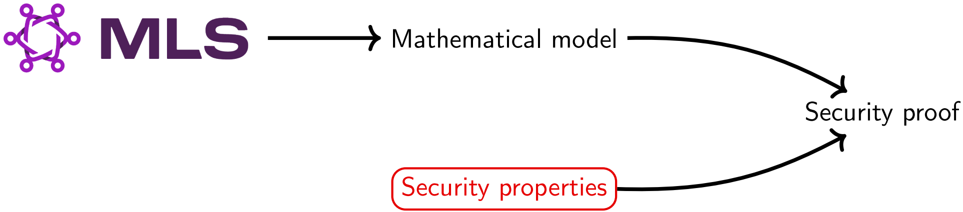
From a protocol analyst perspective:

- Why was this attack not caught by previous pen & paper security proofs?

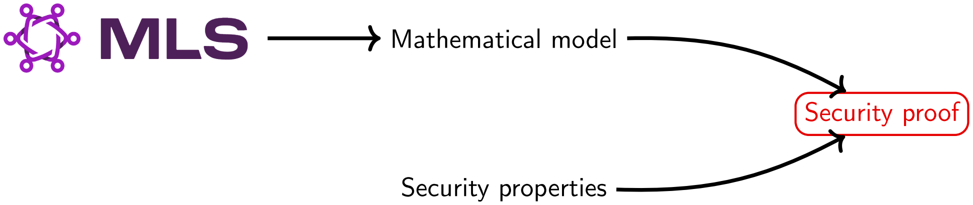
Why the attack was not caught by previous security proofs?



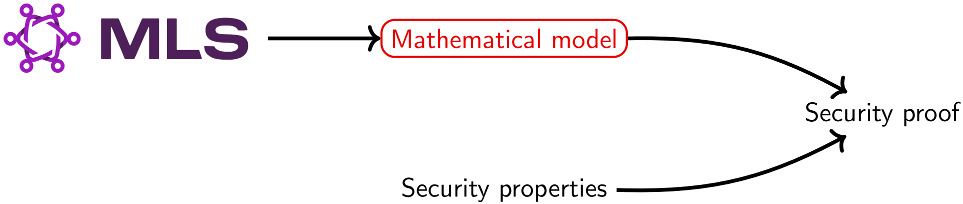
Why the attack was not caught by previous security proofs?



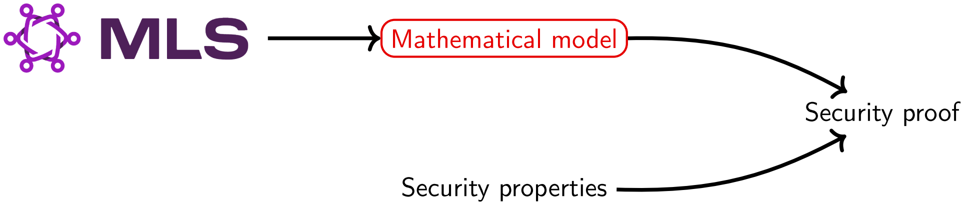
Why the attack was not caught by previous security proofs?



Why the attack was not caught by previous security proofs?



Why the attack was not caught by previous security proofs?



In mathematical models of MLS: no precise message format

$\mathsf{leafNodeTBS} \leftarrow (\mathtt{id}, \mathsf{pk}, \mathsf{spk}, \mathsf{parentHash}, \mathsf{ln\_source}, \mathsf{source})$
$\mathsf{sig} \leftarrow \mathsf{Sig.Sign}(\mathsf{ssk}, \mathsf{leafNodeTBS})$
...
$\mathtt{groupInfoTBS} \leftarrow (\mathsf{groupCtxt}, \gamma'.\tau.\mathsf{public}(), \mathtt{confTag}, \gamma'.\mathsf{leafIdx}())$
$\mathsf{sig} \leftarrow \mathsf{Sig.Sign}(\gamma'.\mathsf{ssk}, \mathtt{groupInfoTBS})$

"ETK: External-Operations TreeKEM and the Security of MLS in RFC 9420", C. Cremers, E. Günsay, V. Wesselkamp, M. Zhao

Lesson for protocol analysts: reason on precise mathematical models

- catch subtle attacks
- bonus: also provide a reference implementation

Lesson for protocol analysts: reason on precise mathematical models

- catch subtle attacks
- bonus: also provide a reference implementation

Problem: reasoning on message formats makes proof more complex

Lesson for protocol analysts: reason on precise mathematical models

- catch subtle attacks
- bonus: also provide a reference implementation

Problem: reasoning on message formats makes proof more complex

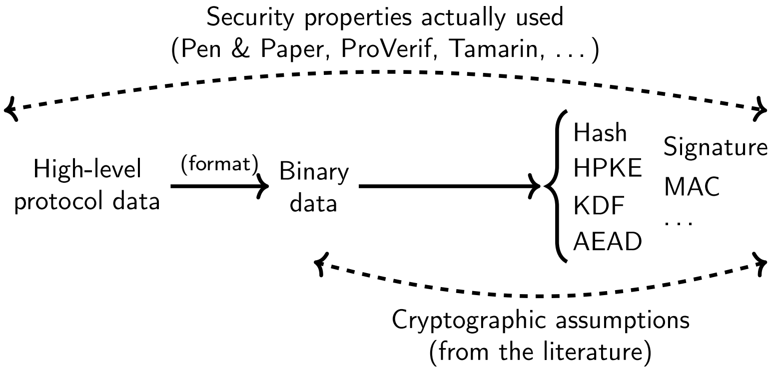
Our solution:

- define a rigorous notion of “secure formats”
- secure formats can soundly be abstracted away
- make a tool to check if a format is secure (Comparse)

Security critical message formats

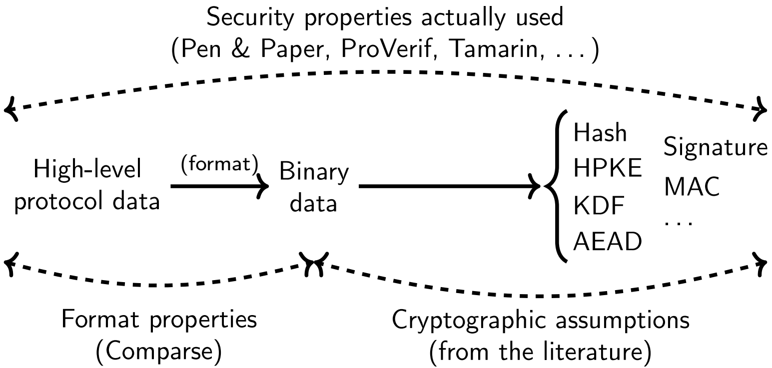
Security critical message formats

("Comparse: . . .", ACM CCS 2023, `https://ia.cr/2023/1390`)



Security critical message formats

("Comparse: …", ACM CCS 2023, https://ia.cr/2023/1390)



A rigorous approach toward domain separation

("Comparse: . . . ", ACM CCS 2023, `https://ia.cr/2023/1390`)

We build a systematic approach toward secure formats and "good domain-separation".

For example, we define "good domain-separation" for signatures as:

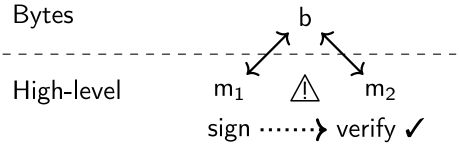
A rigorous approach toward domain separation

("Comparse: . . .", ACM CCS 2023, `https://ia.cr/2023/1390`)

We build a systematic approach toward secure formats and "good domain-separation".

For example, we define "good domain-separation" for signatures as:

- format must be injective (i.e. parseable)



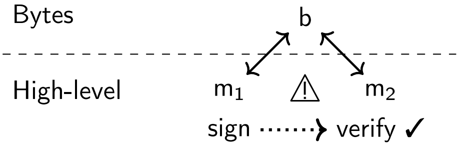
A rigorous approach toward domain separation

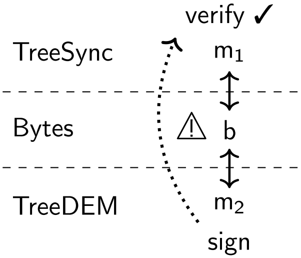
("Comparse: ...", ACM CCS 2023, https://ia.cr/2023/1390)

We build a systematic approach toward secure formats and "good domain-separation".

For example, we define "good domain-separation" for signatures as:

- format must be injective (i.e. parseable)
- choose one format per signature key (across all versions and extensions of the protocol)
- format must not depend on external context





A rigorous approach toward domain separation

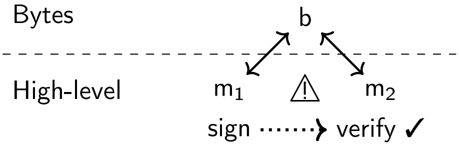
("Comparse: . . . ", ACM CCS 2023, https://ia.cr/2023/1390)

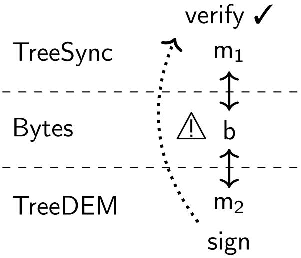
We build a systematic approach toward secure formats and "good domain-separation".

For example, we define "good domain-separation" for signatures as:

- format must be injective (i.e. parseable)
- choose one format per signature key (across all versions and extensions of the protocol)
- format must not depend on external context

This is a sufficient and necessary condition to abstract formats away in signatures!





Good domain-separation in real-world protocols

Claim: in real-world protocols, data sent on the network have “good domain-separation”.

Good domain-separation in real-world protocols

Claim: in real-world protocols, data sent on the network have “good domain-separation”.

```
enum {
    client_hello(1),
    server_hello(2),
    ...
    (255)
} HandshakeType;

struct {
    HandshakeType msg_type;    /* handshake type */
    uint24 length;             /* remaining bytes in message */
    select (Handshake.msg_type) {
        case client_hello:          ClientHello;
        case server_hello:          ServerHello;
        ...
    };
} Handshake;
```

TLS 1.3 Handshake message, properly domain-separated across versions since 1996 (SSLv3)

Ugly message formats in real-world protocols

In the same specification, TLS 1.3 Transcript hash

```
Transcript-Hash(M1, M2, ... Mn) = Hash(M1 || M2 || ... || Mn)
```

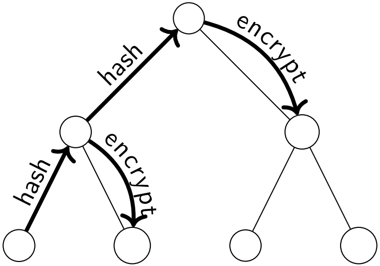
Lesson for protocol designers:
love all message formats equally

- rule out a whole class of attacks
- help protocol analysts willing to model them precisely

Symbolic security of MLS: TreeKEM

Proving security of TreeKEM

(“TreeKEM: ...”, to appear at IEEE S&P 2025, `https://ia.cr/2025/410`)



We prove a confidentiality theorem on TreeKEM.
Challenges:

- requires recursive data types
- inductive proofs
- an unbounded sequence of key derivations
- an unbounded sequence of public-key encryptions (and internally, KEMs)

DY* is a tool of choice for these challenges, still we had to heavily improve it.

Lesson for protocol analysts:
novel protocols may require new tools

// TODO: insert "modern problems require modern solutions" meme

- can't have "one tool to rule them all"
- similar to various pen & paper proof frameworks (game-hop, UC, SSP, ...)

Conclusion

- we produced machine-checked security proofs for parts of MLS (TreeSync & TreeKEM)
- developed a methodology to reason on a precise model of cryptographic standards
- shed light on the importance of message formatting in cryptographic protocols
- and propose a rigorous approach to domain-separation
- we improved the tools to perform machine-checked symbolic security proofs

</> https://github.com/Inria-Prosecco/mls-star

https://ia.cr/2022/1732 (TreeSync)
https://ia.cr/2023/1390 (Comparse)
https://ia.cr/2025/410 (TreeKEM)

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

References

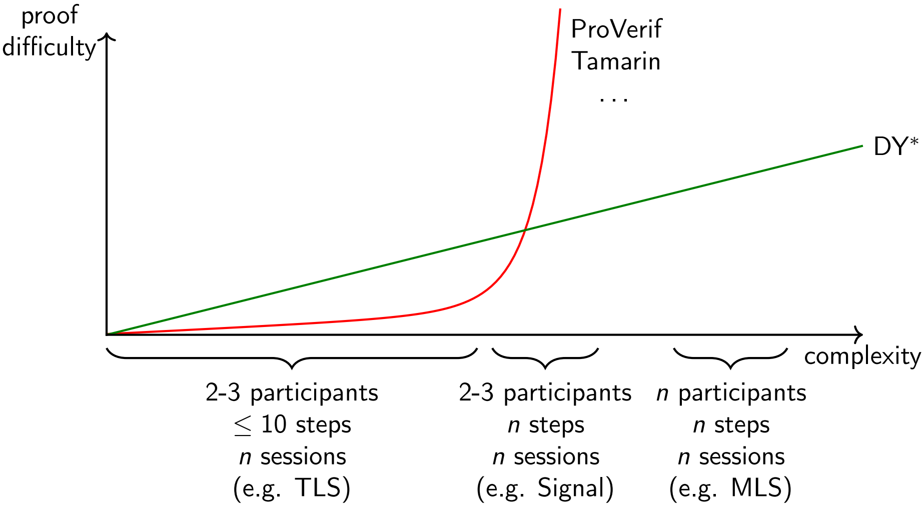
Théophile Wallez, Jonathan Protzenko, Benjamin Beurdouche, and Karthikeyan Bhargavan.
TreeSync: Authenticated group management for messaging layer security.
In *32nd USENIX Security Symposium (USENIX Security 23)*, August 2023.

Théophile Wallez, Jonathan Protzenko, and Karthikeyan Bhargavan.
Comparse: Provably secure formats for cryptographic protocols.
In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, November 2023.

Théophile Wallez, Jonathan Protzenko, and Karthikeyan Bhargavan.
TreeKEM: A modular machine-checked symbolic security analysis of group key agreement in messaging layer security, 2025.
To appear at IEEE S&P 2025. https://eprint.iacr.org/2025/410.

Backup slides

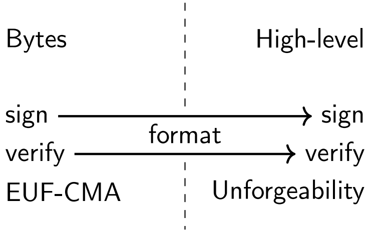
Symbolic protocol provers



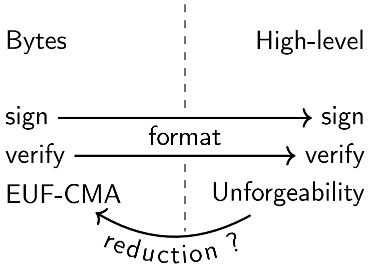
A rigorous approach to domain separation

Bytes	High-level
sign	
verify	
EUF-CMA	

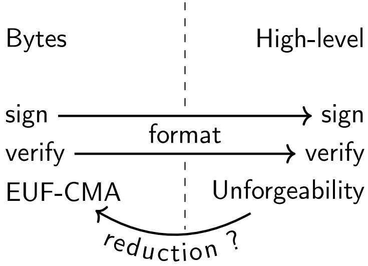
A rigorous approach to domain separation



A rigorous approach to domain separation

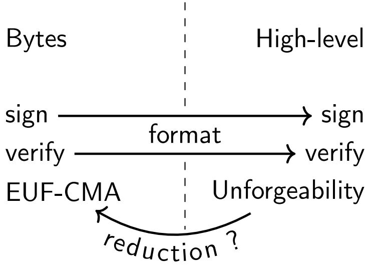


A rigorous approach to domain separation



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.