Formal analysis of LAKE-EDHOC

Charlie Jacomme, Elise Klein, Steve Kremer, Maïwenn Racouchot

March 21th, 2022

Protocol model

Formal verification

The SAPIC+ platform

Protocol description in applied-pi calculus, a high level programming language with abstracted network inputs and outputs. Export to different tools that automatically prove the security or find attacks:

- ProVerif: allows fast proofs
- Tamarin: more precise proofs

(tools used in multiple protocol analysis, like TLS 1.3, 5G-AKA, or EMV)

Primitive & Properties modeling

Modeling of primitives

Computations abstracted by their underlying properties.

E.g., a symmetric encryption is two abstract functions $\mathtt{enc}(), \mathtt{dec}()$, such that:

$$\forall\ m, sk.\ \mathtt{dec}(\mathtt{enc}(m, sk), sk) = m$$

$\hookrightarrow$ This can lead to modelings that abstract too many possible behaviours.

Properties modeling

We use a first-order temporal logic to specify security properties.

$$\begin{aligned}\forall\ pkI\ \ pkR\ \ k\ \sharp t_1\ \sharp t_2.\ AcceptR(pkI, pkR, k)@t_1\ \&\ Honest(pkI)@t_2 \\ \Rightarrow \exists \sharp t_3.\ t_3 < t_1\ \&\ AcceptI(pkI, pkR, k)@t_3\end{aligned}$$

Advanced primitive models

Weak Diffie-Hellman model

- There exists an identity element e such that: $e^x = e$.
- There exists a low-order point h, such that $h^x = h^y$ and $(h \times g^x)^z = g^{xz}$.

Weak signatures model

- Malleability. (ES256)
- Dishonest key where verification always succeed. (edDSA)

Weak hash model

- Length-extensions: $h(x|y) = h(h(x)|y)$ (SHA-1, SHA-256)
- Chosen prefix collisions: given p_1, p_2, the attacker may compute c_1, c_2 such that $h(p_1|c_1) = h(p_2|c_2)$. (MD5, SHA-1, SHA-256?)

The protocol model

LAKE-EDHOC

- The 4 methods executable in parallel;
- includes a Trust-On-First-Use paradigm;
- model all possible compromissions;
- alternate model with the KEM based variant.

Limitations

- No fine grained modeling of the cipher suite negotiation;
- no modeling of the key update mechanism;
- no modeling of the fourth message.

Automated analysis

Summary of results from automated analysis

Property	Threat model					
	Basic	Weak Sig	Weak DH	Ephemeral + Session leaks	Weak Hashes + DH	KEM
Confidentiality	✓	✓	✓	✓	✗	✓
Agent Auth.	✓	✓	✓	✗	✓	✓
Transcript Auth.	✓	∼	✓	✓	✗	✓
Algo Auth.	✓	✓	✓	✓	✗	✓
Session key uniqueness	✓	✓	✗	✓	✗	✗
Non-repudiation soundness	✓	✓	∼	✓	∼	✓
Inj. non-repudiation	✓	∼	∼	✓	∼	∼

✓ : property satisfied
✗ : violation of property
∼ : unclear security

Weak Sig : weak signatures (malleable, yes keys)
Weak DH : small sub-groups
Weah Hash : Length extensions, chosen-prefix collisions

Table 1: Summary of results

High-level feedback

Security proofs

In most (strong) threat models, the protocol provides all expected security properties.

Suggestions for improvements

Simple changes and clarifications, identified through the automated analysis:

1. avoid potential misuse of the existing design;
2. strengthen the TEE implementation;
3. improve the future resilience of the protocol.

↪ We will soon send out mails on the mailing list with concrete proposals for each of those points.

Potential misuse

First potential misuse

Improving the guarantees on session key

The session key PRK_{4x3m} offers weaker properties than the exported keys:

- A dishonest responder may completely control the final value of PRK_{4x3m} (no contributiveness), either through the identity DH element, or a KEM misuse.
- The session key is not linked to the execution, and does not authenticate TH_4.

First potential misuse

Improving the guarantees on session key

The session key PRK_{4x3m} offers weaker properties than the exported keys:

- A dishonest responder may completely control the final value of PRK_{4x3m} (no contributiveness), either through the identity DH element, or a KEM misuse.
- The session key is not linked to the execution, and does not authenticate TH_4.

A concrete example

Authentication of TH_4 is broken when:

- a different key exporter that does not include TH_4 inside the key is used;
- AES_CCM is used, making CYPHERTEXT_4 malleable and thus giving a different value for TH_4 on both sides, despite an explicit key confirmation.

Improve security over the session key

Suggestion 1 - Additional "Master Secret" derivation

Instead of defining key material as the pair (PRK_4x3m, TH_4), introduce a final key derivation which will be the key material and final session key:
PRK_out := KDF(PRK_4x3m, TH_4).

Benefits

- An agent always inserts some of its own randomness inside PRK_out through TH_4: ensures contributiveness and avoids key control.
- Explicit key confirmation over PRK_out does now authenticate TH_4, reducing potential weak key exports.

Second potential misuse

Resending messages

"An EDHOC implementation MAY keep the protocol state to be able to recreate the previously sent EDHOC message and resend it" [page 73]

AEAD IV and key reuse

Recomputing message_3, when the signature is randomized, lead to reusing the same IV and key for distinct messages, which is outside of the recommended use for AEADs.

Suggestion 2 - forbid message recomputation

Forbid this behavior, and only allow to store the explicit value of the last message sent.

Strengthening the TEE implementation

Agent Authentication

Threat model

Authentication operations inside a TEE, but device otherwise compromised.

- leak the initiator ephemeral key at the beginning, and the session key at the end;
- but no access to authentication keys.

An impersonation attack

In Method 1 where I authenticates with static share

1. Att initiates a session with R, impersonating I, and receives g^r;
2. Att initates a session with I, with its own long term key, and forwards g^r;
3. Thanks to leaks, Att can complete session with I, and learn the session key;
4. the session key is the MAC key, and can be used to complete the session with R.

Issue and proposed fix

Main concern

- In method 1,2,3, the session key is actually the MAC key, and is sufficient for impersonation.
- It is not enough for all authentication operations to be safe to ensure authentication, and storing G_I inside a TEE does not increase the security level.

Suggestion 3 - stronger dependence of MAC_2 with G_IY

Make methods 1,2,3 provide the same level of guarantees as method 0 by ensuring that e.g. G_IY is required to compute the MAC, and not just the session key:

MAC_2 :=
EDHOC-KDF(PRK_3e2m, TH_2, "MAC_2", <ID_CRED_R, CRED_R, ?G_IY, ?EAD_2>, length)

Future proofing the protocol

Transcript collisions

Threat model

- The attacker can compute chosen prefix collisions.

  Given p_1, p_2, it can compute c_1, c_2 such that $h(p_1|c_1) = h(p_2|c_2)$
- Agents accept as DH share the identity element (or low-order points).

  The identity element e is such that $e^x = e$.

Consequences

Breaks secrecy, and may allow for downgrade attacks.

```
Trans_E := method |  suitesI  | G_X |  C_I  |  EAD_1  | G_Y | C_R

Trans_I :=  zero  | "suitesI" | g^x | "C_I" | "EAD_1" |  e  | c2 | g^y | C_R
Trans_R :=  zero  | "suitesI" |  e  | "C_I" |   c1    | g^y | "C_R"
```

Mitigations

Suggestion 4

While we don't know if chosen prefix collisions will ever be possible for SHA-256, we can already mitigate the consequences:

- checking for low-order group elements improves the guarantees;
- adding length restrictions over EADs and C_I, C_R;
- ensuring that the message processing fails in case of a typing error, and e.g. reject

  suites = [2* int / btstr] / int

Conclusion

Long-term plans

- Improve and deepen the analysis (key update, fourth message, ...);
- keep the models up to date with the drafts and up to the final RFC;
- maybe look at a computational proof of security in Squirrel (a proof assistant).

Questions?