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)
Primative & Properties modeling

Modeling of primitives
Computations abstracted by their underlying properties.
E.g., a symmetric encryption is two abstract functions $\text{enc}()$, $\text{dec}()$, such that:

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

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

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

$$\forall pkI \text{, pkR} \text{, } k \not\equiv t_1 \not\equiv t_2. \text{AcceptR}(pkI, pkR, k)@t_1 \land \text{Honest}(pkI)@t_2$$

$$\implies \exists \not\equiv t_3. \ t_3 < t_1 \land \text{AcceptI}(pkI, pkR, k)@t_3$$
### 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
<table>
<thead>
<tr>
<th>Property</th>
<th>Basic</th>
<th>Weak Sig</th>
<th>Weak DH</th>
<th>Ephemeral + Session leaks</th>
<th>Weak Hashes + DH</th>
<th>KEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Agent Auth.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transcript Auth.</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Algo Auth.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Session key uniqueness</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Non-repudiation soundness</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
</tr>
<tr>
<td>Inj. non-repudiation</td>
<td>✓</td>
<td>~</td>
<td>~</td>
<td>✓</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

✓: property satisfied
X: violation of property
~: unclear security

Table 1: Summary of results

Weak Sig: weak signatures (malleable, yes keys)
Weak DH: small sub-groups
Weak Hash: Length extensions, chosen-prefix collisions
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 $\text{TH}_4$. 
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:

\[ PRK_{\text{out}} := \text{KDF}(PRK_{4x3m}, TH_{4}). \]

**Benefits**

- An agent always inserts some of its own randomness inside \(PRK_{\text{out}}\) through \(TH_{4}\): ensures contributiveness and avoids key control.
- Explicit key confirmation over \(PRK_{\text{out}}\) does now authenticate \(TH_{4}\), reducing potential weak key exports.
**Second potential misuse**

<table>
<thead>
<tr>
<th>Resending messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>“An EDHOC implementation MAY keep the protocol state to be able to recreate the previously sent EDHOC message and resend it” [page 73]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AEAD IV and key reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recomputing message_3, when the signature is randomized, lead to <strong>reusing the same IV and key</strong> for distinct messages, which is outside of the recommended use for AEADs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Suggestion 2 - forbid message recomputation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forbid this behavior, and only allow to store the explicit value of the last message sent.</td>
</tr>
</tbody>
</table>
Strengthening the TEE implementation
**Agent Authentication**

<table>
<thead>
<tr>
<th>Threat model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication operations inside a TEE, but device otherwise compromised.</td>
</tr>
<tr>
<td>• leak the initiator ephemeral key at the beginning, and the session key at the end;</td>
</tr>
<tr>
<td>• but no access to authentication keys.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>An impersonation attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Method 1 where I authenticates with static share</td>
</tr>
<tr>
<td>1. Att initiates a session with R, impersonating I, and receives $g^r$;</td>
</tr>
<tr>
<td>2. Att initiates a session with I, with its own long term key, and forwards $g^r$;</td>
</tr>
<tr>
<td>3. Thanks to leaks, Att can complete session with I, and learn the session key;</td>
</tr>
<tr>
<td>4. the session key is the MAC key, and can be used to complete the session with R.</td>
</tr>
</tbody>
</table>
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:

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.

\[
\begin{align*}
\text{Trans}_E := & \quad \text{method} \mid \text{suitesI} \mid G_X \mid C_I \mid EAD_1 \mid G_Y \mid C_R \\
\text{Trans}_I := & \quad \text{zero} \mid "\text{suitesI}" \mid g^x \mid "C_I" \mid "EAD_1" \mid e \mid c_2 \mid g^y \mid C_R \\
\text{Trans}_R := & \quad \text{zero} \mid "\text{suitesI}" \mid e \mid "C_I" \mid c_1 \mid g^y \mid "C_R"
\end{align*}
\]
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?