A Survey on Protocol Verification and Security APIs

Weekly Seminar MAIS Darmstadt
(joint work with Steve Kremer and Graham Steel)

Robert Künneemann
Outline

• Security APIs and the analysis of stateful protocols
• Horn clause resolution
• stateful analysis using multiset rewriting
• case study (MSR): the Yubikey protocol
• a translation from a high-level description to MSR and why it is needed
Introduction

(Security APIs and the analysis of stateful protocols)
Where do we want to go?

Participant A  Protocol  Participant B
Where do we want to go?

Participant A  Protocol  Participant B
Where do we want to go?

Participant A  Protocol  Participant B
Where do we want to go?

 Participant A

 Protocol

 Participant B
Where do we want to go?

Participant A

Protocol

Participant B

should never leak keys
smart cards
A really safe computer
Commands

1. generate keys: \( h \rightarrow (k, \text{attr}) \)

2. change attributes

3. commands with pre-conditions

\[ h_1 \rightarrow (k_1, \text{wrap}), h_2 \rightarrow (k_2, a) : \{|k_2|\}_{k_1} \]
Running Example: Wrap/Dec

1. Generate Key

2. Change Attributes

3. Wrap: \( h_1 \to k_1, h_2 \to k_2 : \{|k_2|\}_{k_1} \)

4. Dec: \( h_1 \to k_1, \{|m|\}_{k_1} : m \)
A simple attack
A simple attack

set_w(h)
A simple attack

\[ \text{set}_w(h) \]
A simple attack

set_w(h)

wrap(h,h)
A simple attack

\[ \text{set}_w(h) \quad \text{wrap}(h,h) \quad c = \{|k|\}_k \]
A simple attack

\[ \text{set}_w(h) \rightarrow \text{wrap}(h,h) \rightarrow c = \{ k \}^c_k \rightarrow \text{set}_d(h) \]
A simple attack

\[
\text{set\_w}(h) \rightarrow \text{wrap}(h,h) \rightarrow c = \{|k|\}_k \rightarrow \text{set\_d}(h)
\]
A simple attack

\[
\begin{align*}
&\text{set}_w(h) \\
&\text{wrap}(h,h) \\
&c = \{ | k \rangle \}_k \\
&\text{set}_d(h) \\
&\text{decrypt}(h,c)
\end{align*}
\]
A simple attack

\[ \text{set\_w}(h) \rightarrow \text{wrap}(h,h) \rightarrow c = \{|k|\}_k \rightarrow \text{set\_d}(h) \rightarrow \text{decrypt}(h,c) \rightarrow k \]
A simple attack

\[
\text{set}_w(h) \rightarrow \text{wrap}(h,h) \rightarrow \text{c} = \{|k|\}_k \rightarrow \text{set}_d(h) \rightarrow \text{decrypt}(h,c) \rightarrow \text{k !!}
\]
Policies

init \rightarrow \text{wrap}

init \rightarrow \text{wrap}

init \rightarrow \text{dec}

init \rightarrow \text{dec}

init \rightarrow \text{wrap}

init \rightarrow \text{dec}
Model as a protocol

Attacker
Model as a protocol

state (keys+attrs) grows non-monotonically → facts can disappear
Model as a protocol

state (keys+attrs) grows non-monotonically → facts can disappear

capture causality
Model as a protocol

state (keys+attrs) grows non-monotonically
→ facts can disappear

capture causality  avoid state explosion
Existing Approaches
Existing Approaches

• ProVerif\textsuperscript{[B01]} (the most popular):
  • translate protocol in pi-calculus to Horn clauses, then perform resolution
    \[
    \text{mess}(c, m_1) \land \ldots \land \text{mess}(c, m_n) \rightarrow \text{mess}(c, m_{\text{out}})
    \]
  • sound, but not complete
  • might diverge, but verifies most stateless protocols
State and Horn Clauses

- direct encoding: private channels
- Abstractions: translations into Horn clauses
  - StatVerif [ARR11]
  - abstraction by set-membership [M10]
- Strand Spaces w/ state [G12]
State and Horn Clauses

- direct encoding: private channels
- Abstractions: translations into Horn clauses
  - StatVerif [ARR11]
  - abstraction by set-membership
  - Strand Spaces w/ state [G]

\[
\text{fact persists} \quad \text{mess}(c, M)
\]
State and Horn Clauses

- direct encoding: private channels
- Abstractions: translations into Horn clauses
  - StatVerif [ARR11]
  - abstraction by set-membership [M10]
- Strand Spaces w/ state [G12]
State and Horn Clauses

- direct encoding: private channels
- Abstractions: translations into Horn clauses
  - StatVerif [ARR11]
  - abstraction by set-membership
  - Strand Spaces w/ state

Put state into facts
\[ \text{mess}(\sigma, c, M) \]
only fixed number of cells
State and Horn Clauses

- direct encoding: private channels
- Abstractions: translations into Horn clauses
  - StatVerif [ARR11]
  - abstraction by set-membership [M10]
- Strand Spaces w/ state [G12]
State and Horn Clauses

- direct encoding
- Abstractions: translations into Horn clauses
- StatVerif [ARR11]
- abstraction by set-membership [M10]
- Strand Spaces w/ state [G12]

abstract only certain objects

\[ senc(k, m) \rightarrow senc(val(0, 1, 0), m) \]

\((k \notin Init, k \in Wrap, k \notin Dec)\)

no key-compromise, re-import
State and Horn Clauses

- direct encoding: private channels
- Abstractions: translations into Horn clauses
  - StatVerif [ARR11]
  - abstraction by set-membership [M10]
- Strand Spaces w/ state [G12]
State and Horn Clauses

- direct encoding
- Abstractions: translations into Horn clauses
  - StatVerif [ARR11]
  - abstraction by set-membership [M10]
- Strand Spaces w/ state [G12]

captures causality (protocol runs as acyclic graphs)
no tool support
Tamarin

[SMCB 10]
How tamarin works

- State consists of permanent and linear facts
- Multiset Rewriting Rules for:
  - Protocol
  - Message Deduction
  - Equational Theory
How tamarin works

Rules describing the adversary:

\[ MD = \{ \text{Out}(x) \rightarrow [\cdot] \rightarrow !K(x), \quad !K(x) \rightarrow [ K(x) ] \rightarrow \text{In}(x) \} \]
\[ \cup \{ [\cdot] \rightarrow !K(x:pub), \quad \text{Fr}(x:fresh) \rightarrow [\cdot] \rightarrow !K(x:fresh) \} \]
\[ \cup \{ !K(x_1), \ldots \rightarrow K(x_k) \rightarrow [\cdot] \rightarrow K(f(x_1, \ldots, x_k)) \mid f \in \sum_{DH}^k \} \]
How tamarin works

\[ l \rightarrow [a] \rightarrow r \]  
\hspace{1cm} \text{ground instantiation of a rule}

\( S \)  
\hspace{1cm} \text{contains facts from } l

\( S' \)  
\hspace{1cm} \text{is } S \text{ minus old and plus new facts}

\( S' \xrightarrow{a} S' \)

\text{traces}(P)  
\hspace{1cm} \text{all possible sequences of actions}
constraint solving

• property is negated (counter example)
• translated into graph constraints
• refine constraints (case distinction)
• arrive at solved constraint system:
  • report attack
• no refinements to apply anymore:
  • successful verification
constraint solving

• sound+complete, but might diverge

• example protocol:

\{ \boxed{\vdash A(1), A(x) \quad \boxed{\vdash A(x)} } \}
Yubikey Protocol
Yubikey-protocol

- larger case study of a one-time password protocol
Yubikey-protocol

Network

\[ \text{usb} \rightarrow \text{enc(otp,k)} \]

\[ \text{enc(otp,k)} 
\]

\[ \text{ok} \]

\[ \text{counter larger than last one saved} \]
Yubikey-protocol

are replay attacks possible?

\[ \text{otp} = \langle \text{id}, \text{counter}, ... \rangle \]

counter larger than last one saved
Yubikey-protocol

Network

\[\text{enc(otp,k)} \rightarrow \text{enc(otp,k)}, \text{enc(k,master)} \rightarrow \text{counter values, timestamp} \rightarrow \text{enc(otp,k)} \rightarrow \text{ok}\]
Yubikey-protocol

Is this an improvement?

enc(otp,k), enc(k, master)

counter values, timestamp

enc(otp,k)

Network

usb

enc(otp,k)

ok
Yubikey-protocol

• larger case study of a one-time password protocol

• tamarin can show (with manual intervention) that replay-attacks are impossible

• tamarin can show that the YubiHSM provides secrecy of the otp-keys in case of server compromise (if configured carefully, 2 servers necessary)
A calculus for stateful protocols
A calculus for stateful protocols

• extend ProVerif’s input language with operators for state manipulation:
  • write to map-like structure
  • locking!
Wrap/Dec

!(
  in('create');
  new h; new k;
  event NewKey(h,k);
  insert '<key',h>,k;
  insert '<att',h>, 'init';
  out(h)
)

Mittwoch, 10. Juli 13
Wrap/Dec (ctd.)

!( in('<set_dec',h>);
  lock '<att',h>;
  lookup '<att',h> as a in
    if a='init' then
      delete '<att',h>;
      insert '<att',h>, 'dec';
  unlock '<att',h> )
Why?

• easier for novices (and ProVerif experts)

• large protocols are tedious in MSR

• important issue: rules are atomic, locking issues are not in the model (i.e. race conditions could be overseen)

• Yubikey protocol: 49 (translated) VS 4 rules (manual)
Challenges

• "Digestibility" - translated rules should allow for easy verification in tamarin

• Completeness + Soundness
A gist of the translation

• every node in process tree represented by (at least) one rule

• a fact \( \text{state}_p(v_1, \ldots, v_n) \) is present if statement at position \( p \) with can be executed next

• \textbf{ex:} \( \text{state}_p(\ldots) \rightarrow \text{state}_{p.1}(\ldots), \text{state}_{p.2}(\ldots) \)

• carefully crafted axioms
## Case studies

<table>
<thead>
<tr>
<th>Example</th>
<th>Typing Lemmas</th>
<th>Automated Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security API à la PKCS#11</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>Yubikey Protocol [19, 28]</td>
<td>3</td>
<td>no</td>
</tr>
<tr>
<td>GJM Contract-Signing protocol [3, 15]</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>Mödersheim’s Example (locks/inserts) [22]</td>
<td>0</td>
<td>little interaction</td>
</tr>
<tr>
<td>Mödersheim’s Example (embedded MSRs) [22]</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>Needham-Schroeder-Lowe [21]</td>
<td>1</td>
<td>yes</td>
</tr>
</tbody>
</table>
Further Information


Steve Kremer, Robert Künneumann. Automated analysis of security protocols with global state. (In submission)
Conclusions

• protocols can be represented using Horn clauses

• very efficient (resolution) and sufficiently precise for most stateless protocols

• stateful protocols need more precision

• MSRs as workhorse, high-level representation to avoid mistakes
Thanks for your attention.


