Secure sessions

Karthikeyan Bhargavan  Ricardo Corin  Pierre-Malo Deniélou
Cédric Fournet  James J. Leifer

29 April 2010
Joint Institutes Workshop, Orsay
Goal

Make it simple to write *distributed programs* that engage in *orchestrated patterns* of *secure* communication between *multiple* peers.

Alice

Bob

Charlie
Goal

Make it simple to write *distributed programs* that engage in *orchestrated patterns* of *secure* communication between *multiple* peers.

Alice

Bob

Charlie

Piece of cake! (Assuming we control the network and all the peers.)
But the network is not under our control...
...and our peers may not be trustworthy

"On the Internet, nobody knows you're a dog."
Secure distributed programming

Only realistic security assumption:
The network and any coalition of peers are potentially malicious.
Only realistic security assumption:
The network and any coalition of peers are potentially malicious.

Designing a (correct) security protocol by hand is hard:
- involves low-level, error-prone coding below communication abstractions,
- depends on global message choreography,
- needs to protect against coalitions of compromised peers.
Secure distributed programming

Only realistic security assumption:
The network and any coalition of peers are potentially malicious.

Designing a (correct) security protocol by hand is hard:
- involves low-level, error-prone coding below communication abstractions,
- depends on global message choreography,
- needs to protect against coalitions of compromised peers.

Therefore, we propose:
- to automatically generate tailored cryptographic protocols protecting against the network and compromised peers;
- to hide implementation details and provide mechanised proofs of correctness.
Sessions (contracts, conversations, workflows, ...)

Text representation:

role \( c = \text{send} \ Request\{c,w,q\} \);
loop: \( \text{recv} \ [ \text{Reply}\{x\} \rightarrow \text{send} \ \text{Extra}\{q\};\text{loop} + \text{Enough} ] \)

role \( w = \text{recv} \ Request\{c,w,q\} \rightarrow\)
loop: \( \text{send} \ ( \text{Reply}\{x\}; \ \text{recv} \ \text{Extra}\{q\} \rightarrow \text{loop} + \text{Enough} ) \)
Sessions (contracts, conversations, workflows, ...)

\[(c, w, q) \text{Request}(c, w, q)\] 

\[\text{Enough}()\] 
\[\text{Reply}(x)\] 
\[\text{Extra}(q)\]

Execution

Labels:

Store:
\[c:\]
\[w:\]
\[q:\]
\[x:\]
**Sessions** (contracts, conversations, workflows, ...)

\[
(c, w, q) \xrightarrow{} \text{Request}(c, w, q)
\]

- If \(c\) receives \(c\) from \(w\) and \(q\) from \(x\), then \(w\) sends \(\text{Reply}(x)\) to \(c\).
- \(c\) sends \(\text{Extra}(q)\) to \(c\).

**Execution**

**Labels:** Request

**Store:**
- \(c\): Alice
- \(w\): Bob
- \(q\): “Gone with the wind”
- \(x\):
**Sessions** (contracts, conversations, workflows, ...)

(c, w, q) Request(c, w, q) → w

- () Enough()
- (x) Reply(x)
- (q) Extra(q)

(c)

---

**Execution**

Labels: Request-Reply

Store:
- c: Alice
- w: Bob
- q: “Gone with the wind”
- x: “8 euros”
Sessions (contracts, conversations, workflows, ...)
Sessions (contracts, conversations, workflows, ...)

(c, w, q) Request(c, w, q)

(w) Reply(x)

(c) Extra(q)

(c) Enough()

Execution

Labels: Request-Reply-Extra-Reply

Store:
- c: Alice
- w: Bob
- q: “In stock?”
- x: “yes”
Sessions (contracts, conversations, workflows, ...)

(c, w, q) Request(c, w, q)

(c, w, q) Request(c, w, q)

(product identifier)

() Enough()

(x) Reply(x)

(q) Extra(q)

Execution

Labels: Request-Reply-Extra-Reply-Extra

Store:
- c: Alice
- w: Bob
- q: “Delivery date?”
- x: “yes”
Sessions (contracts, conversations, workflows, ...)

$c \xrightarrow{(c, w, q) \text{Request}(c, w, q)} w \xrightarrow{(x) \text{Reply}(x)} c$

$c \xrightarrow{(q) \text{Extra}(q)} w \xrightarrow{(q) \text{Extra}(q)} c$

$(q) \text{Extra}(q) \xrightarrow{(q) \text{Extra}(q)} c$

Execution

Labels: Request-Reply-Extra-Reply-Extra-Enough

Store:
- $c$: Alice
- $w$: Bob
- $q$: “Delivery date?”
- $x$: “yes”

Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Deniès, Cédric Fournet, James J. Leifer
Expressivity

- Loops, branching, value passing, and value rebinding (as we already saw)

- Commitment “coin flips by telephone” (c commits to x without prior knowledge of y; likewise, w chooses y without knowledge of x)

- Dynamic principal binding (the proxy p gets to choose the web server w based on the client c and her login credentials q)
Threats against session integrity

**Powerful Attacker model**
- can spy on transmitted messages
- can join a session as any role
- can initiate sessions
- can access the libraries (networking, crypto)
- cannot forge signatures

**Attacks against an insecure implementation**
- (Integrity) Rewrite Offer by Reject
- (Replay) Intercept Reject and replay old Offer, triggering a new iteration
- (Sender authentication) send Confirm to \( o \) without having received an Accept
- ... and many more against the store
Protocol outline

Principles of our protocol generation

1. Each edge is implemented by a unique concrete message.
2. We want static message handling for efficiency.

Against replay attacks

- between session executions: session nonces
- between loop iterations: time stamps
- at session initialisations: anti-replay caches

Against session flow attacks

- Signatures of the entire message history (optimisations possible ...)

Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Denielou, Cédric Fournet, James J. Leifer
Optimisation: visibility

Do we really need to include a complete signed history in every message?

Execution paths: which signatures to convince the receiver?

- Request–Contract–Reject
- Request–Contract–Offer–Change–Offer–Change
- Request–Contract–(Offer–Change)$^n$–Reject–Abort

Visibility: at most one signature from each of the previous roles is enough.
Message format

- Session code
- Session Id
- Nonce
- Time stamp
- Payload
- Hashes
- Header:
  - sid
  - ts
  - st
- Hashes:
  - hx
  - hy
- MAC
- MACs

Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Denielou, Cédric Fournet, James J. Leifer

Secure sessions
An extension of ML with sessions

ML Application code

Session declarations

ML Application code

Session implementation

ML compiler

Concrete Model

S2ml, A secure session compiler

Networking & Cryptography

Concrete

Executable
An extension of ML with sessions

S2ml, A secure session compiler

Networking & Cryptography

Symbolic

Symbolic Model

formally verified code

ML compiler

Session declarations

Session implementation

ML Application code

ML Application code
An extension of ML with sessions

S2ml, A secure session compiler

Networking & Cryptography

Symbolic

Concrete

formally verified code

ML compiler

Executable
Theorem (Session Integrity)

For any run of a $S_1$....$S_n$-system, there is a partition of the compliant events such that each equivalence class coincides with a compliant subtrace of a session $S_i$ from $S_1$...$S_n$. 
Theorem (Session Integrity)

For any run of a $S_1$....$S_n$-system, there is a partition of the compliant events such that each equivalence class coincides with a compliant subtrace of a session $S_i$ from $S_1$...$S_n$.

All events: $\text{[diagram]}$

Compliant events: $\text{[diagram]}$

...corresponding to $S_1$ events: $\text{[diagram]}$

...and $S_2$ events: $\text{[diagram]}$
## Performance evaluation

### Performance of the code generation

<table>
<thead>
<tr>
<th>Session S</th>
<th>Roles</th>
<th>File .session (loc)</th>
<th>Application (loc)</th>
<th>Local graph (loc)</th>
<th>Graph (loc)</th>
<th>Graph S.mli (loc)</th>
<th>Graph S.ml (loc)</th>
<th>Compilation (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>2</td>
<td>5</td>
<td>21</td>
<td>8</td>
<td>12</td>
<td>19</td>
<td>247</td>
<td>1.26</td>
</tr>
<tr>
<td>Rpc</td>
<td>2</td>
<td>7</td>
<td>25</td>
<td>10</td>
<td>18</td>
<td>23</td>
<td>377</td>
<td>1.35</td>
</tr>
<tr>
<td>Forward</td>
<td>3</td>
<td>10</td>
<td>33</td>
<td>12</td>
<td>25</td>
<td>34</td>
<td>632</td>
<td>1.66</td>
</tr>
<tr>
<td>Auth</td>
<td>4</td>
<td>15</td>
<td>45</td>
<td>16</td>
<td>38</td>
<td>49</td>
<td>1070</td>
<td>1.86</td>
</tr>
<tr>
<td>Ws</td>
<td>2</td>
<td>7</td>
<td>33</td>
<td>12</td>
<td>24</td>
<td>25</td>
<td>481</td>
<td>1.36</td>
</tr>
<tr>
<td>Wsn</td>
<td>2</td>
<td>15</td>
<td>44</td>
<td>13</td>
<td>42</td>
<td>29</td>
<td>782</td>
<td>1.50</td>
</tr>
<tr>
<td>Wsne</td>
<td>2</td>
<td>19</td>
<td>45</td>
<td>15</td>
<td>48</td>
<td>31</td>
<td>881</td>
<td>1.90</td>
</tr>
<tr>
<td>Shopping</td>
<td>3</td>
<td>29</td>
<td>70</td>
<td>21</td>
<td>85</td>
<td>49</td>
<td>1780</td>
<td>2.43</td>
</tr>
<tr>
<td>Conf</td>
<td>3</td>
<td>48</td>
<td>86</td>
<td>37</td>
<td>181</td>
<td>78</td>
<td>3451</td>
<td>3.32</td>
</tr>
<tr>
<td>Loi</td>
<td>6</td>
<td>101</td>
<td>189</td>
<td>57</td>
<td>310</td>
<td>141</td>
<td>7267</td>
<td>6.29</td>
</tr>
</tbody>
</table>

### Performance of the generated code for Conf (10 000 messages)

<table>
<thead>
<tr>
<th>Version</th>
<th>Time</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unprotected (no key establishment)</td>
<td>1.31 s</td>
<td>0 %</td>
</tr>
<tr>
<td>Don’t sign but do cache checking</td>
<td>1.43 s</td>
<td>9 %</td>
</tr>
<tr>
<td>Sign but don’t verify</td>
<td>1.66 s</td>
<td>27 %</td>
</tr>
<tr>
<td>Fully protected</td>
<td>1.77 s</td>
<td>35 %</td>
</tr>
</tbody>
</table>
Conclusion

- Security protocols are hard to write by hand. They are long, complicated, difficult to verify, and fragile in the face of specification change.
- Automatic generation with mechanised verification is the future!
Conclusion

- Security protocols are hard to write by hand. They are long, complicated, difficult to verify, and fragile in the face of specification change.
- Automatic generation with mechanised verification is the future!

We have:

- designed a high-level session language,
- built a compiler for generating secure implementations from session specifications,
- mechanised the verification of the resulting security protocols (executable code not just models!)

http://www.msr-inria.inria.fr/projects/sec/sessions/
Conclusion

- Security protocols are hard to write by hand. They are long, complicated, difficult to verify, and fragile in the face of specification change.
- Automatic generation with mechanised verification is the future!

We have:
- designed a high-level session language,
- built a compiler for generating secure implementations from session specifications,
- mechanised the verification of the resulting security protocols (executable code not just models!)

http://www.msr-inria.inria.fr/projects/sec/sessions/

Thank you and *bon appétit*!