StatVerif: Modelling protocols that involve persistent state

Mark D. Ryan
University of Birmingham

Joint work with Myrto Arapinis, Stéphanie Delaune, Steve Kremer, Joshua Phillips and Graham Steel

7–8 December 2011
Outline

- The ProVerif method
- Protocols with persistent state
  - The TPM
- StatVerif
Verifying cryptographic protocols

“Provable/computational security”
1. Computationally bounded (polynomial) attacker
2. Exact cryptographic operations on bitstrings
3. Bitstring (more concrete) model
4. Prove difficulty of violating security property is equivalent to solving a hard problem

“Formal/symbolic methods”
1. Idealised (worst case) attacker
2. Idealised (best case) perfect cryptography
3. Symbolic (more abstract) model of protocol
4. Prove impossibility of violating security property within the model
Attacker model

We model a very powerful attacker, with “Dolev-Yao” capabilities:

- it completely controls the communication channels, so it is able to record, alter, delete, insert, redirect, reorder, and reuse past or current messages, and inject new messages. (The network is the attacker.)
- manipulate data in arbitrary ways, including applying crypto operations provided has the necessary keys.
- It controls dishonest participants.

“It’s always better to assume the worst. Assume your adversaries are better than they are. Assume science and technology will soon be able to do things they cannot yet. Give yourself a margin for error. Give yourself more security than you need today.” - Bruce Schneier
Coding protocols as processes

Original handshake protocol:

```plaintext
let Server =
  in (ch, pkC');
  new k;
  out (ch, enc(pkC', sign(skS, k ) ));
  in (ch, m);
  0.
```
The handshake protocol in full

free ch.

(* Public key cryptography *)
fun pk/1.
fun enc/2. fun dec/2.
equation dec(x, enc(pk(x), y) ) = y.

(* Signatures *)
fun sign/2. fun checksign/2. fun getmess/1. fun ok/0.
equation checksign(pk(x), sign(x,y)) = ok.
equation getmess(sign(x,y)) = y.

(* Shared-key cryptography *)
fun senc/2. fun sdec/2.
equation sdec(senc(x,y),x) = y.
let Server =
    in (ch, pkC');
    new k;
    out (ch, enc(pkC', sign(skS, k)));
    in (ch, m);
    0.

let Client =
    in (ch, pkS');
    in (ch, m);
    let m' = dec(skC, m) in
    if checksign(pkS', m') = ok then
        let k' = getmess(m) in
        if pkS' = pkS then
            out (ch, senc(k', s)).
The applied pi calculus can model the following:

- Reachability properties (e.g., secrecy)
- Correspondence assertions (e.g., authentication)
- Observational equivalence (e.g., strong secrecy; for instance, ballot secrecy)
Handshake protocol - analysis

- $C$ publishes her public key
- $I$ starts a session with $S$
- $I$ learns $\text{sign}_{skS}(k)$ and $k$
- $I$ replays $\text{sign}_{skS}(k)$ in a session with $S$
- $I$ is able to output secret $s$

Adversary process $I$

\begin{align*}
\text{in} & \ (c, \ xPK); \\
\text{out} & \ (c, \ pkM); \\
\text{in} & \ (c, \ y); \\
\text{let} \ \text{sig} = \text{dec}_{skM}(y) \ \text{in} \\
\text{out} & \ (c, \ \text{enc}_{xPK}(\text{sig})); \\
\text{in} & \ (c, \ z); \\
\text{out} & \ (c, \ \text{sdec}_{\text{getmsg}(\text{sig})}(z))
\end{align*}
Protocols with persistent state
Agents that have persistent state:

- Web servers, database servers, . . .
- Hardware tokens
  - Smart cards: capabilities, . . .
  - RFID tags: their identity, . . .
  - TPM: PCR values, session nonces, . . .
  - HSM: PIN codes, . . .
- Trusted party in contract signing protocols
- VANETs
- . . .
The trusted platform module
Digital rights management

unforgeable configuration report

Secure environment
Richard Stallman
Creator of GNU, Emacs, GCC, GPL, the Free Software Foundation

“With a plan they call trusted computing, large media corporations, together with computer companies such as Microsoft and Intel, are planning to make your computer obey them instead of you.”

He calls it “treacherous computing”.
“TC can support remote censorship. In its simplest form, applications may be designed to delete pirated music under remote control.”

“In 2010 President Clinton may have two red buttons on her desk - one that sends the missiles to China, and another that turns off all the PCs in China.”

He also talks of commercial bullying, economic warfare and political censorship.
Attestation from cloud
The TPM has 24 platform configuration registers, PCRs.

**Updating a PCR**

The command `TPM_Extend(PCR p, Data x)` effects the assignment

\[ p := \text{SHA-1}(p \parallel x) \]
StatVerif
StatVerif syntax: processes

\[
P, Q ::= 
\begin{align*}
& \text{out}(M, N); P \\
& \text{in}(M, x); P \\
& P \parallel Q \\
& !P \\
& \text{new } a; P \\
& \text{let } x = g(M_1, \ldots, M_n) \text{ in } P \text{ else } Q \\
& \text{if } M = N \text{ then } P \text{ else } Q \\
& [s \mapsto M] \\
& \text{read } s \text{ as } x; P \\
& s := M; P \\
& \text{lock}; P \\
& \text{unlock}; P
\end{align*}
\]

\textbf{processes} \hspace{1cm} \textbf{output} \\
\hspace{1cm} \textbf{input} \\
\hspace{1cm} \textbf{parallel composition} \\
\hspace{1cm} \textbf{replication} \\
\hspace{1cm} \textbf{restriction} \\
\hspace{1cm} \textbf{destructor application} \\
\hspace{1cm} \textbf{conditional} \\
\hspace{1cm} \textbf{state cell} \\
\hspace{1cm} \textbf{read} \\
\hspace{1cm} \textbf{write} \\
\hspace{1cm} \textbf{begin locked section} \\
\hspace{1cm} \textbf{end locked section}
let Server =
in (ch, x);
new n;
out (ch, enc(k, (x,n)));

attacker:x → attacker:enc(k[], (x,n[x]));
let Server =
    in (ch, x);
    new n;
    out (ch, enc(k, (x,n)));

attacker:x → attacker:enc(k[], (x,n[x]));

attacker:u,x → attacker:u,enc(k[], (x,n[x]));
let Server =
    in (ch, x);
    u := h(u,x);

attacker:u,x ∧ attacker:u,y → attacker:h(u,x),y;
The translation of a StatVerif process generates clauses built around the following two predicates

- \( \text{att}(\tilde{M}, N) \) means that state \( \tilde{M} \) is reachable and in that state the attacker knows the value \( N \);

- \( \text{mes}(\tilde{M}, K, N) \) means that state \( \tilde{M} \) is reachable and in that state the value \( N \) is available on channel \( K \).
Attacker clauses: constructors and destructors

- The attacker can build new messages by applying any **constructor** to messages he knows.

For each constructor \( f(M_1, \ldots, M_n) \)
\[
\text{att}(xs, M_1) \land \cdots \land \text{att}(xs, M_n) \rightarrow \text{att}(xs, f(M_1, \ldots, M_n))
\]

**Asymmetric encryption**
\[
\text{att}(xs, xk) \land \text{att}(xs, xm) \rightarrow \text{att}(xs, \text{aenc}(xk, xm))
\]

- The attacker can analyse messages by applying any **destructor** to messages he knows.

For each destructor \( g(M_1, \ldots, M_n) \rightarrow M \)
\[
\text{att}(xs, M_1) \land \cdots \land \text{att}(xs, M_n) \rightarrow \text{att}(xs, M)
\]

**Asymmetric-key decryption**
\[
\text{att}(xs, xk) \land \text{att}(xs, \text{aenc}(\text{pbk}(xk), xm)) \rightarrow \text{att}(xs, xm)
\]
The attacker can **send messages** on public channels

\[
\text{att}(xs, xc) \land \text{att}(xs, xm) \rightarrow \text{mes}(xs, xc, xm)
\]

The attacker can **eavesdrop** on public channels

\[
\text{att}(xs, xc) \land \text{mes}(xs, xc, xm) \rightarrow \text{att}(xs, xm)
\]
Consider the protocol new $\tilde{m}; ([s_1 \leftarrow M_1] \mid \cdots \mid [s_n \leftarrow M_n] \mid P)$.

- The attacker can **read** from public state cells

  For all $i \in \{1, \ldots, n\}$
  \[
  \text{att}((xs_1, \ldots, xs_n), s_i[]) \rightarrow \text{att}((xs_1, \ldots, xs_n), xs_i)
  \]

- The attacker can **write** to public state cells

  For all $i \in \{1, \ldots, n\}$
  \[
  \begin{align*}
  &\text{att}((xs_1, \ldots, xs_i, \ldots, xs_n), s_i[]) \land \\
  &\text{att}((xs_1, \ldots, xs_i, \ldots, xs_n), ys_i) \land \\
  &\text{mes}((xs_1, \ldots, xs_i, \ldots, xs_n), zc, zm) \rightarrow \\
  &\text{mes}((xs_1, \ldots, ys_i, \ldots, xs_n), zc, zm)
  \end{align*}
  \]
  \[
  \begin{align*}
  &\text{att}((xs_1, \ldots, xs_i, \ldots, xs_n), s_i[]) \land \\
  &\text{att}((xs_1, \ldots, xs_i, \ldots, xs_n), ys_i) \land \\
  &\text{att}((xs_1, \ldots, xs_i, \ldots, xs_n), zm) \rightarrow \\
  &\text{att}((xs_1, \ldots, ys_i, \ldots, xs_n), zm)
  \end{align*}
  \]
Protocol clauses

\[
\begin{align*}
[0] \rho H \ell \phi_\mu & = 0 \\
[Q_1 \mid Q_2] \rho H \ell \phi_{false} & = [Q_1] \rho H \ell \phi_{false} \cup [Q_2] \rho H \ell \phi_{false} \\
[!Q] \rho H \ell \phi_{false} & = \{ [Q] (\rho \cup \{ a \mapsto a(\ell)\}) H \ell \phi_\mu \} \rho H \ell \phi_{false} \\
\text{if } M = N \text{ then } Q_1 \text{ else } Q_2 & = [Q_1] (\rho H \ell \phi_{false} \cup [Q_2] \rho H \ell \phi_{false}) \\
\text{lock; } Q & = [Q] \rho H \ell \phi_{true} \\
\text{unlock; } Q & = [Q] \rho H \ell \phi_{false} \\
[s_i := M; Q] & = [Q] (\rho H \ell \phi_{false} \cup \{ H \land \text{mes}(\phi_0, \rho(M), \rho(N)) \land \text{att}(\phi_0, \rho(M), \rho(N)) \} \rho H \ell \phi_{false}) \\
[s_i := M; Q] & = [Q] (\rho H \ell \phi_{true} \cup \{ H \land \text{mes}(\phi_0, \rho(M), \rho(N)) \land \text{att}(\phi_0, \rho(M), \rho(N)) \} \rho H \ell \phi_{true}) \\
\text{read } s_i \text{ as } x; Q & = [Q] (\rho \cup \{ x \mapsto vs_i, vs_1 \mapsto vs_1, \ldots, vs_i \mapsto vs_i, vs_n \mapsto vs_n \}) (H \land \text{mes}(\phi_0, \rho(M), \rho(N)) \land \text{att}(\phi_0, \rho(M), \rho(N)) \land \text{vc, vm fresh}) \\
\text{read } s_i \text{ as } x; Q & = [Q] (\rho \cup \{ x \mapsto M_i, vs_1 \mapsto vs_1, \ldots, vs_i \mapsto vs_i, vs_n \mapsto vs_n \}) (H \land \text{mes}(\phi_0, \rho(M), \rho(N)) \land \text{att}(\phi_0, \rho(M), \rho(N)) \land \text{vc, vm fresh}) \\
\end{align*}
\]
Protocol clauses

**Assignments**

\[
\begin{align*}
\llbracket s_i := M; Q \rrbracket \rho H \ell \phi_{false} &= \llbracket Q \rrbracket \rho' H \ell \phi_{false} \\
&\quad \cup \{ H \land \text{mes}(\phi_0, vc, vm) \Rightarrow \text{mes}(\phi_1, vc, vm) \} \\
&\quad \cup \{ H \land \text{att}(\phi_0, vm) \Rightarrow \text{att}(\phi_1, vm) \}
\end{align*}
\]

where \( \phi_0 = (vs_1, \ldots, vs_{i-1}, vs_i, vs_{i+1}, \ldots, vs_n) \),
and \( \phi_1 = (vs_1, \ldots, vs_{i-1}, \rho(M), vs_{i+1}, \ldots, vs_n) \)
with \( vs_1, \ldots, vs_n, vc, vm \) fresh
and \( \rho' = \rho \cup \{ vs_1 \mapsto vs_1, \ldots, vs_n \mapsto vs_n, vc \mapsto vc, vm \mapsto vm \} \)

\[
\begin{align*}
\llbracket s_i := M; Q \rrbracket \rho H \ell \phi_{true} &= \llbracket Q \rrbracket (\rho \cup \{ vc \mapsto vc, vm \mapsto vm \}) H \ell \phi'_{true} \\
&\quad \cup \{ H \land \text{mes}(\phi, vc, vm) \Rightarrow \text{mes}(\phi', vc, vm) \} \\
&\quad \cup \{ H \land \text{att}(\phi, vm) \Rightarrow \text{att}(\phi', vm) \}
\end{align*}
\]

where \( \phi = (M_1, \ldots, M_{i-1}, M_i, M_{i+1}, \ldots, M_n) \),
and \( \phi' = (M_1, \ldots, M_{i-1}, \rho(M), M_{i+1}, \ldots, M_n) \),
and \( vc, vm \) fresh
Main result

**Theorem (The StatVerif compiler is correct)**

Let $M$ be a message. Let $P$ be a protocol of the form

$$\text{new } \tilde{m}; ([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid Q)$$

Clauses($P$) $\vdash$ secrecy($M$) $\Rightarrow$ $P$ $\models$ secrecy($M$)

If $\forall \tilde{K}$ the fact $\text{att}(\tilde{K}, M)$ is not derivable from Clauses($P$), then $P$ preserves the secrecy of $M$. 
Some clauses for commands of the TPM

**TPM_Read:** $\text{att}(x_p, x) \rightarrow \text{att}(x_p, x_p)$

**TPM_CreateWrapKey:**
$\text{att}(x_p, x_{pcr}) \land \text{key}(x_p, x_{sk}, x_{pk}, x_p) \rightarrow$
$\text{att}(x_p, \langle \text{pk}(\text{bindk}[x_{pcr}]), \text{wrap}(x_{pk}, \text{bindk}[x_{pcr}], \text{tpmpf}[], x_{pcr}) \rangle)$

**TPM_LoadKey2:**
$\text{att}(x_p, \text{pk}(x_{key})) \land \text{att}(x_p, \text{wrap}(x_{pk}, x_{key}, \text{tpmpf}[], x_{pcr})) \land$
$\text{key}(x_p, x_{sk}, x_{pk}, x_p) \rightarrow$
$\text{key}(x_p, x_{key}, \text{pk}(x_{key}), x_{pcr})$

**TPM_Unbind:**
$\text{att}(x_p, \text{aenc}(x_{pk}, x_{data})) \land \text{key}(x_p, x_{sk}, x_{pk}, x_p) \rightarrow \text{att}(x_p, x_{data})$

**TPM_Extend:**
$\text{att}(x_p, x_v) \land \text{att}(x_p, x) \rightarrow \text{att}(\text{h}(x_p, x_v), x)$
$\text{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \land \text{att}(x_p, x_v) \rightarrow \text{key}(\text{h}(x_p, x_v), x_{sk}, x_{pk}, x_{pcr})$
Making ProVerif work on the Horn clauses

Safe abstraction: Replace

\[ \text{att}(x_p, x_v) \land \text{att}(x_p, x) \rightarrow \text{att}(h(x_p, x_v), x) \]

with \( n \) instances, in which \( x_p \) is

- \( \text{zero}[] \)
- \( h(\text{zero}[], x_1) \)
- \( h(h(\text{zero}[], x_1), x_2) \)
- \( h(h(h(\text{zero}[], x_1), x_2), x_3) \)
- \( \ldots \)
Current and future work

- **Prototype implementation** by Joshua Phillips

- **More case studies**
  - UMTS protocols,
  - RFID protocols,
  - ...

- Extension to **authentication properties**

- Extension to **observational equivalence**

- Abstractions