StatVerif: verification of stateful processes

Mark D. Ryan

joint work with Myrto Arapinis, Joshua Phillips and Eike Ritter

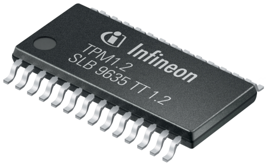
FOSAD 2012

Mutable shared state

Agents that have **mutable state shared between runs**:

- Hardware tokens
  - Smart cards: capabilities, ...
  - RFID tags: pseudonyms, ...
  - TPMs: PCRs, ...
  - HSM: PIN codes, ...
- Trusted parties in contract signing
- Mobile phones: TMSIs, location
- Web servers, database servers, ...
- VANETS: pseudonyms
- ...

The Trusted Platform Module (TPM)



1. Secure storage
2. Platform authentication
3. Platform integrity reporting

- 200M in existence (laptops, desktops, servers)
- specified by Trusted Computing Group ($>$ 700 pages)

TPM functionality

Secure storage

- TPM stores keys and other sensitive data in shielded memory
- A user can store content that is encrypted by keys only available to the TPM

Platform authentication

- Each TPM chip has a unique and secret key
- A platform can obtain keys by which it can authenticate itself

Platform measurement and reporting

- TPM contains some internal memory slots called PCRs, and some keys can be locked to a particular PCR value
- PCR values can be modified, but only in a specific way

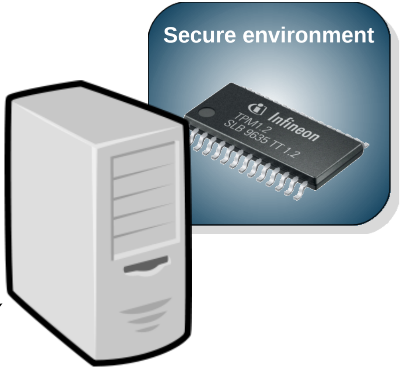
Digital rights management

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".

Attestation from cloud



Cloud server

Coding persistent state in pi calculus

$$
\begin{array}{rcll}
\text{let TPM} & = & \nu p.((!\text{TPMsession}) \mid \text{TPMstate}) & \\[1em]
\text{let TPMsession} & = & p(x). & \text{read \& lock state} \\
 & & c(y). & \text{read command} \\
 & & \ldots & \text{process command} \\
 & & \bar{c}\langle output \rangle. & \text{return output} \\
 & & \bar{p}\langle newstate \rangle. & \text{store \& release state} \\[1em]
\text{let TPMstate} & = & \bar{p}\langle initstate \rangle. & \text{provide initial state} \\
 & & !(p(x). & \text{receive value into state} \\
 & & \quad \bar{p}\langle x \rangle) & \text{return stored state}
\end{array}
$$

This is correct in applied pi calculus. However, because of the abstractions ProVerif makes, one cannot use this encoding to prove properties of the TPM in ProVerif.

Outline of the talk

StatVerif syntax: processes

$P, Q ::=$	**processes**
$out\ (M, N); P$	output
$in\ (M, x); P$	input
$P \mid Q$	parallel composition
$!P$	replication
$new\ a; P$	restriction
$\text{let } x = g(M_1, \ldots, M_n) \text{ in } P \text{ else } Q$	destructor application
$\text{if } M = N \text{ then } P \text{ else } Q$	conditional
$[s \mapsto M]$	state cell
$read\ s_1, \ldots, s_n\ as\ x_1, \ldots, x_n; P$	read
$s_1, \ldots, s_n := M_1, \ldots, M_n; P$	write
$lock\ s_1, \ldots, s_n; P$	lock
$unlock\ s_1, \ldots, s_n; P$	unlock

Example

$$\begin{aligned}
\mathit{let}\ \mathit{TPMsession} = {} & \mathit{lock}\ s; \\
& \mathit{read}\ s\ \mathit{as}\ x_{\mathit{state}}; \\
& \mathit{in}\ (c, x_{\mathit{command}}); \\
& \ldots \mathit{process\ command\ and\ compute\ new\ state} \ldots \\
& s := \mathit{newstate}; \\
& \mathit{unlock}\ s; \\
& \mathit{out}\ (c, \mathit{output})
\end{aligned}$$

Outline of the talk

StatVerif semantics: semantic configurations

Semantic configurations are tuples $(\mathcal{E}, \mathcal{S}, \mathcal{P})$ where:

- $\mathcal{E}$ is a set of names,
- $\mathcal{S}$ maps cell names to their current values, and
- $\mathcal{P}$ is a multiset of pairs (P, λ) with P a process and λ a set of cell names. $s \in \lambda$ iff P has exclusive access to cell s.

Initial configuration for a process P: $(\mathrm{fn}(P), \emptyset, \{(P, \emptyset)\})$

StatVerif semantics: reduction relation

$$\dots$$

$$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{([s \mapsto M], \emptyset)\}) \rightarrow (\mathcal{E}, \mathcal{S} \cup \{s \mapsto M\}, \mathcal{P})$$
$$\textbf{if } s \in \mathrm{dom}(\mathcal{E}) \textbf{ and } s \notin \mathrm{dom}(\mathcal{S})$$

$$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathit{lock}\ s_1, \dots, s_n; P, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \lambda \cup \{s_1, \dots, s_n\})\})$$
$$\textbf{if } \forall (Q, \lambda') \in \mathcal{P}.\ \lambda' \cap \{s_1, \dots, s_n\} = \emptyset$$

$$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathit{unlock}\ s_1, \dots, s_n; P, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \lambda \setminus \{s_1, \dots, s_n\})\})$$

$$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathit{read}\ s_1, \dots, s_n$$
$$\mathit{as}\ x_1, \dots, x_n; P, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P\{\mathcal{S}(s_1)/x_1, \dots, \mathcal{S}(s_n)/x_n\}, \lambda)\})$$
$$\textbf{if } \{s_1, \dots, s_n\} \subseteq \mathrm{dom}(\mathcal{S}) \textbf{ and } \forall (Q, \lambda') \in \mathcal{P}.\ \lambda' \cap \{s_1, \dots, s_n\} = \emptyset$$

$$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(s_1, \dots, s_n :=$$
$$M_1, \dots, M_n; P, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}[s_1 \mapsto M_1, \dots, s_n \mapsto M_n], \mathcal{P} \cup \{(P, \lambda)\})$$
$$\textbf{if } \{s_1, \dots, s_n\} \subseteq \mathrm{dom}(\mathcal{S}) \textbf{ and } \forall (Q, \lambda') \in \mathcal{P}.\ \lambda' \cap \{s_1, \dots, s_n\} = \emptyset$$

Bounded number of cell names

Our translation only applies to StatVerif processes of the form:

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

such that **P** contains no $[s \mapsto M]$.

Outline of the talk

The Horn clauses representation

The translation of a StatVerif process generates clauses built around the following two predicates

- $attacker(\tilde{M}, N)$ means that state $\tilde{M}$ is reachable and in that state the attacker might know the value N;
- $message(\tilde{M}, K, N)$ 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

Asymmetric encryption
$attacker(xs, xk) \rightarrow attacker(xs, pk(xk))$
$attacker(xs, xk) \wedge attacker(xs, xm) \rightarrow attacker(xs, aenc(xk, xm))$

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

Asymmetric-key decryption

$$attacker(xs, xk) \wedge attacker(xs, aenc(pk(xk), xm)) \rightarrow attacker(xs, xm)$$

Attacker clauses: public channels

- The attacker can **send messages** on public channels

$$attacker(xs, xc) \wedge attacker(xs, xm) \rightarrow message(xs, xc, xm)$$

- The attacker can **eavesdrop** on public channels

$$attacker(xs, xc) \wedge message(xs, xc, xm) \rightarrow attacker(xs, xm)$$

Example of protocol clauses: the hardware token

$$
\begin{aligned}
\textit{let config} = {} & \textit{lock } s; \\
& \textit{in } (c, x); \\
& \textit{if } y = \textit{init then} \\
& s := x; \\
& \textit{unlock } s
\end{aligned}
$$

A tag not yet configured ($\textit{if } y = \textit{init then}$) can be configured with any value x sent to it:

$$
\begin{aligned}
& \textit{message}(\textit{init}[], c[], x) \wedge \textit{message}(\textit{init}[], xc, xm) \\
& \quad \wedge \textit{message}(\textit{init}[], yc, ym) \rightarrow \textit{message}(x, yc, ym)
\end{aligned}
$$

$$
\begin{aligned}
& \textit{message}(\textit{init}[], c[], x) \wedge \textit{message}(\textit{init}[], xc, xm) \\
& \quad \wedge \textit{attacker}(\textit{init}[], ym) \rightarrow \textit{attacker}(x, ym)
\end{aligned}
$$

Protocol clauses

$$
\begin{array}{lcl}
\llbracket 0 \rrbracket \rho H \iota \phi \lambda & = & \emptyset \\
\llbracket Q_1 \mid Q_2 \rrbracket \rho H \iota \phi \emptyset & = & \llbracket Q_1 \rrbracket \rho H \iota \phi \emptyset \cup \llbracket Q_2 \rrbracket \rho H \iota \phi \emptyset \\
\llbracket !Q \rrbracket \rho H \iota \phi \emptyset & = & \llbracket Q \rrbracket \rho H \iota \phi \emptyset \\
\llbracket \mathit{new}\ a; Q \rrbracket \rho H \iota \phi \lambda & = & \begin{cases} \llbracket Q \rrbracket (\rho \cup \{a \mapsto a[\iota]\}) H \iota \phi \lambda & \textbf{if } a \in \mathrm{bn}(P) \\ \llbracket Q \rrbracket (\rho \cup \{a \mapsto \mathit{attn}[]\}) H \iota \phi \lambda & \textbf{otherwise} \end{cases} \\
\llbracket \mathit{in}\ (M, x); Q \rrbracket \rho H \iota \phi \lambda & = & \llbracket Q \rrbracket \rho' H' (x :: \iota) \phi' \lambda \\
\multicolumn{3}{l}{\textbf{where } \phi' = \phi[k \mapsto \text{fresh} \mid k \notin \lambda] \textbf{ and } \rho' = \rho \cup \{x \mapsto x\} \cup \{\phi'_k \mapsto \phi'_k \mid k \notin \lambda\} \textbf{ and } H' = H \wedge \mathit{message}(\phi', \rho(M), x)} \\
\llbracket \mathit{out}\ (M, N); Q \rrbracket \rho H \iota \phi \lambda & = & \{H \Rightarrow \mathit{message}(\phi, \rho(M), \rho(N))\} \cup \llbracket Q \rrbracket \rho H \iota \phi \lambda \\
\llbracket \mathit{let}\ x = g(M_1, \ldots, M_n)\ \mathit{in}\ Q_1\ \mathit{else}\ Q_2 \rrbracket \rho H \iota \phi \lambda & = & \llbracket Q_2 \rrbracket \rho H \iota \phi \lambda \\
& \bigcup & \Big\{ \llbracket Q_1 \rrbracket ((\rho\sigma) \cup \rho')(H\sigma)(\iota\sigma)(\phi\sigma)\lambda \ \Big| \\
& & \hfill g(p_1, \ldots, p_n) \rightarrow p \in \mathit{def}(g) \\
& & \hfill \textbf{and } \sigma' = \{z \mapsto \text{fresh} \mid z \in \mathrm{fv}(g(p_1, \ldots, p_n))\} \\
& & \hfill \textbf{and } \sigma = \mathit{mgu}(g(M_1\rho, \ldots, M_n\rho), g(p_1\sigma', \ldots, p_n\sigma')) \\
& & \hfill \textbf{and } \rho' = \{x \mapsto p\sigma'\} \cup \{z\sigma' \mapsto z\sigma' \mid z \in \mathrm{fv}(g(p_1, \ldots, p_n))\} \Big\} \\
\llbracket \mathit{if}\ M = N\ \mathit{then}\ Q_1\ \mathit{else}\ Q_2 \rrbracket \rho H \iota \phi \lambda & = & \llbracket Q_1 \rrbracket (\rho\sigma)(H\sigma)(\iota\sigma)(\phi\sigma)\lambda \cup \llbracket Q_2 \rrbracket \rho H \iota \phi \lambda \\
& & \hfill \textbf{where } \sigma = \mathit{mgu}(\rho(M), \rho(N)\ ) \\
\llbracket \mathit{lock}\ s_{i_1}, \ldots, s_{i_m}; Q \rrbracket \rho H \iota \phi \lambda & = & \llbracket Q \rrbracket (\rho \cup \{\phi'_k \mapsto \phi'_k \mid k \notin \lambda\}) H \iota \phi' (\lambda \cup \{i_1, \ldots, i_m\}) \\
& & \hfill \textbf{where } \phi' = \phi[k \mapsto \text{fresh} \mid k \notin \lambda] \\
\llbracket \mathit{unlock}\ s_{i_1}, \ldots, s_{i_m}; Q \rrbracket \rho H \iota \phi \lambda & = & \llbracket Q \rrbracket (\rho \cup \{\phi'_k \mapsto \phi'_k \mid k \notin \lambda\}) H \iota \phi' (\lambda \setminus \{i_1, \ldots, i_n\}) \\
& & \hfill \textbf{where } \phi' = \phi[k \mapsto \text{fresh} \mid k \notin \lambda] \\
\llbracket \mathit{read}\ s_{i_1}, \ldots, s_{i_m}\ \mathit{as}\ x_1, \ldots, x_m; Q \rrbracket \rho H \iota \phi \lambda & = & \llbracket Q \rrbracket \rho' H' (x_1 :: \cdots :: x_m :: \iota) \phi' \lambda \\
\multicolumn{3}{r}{\textbf{where } \rho' = \rho \cup \{x_j \mapsto \phi'_{i_j} \mid 1 \leq j \leq m\} \cup \{\phi'_k \mapsto \phi'_k \mid k \notin \lambda\} \cup \{vc \mapsto vc, vm \mapsto vm\}} \\
& & \hfill \textbf{and } \phi' = \phi[k \mapsto \text{fresh} \mid k \notin \lambda] \\
& & \hfill \textbf{and } H' = H \wedge \mathit{message}(\phi', vc, vm) \\
& & \hfill \textbf{and } vc, vm \text{ fresh} \\
\llbracket s_{i_1}, \ldots, s_{i_m} := M_1, \ldots, M_m; Q \rrbracket \rho H \iota \phi \lambda & = & \llbracket Q \rrbracket (\rho \cup \{\phi'_k \mapsto \phi'_k \mid k \notin \lambda\}) H \iota \phi'' \lambda \\
& \cup & \{H \wedge \mathit{message}(\phi', vc, vm) \Rightarrow \mathit{message}(\phi'', vc, vm)\} \\
& \cup & \{H \wedge \mathit{attacker}(\phi', vm) \Rightarrow \mathit{attacker}(\phi'', vm)\} \\
& & \hfill \textbf{where } \phi' = \phi[k \mapsto \text{fresh} \mid k \notin \lambda] \\
& & \hfill \textbf{and } \phi'' = \phi'[i_j \mapsto \rho(M_j) \mid 1 \leq j \leq m] \\
& & \hfill \textbf{and } vc, vm \text{ fresh}
\end{array}
$$

Protocol clauses

$$
\begin{array}{lcll}
[\![0]\!]\rho H\iota\phi\lambda & = & \emptyset & \\
[\![Q_1 \mid Q_2]\!]\rho H\iota\phi\emptyset & = & [\![Q_1]\!]\rho H\iota\phi\emptyset \cup [\![Q_2]\!]\rho H\iota\phi\emptyset & \\
[\![!Q]\!]\rho H\iota\phi\emptyset & = & [\![Q]\!]\rho H\iota\phi\emptyset & \\
[\![new\ a; Q]\!]\rho H\iota\phi\lambda & = & \begin{cases} [\![Q]\!](\rho \cup \{a \mapsto a[\iota]\})H\iota\phi\lambda & \textbf{if } a \in \mathrm{bn}(P) \\ [\![Q]\!](\rho \cup \{a \mapsto attn[]\})H\iota\phi\lambda & \textbf{otherwise} \end{cases} & \\
[\![in\ (M, x); Q]\!]\rho H\iota\phi\lambda & = & [\![Q]\!]\rho' H'(x :: \iota)\phi'\lambda & \\
\multicolumn{4}{l}{\textbf{where } \phi' = \phi[k \mapsto \text{fresh} \mid k \notin \lambda] \textbf{ and } \rho' = \rho \cup \{x \mapsto x\} \cup \{\phi'_k \mapsto \phi'_k \mid k \notin \lambda\} \textbf{ and } H' = H \wedge message(\phi', \rho(M), x)} \\
[\![out\ (M, N); Q]\!]\rho H\iota\phi\lambda & = & \{H \Rightarrow message(\phi, \rho(M), \rho(N))\} \cup [\![Q]\!]\rho H\iota\phi\lambda & \\
[\![let\ x = g(M_1, \ldots, M_n)\ in\ Q_1\ else\ Q_2]\!]\rho H\iota\phi\lambda & = & [\![Q_2]\!]\rho H\iota\phi\lambda &
\end{array}
$$

Assignments

$$
\begin{aligned}
[\![s_{i_1}, \ldots, s_{i_m} :=\ & M_1, \ldots, M_m; Q]\!]\rho H\iota\phi\lambda = [\![Q]\!](\rho \cup \{\phi'_k \mapsto \phi'_k \mid k \notin \lambda\})H\iota\phi''\lambda \\
& \cup \{H \wedge message(\phi', vc, vm) \Rightarrow message(\phi'', vc, vm)\} \\
& \cup \{H \wedge attacker(\phi', vm) \Rightarrow attacker(\phi'', vm)\} \\
& \textbf{where } \phi' = \phi[k \mapsto \text{fresh} \mid k \notin \lambda] \\
& \textbf{and } \phi'' = \phi'[i_j \mapsto \rho(M_j) \mid 1 \le j \le m] \\
& \textbf{and } vc, vm \text{ fresh}
\end{aligned}
$$

and $H' = H \wedge message(\phi', vc, vm)$
and vc, vm fresh

$$
\begin{array}{lcl}
[\![s_{i_1}, \ldots, s_{i_m} := M_1, \ldots, M_m; Q]\!]\rho H\iota\phi\lambda & = & [\![Q]\!](\rho \cup \{\phi'_k \mapsto \phi'_k \mid k \notin \lambda\})H\iota\phi''\lambda \\
& \cup & \{H \wedge message(\phi', vc, vm) \Rightarrow message(\phi'', vc, vm)\} \\
& \cup & \{H \wedge attacker(\phi', vm) \Rightarrow attacker(\phi'', vm)\} \\
& & \textbf{where } \phi' = \phi[k \mapsto \text{fresh} \mid k \notin \lambda] \\
& & \textbf{and } \phi'' = \phi'[i_j \mapsto \rho(M_j) \mid 1 \le j \le m] \\
& & \textbf{and } vc, vm \text{ fresh}
\end{array}
$$

Main result

Theorem (The StatVerif compiler is correct)

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

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

such that P contains no $[s \mapsto M]$.

$$\mathbf{Clauses(P)} \vdash \mathbf{secrecy(M)} \quad \Rightarrow \quad \mathbf{P} \models \mathbf{secrecy(M)}$$

If $\forall \tilde{K}$ the fact $attacker(\tilde{K}, M)$ is not derivable from $Clauses(P)$, then P preserves the secrecy of M.

Outline of the talk

Making ProVerif work on the Horn clauses

Making ProVerif work on the Horn clauses

Theorem: There is no selection function such that the saturation of

$$\left\{ \begin{array}{l} attacker(u,x) \rightarrow attacker(h(u,v),x), \\ attacker(u,enc(u,m)) \rightarrow attacker(u,m) \end{array} \right\}$$

will always terminate.

Making ProVerif work on the Horn clauses

Safe abstraction:

Replace

$$attacker(x_p, x_v) \wedge attacker(x_p, x) \rightarrow attacker(h(x_p, x_v), x)$$

with n instances, in which x_p is

- $zero[]$
- $h(zero[], x_1)$
- $h(h(zero[], x_1), x_2)$
- $h(h(h(zero[], x_1), x_2), x_3)$
- ...

Notion of k-stability

Definition: k-stable

A rule R is k-stable if for any substitution θ grounding for R, for any PCR value $u = h(u_1, u_2)$ such that $length_{pcr}(u) > k$ we have that:

- either $(R\theta)[h(u_1, u_2) \rightarrow u_1] = R(\theta[h(u_1, u_2) \rightarrow u_1])$,
- or $(R\theta)[h(u_1, u_2) \rightarrow u_1]$ is a tautology.

Examples

- $attacker(x_p, certkey(aik[], pair(x_{pk}, h(zero[], a_1)))) \rightarrow attacker(x_p, aenc(x_{pk}, s_1))$
- $attacker(x_p, x_v) \wedge attacker(x_p, x) \rightarrow attacker(h(x_p, x_v), x)$

Proposition

Let $\mathcal{R}$ be a finite set of rules and Q be a query such that $\mathcal{R}$ and Q are k-stable. If Q is satisfiable then there exists a k-bounded derivation witnessing this fact.

Syntactic criterion to check k-stability

Lemma

Let $k \geq 0$ be an integer and $R = H \rightarrow C$ be a rule such that:

1. for all $h(v_1, v_2) \in st(R)$, $length_{pcr}(v_1, v_2) \leq k$;
2. for all $h(v_1, v_2) \in st(H)$, we have that $v_1 \notin \mathcal{X}$;
3. for all $h(v_1, v_2) \in st(C)$ such that $v_1 \in \mathcal{X}$, we have that $C[h(v_1, v_2) \rightarrow v_1] \in H$.

Then, we have that the rule is k-stable.

Examples

- $attacker(x_p, certkey(aik[], pair(x_{pk}, h(zero[], a_1)))) \rightarrow attacker(x_p, aenc(x_{pk}, s_1))$
- $attacker(x_p, x_v) \wedge attacker(x_p, x) \rightarrow attacker(h(x_p, x_v), x)$

$\longrightarrow$ Going back to our running example, it is sufficient to consider 1-bounded derivation when checking satisfiability of a query.

Outline of the talk

Conclusion

- Defined the **StatVerif calculus**
- Designed **StatVerif compiler**
- **Proved correctness** of the StatVerif compiler
- Proved security of:
  - our **hardware token device**
  - the Mackenzie *et al.* **optimistic contract signing protocol**
  - **TPM authentication protocols**

Current and future work

- **Prototype implementation**
- **More case studies**
  - UMTS protocols,
  - RFID protocols,
  - ...
- Extension to **authentication properties**
- Extension to **observational equivalence**