Towards Automated Computationally Faithful Verification of Cryptoprotocols

Jan Jürgens
Dep. of Computer Science, TU München
Germany
jürgens@in.tum.de
http://www.jurjens.de/jan

Security Analysis a la Dolev-Yao

Specify protocol participants as processes following Dolev, Yao 1982: In addition to expected participants, model attacker who:
• may participate in some protocol runs,
• knows some data in advance,
• may intercept messages on the public network,
• injects messages that it can produce into the public network

Symbolic Analysis: Limitations

Keys are symbols, crypto-algorithms are abstract operations.
• Can only decrypt with right keys.
• Can only compose with available messages.
• Cannot perform statistical attacks.
Crypto assumed perfect, which it isn’t.

Computationally faithful analysis

Abadi, Rogaway 2000; Abadi, Jürgens 2001:
Symbolic equivalence-based analysis faithful wrt. classical complexity-theoretical model (symmetric encryption, passive adversaries).
Problem: Symbolic model from AJ01 does not directly support automated verification.
Here: Ongoing work to extend above work to automated verification using first-order logic ATP’s (Dolev-Yao style).

Context: „Verisoft“ Project

Goal: Practical application of formal methods.
Planned for 8 years from 7/2003; 12 industrial + academic partners.
Full formal verification from application software down to operating system and processor.
Intended result: Verified C-implementation.
One application example: Biometric authentication protocol (T-Systems).
Goal: Mechanical proof of complexity-theoretical security.

Security analysis in first-order logic

Idea: Given set $P$ of control flow diagrams (of C-programs), approximate set of possible data values known to adversary from above.
Predicate $\text{knows}(E)$ meaning that the adversary may get to know $E$ during the execution of the protocol.
Say that a data value $s$ is secret in $P$ if one cannot derive $\text{knows}(s)$. 
**Crypto Expressions**

Term algebra generated by $\text{Var} \cup \text{Keys} \cup \text{Data}$ and

- $\_ \cdot \_ \cdot \_ \cdot \ldots$ (concatenation)
- $\_ \cdot \_ \cdot \_ \cdot \_ \cdot \ldots$ (inverse key)
- $\_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \ldots$ (encryption)
- $\text{Sign}_\_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \ldots$ (signing)
- $\text{Dec}_\_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \ldots$ (decryption)
- $\text{Ext}_\_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \_ \cdot \ldots$ (extracting from signature)

with appropriate equations.

**FOL rules for Crypto Expressions**

$\forall E_1, E_2 \left( \text{knows}(E_1) \Rightarrow \text{knows}(E_2) \right)$

$\forall E_1, E_2 \left( \text{knows}(E_1) \land \text{knows}(E_2) \Rightarrow \text{knows}(E_1) \lor \text{knows}(E_2) \right)$

$\forall E_1, E_2 \left( \text{knows}(E_1) \land \text{knows}(E_2) \Rightarrow \text{knows}(E_1) \lor \text{knows}(E_2) \right)$

$\forall E_1, E_2 \left( \text{knows}(E_1) \land \text{knows}(E_2) \Rightarrow \text{knows}(E_1) \lor \text{knows}(E_2) \right)$

$\forall E_1, E_2 \left( \text{knows}(E_1) \land \text{knows}(E_2) \Rightarrow \text{knows}(E_1) \lor \text{knows}(E_2) \right)$

$\forall E_1, E_2 \left( \text{knows}(E_1) \land \text{knows}(E_2) \Rightarrow \text{knows}(E_1) \lor \text{knows}(E_2) \right)$

**Model for Security Protocols**

State machine (Mealy automaton) with control states, local variables and transitions between states labeled $(\text{in}(\text{var_in}), \text{cond}(\text{vars}), \text{out}(\text{msg_out}))$

where $\text{msg_in}$ is a local variable to which the incoming message is assigned, $\text{msgs}$ can be variables to which messages have been previously assigned, and $\text{msg_out}$ is an output expression (each possibly empty).

Generate from protocol specs/code.

**Security protocols into 1st order logic**

Define $\text{knows}(E)$ for any $E$ initially known to the adversary (protocol-specific).

Control flow diagram: Each transition of form $(\text{in}(\text{msg_in}), \text{cond}(\text{msgs}), \text{out}(\text{msg_out}))$

is translated (in a nested way) to:

$\forall \text{msg_in}. \left( \text{knows}(\text{msg_in}) \land \text{cond}(\text{msgs}) \Rightarrow \text{knows}(\text{msg_out}) \right)$

(Where for simplicity we use same names for logical and local variables).

Adversary knowledge approximated from above. Can put in more info, then more exact (+ less efficient).

**Example: Proposed Variant of TLS (SSL)**

$\text{knows}(N) \ldots \land \forall \text{exp} \ldots \left( \text{knows}((\text{arg}_S, 1, 3)) \land \text{knows}((\text{arg}_S, 1, 2)) \land \right.$

$\text{snd}((\text{Ext}_S, 1, 2, (\text{arg}_S, 1, 3))) = \text{arg}_S, 1, 2$\n
$\Rightarrow \text{knows}(\text{“arguments of resp method”}) \land \ldots$

**Analysis**

- E-METHOD cap03 single processor running on host ...
- No variant of hand-check tool ...
- Time limit information: 200 total (entering statistics module), problem analysis ...
- First-order problem ...
- Schedule selection: problem is born with equality (class he), schedule: 905 300 597 ...
- Entering next strategy 605 with resource 3 seconds ...
- Analyzing results ...
- Proof found ...
- Time limit information: 296 total / 297 strategy (learning wrappers) ...
-
Computationally faithful?

Works fine for Dolev-Yao style analysis but: doesn't detect partial violation of secrecy.
Add another clause to each implication:
Whenever condition in automaton is reached, all its subterms relevant to its validity are added to adversary knowledge.
Again approximation on the „safe“ side which works fine for practical examples.

Comparison to symbolic AJ01

Equivalence-based approach: „extrinsic“.
Compute observable traces (somehow) and compare. Close to intuitions (but maybe not immediately clear how to efficiently verify eg with atp’s).
Present approach: „intrinsic“. Stay as close to protocol model as possible when trying to detect information flow to enable efficient verification.

The computational view

An encryption scheme consists of algorithms:

\[ \mathcal{K} : \text{Parameter} \times \text{Coins} \rightarrow \text{Key} \]

\[ \mathcal{E} : \text{Key} \times \text{String} \times \text{Coins} \rightarrow \text{Cipher} \]

\[ \mathcal{D} : \text{Key} \times \text{String} \rightarrow \text{Plain} \]

where Parameter \( \equiv 1^* \) (numbers in unary),

Key, Plain, Cipher \( \subseteq \text{String} \).

For all \( \eta \in \text{Parameter}, k \in \mathcal{K}(\eta) \), and \( r \in \text{Coins}, \)

- if \( m \in \text{Plain} \) then \( \mathcal{D}_k(\mathcal{E}_k(m, r)) = m \),
- if \( m \notin \text{Plain} \) then \( \mathcal{D}_k(\mathcal{E}_k(m, r)) = \perp \) (error message)

Indistinguishable Ensembles

A function \( \epsilon : \mathbb{N} \rightarrow \mathbb{R} \) is negligible if for all \( c > 0 \) there exists \( N \) such that \( \epsilon(\eta) \leq \eta^{-c} \) for all \( \eta \geq N \).

An ensemble (or probability ensemble) is a collection of distributions on strings, \( \mathbb{D} = (\mathbb{D}_k), \) one for each \( \eta \).

We say that \( \mathbb{D} \) and \( \mathbb{D}' \) are indistinguishable and write \( \mathbb{D} \approx \mathbb{D}' \), if

\[ \epsilon(\eta) \triangleq \Pr[x \in \mathbb{D}_k : A(\eta, x) = 1] - \Pr[x \in \mathbb{D}'_k : A(\eta, x) = 1] \]

is negligible for all polynomial-time adversaries \( A \).

Secure Encryption (variant)

Let \( \Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}) \) be an encryption scheme, let \( \eta \in \text{Parameter} \) be a security parameter.

Define

\[ \text{Adv}_{\Pi, \eta}(A) \triangleq \Pr[k, k' \stackrel{\$}{\leftarrow} \mathcal{K}(\eta), x \stackrel{\$}{\leftarrow} \mathcal{E}_k() : A^\Pi(k', x) = 1] - \Pr[k, k' \stackrel{\$}{\leftarrow} \mathcal{K}(\eta), x \stackrel{\$}{\leftarrow} \mathcal{E}_k() : A^\Pi(k, x) = 1] \]

Encryption scheme \( \Pi \) is secure if \( \text{Adv}_{\Pi, \eta}(A) \) is negligible for all polynomial-time adversaries \( A \).

(Goldwasser & Micali, Bellare et al.; Abadi & Rogaway)

Repetition concealing, message-length concealing, which-key concealing

Wrong key?

In formal models, decrypting a message with the “wrong” key is a noticeable error. Computational counterpart:

Encryption scheme \( \Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}) \) is confusion-free if for all \( m \in \text{String} \) the probability

\[ \Pr[k, k' \stackrel{\$}{\leftarrow} \mathcal{K}(\eta), x \stackrel{\$}{\leftarrow} \mathcal{E}_k(m) : D^\Pi_k(x) \neq \perp] \]

is negligible.

Related: committing encryption (M. Fischlin)
Computational interpretation

To any set $P$ of control flow graphs assign distribution $[[P]]_{II}$ on input/output histories (given an encryption scheme $II$ and a security parameter $\eta$). Given an initial probability event $\tau$, map each key symbol $K$ to a bitstring $\pi(K)$, using $K(\eta)$. Mark all occurrences of encryptions $(E)_r$ with a different coin symbol $r$. $(E)_r$. Map each coin symbol $r$ to a bit string $\pi(r)$. Then for expressions:

- $[[b]]_{II,\eta} = b$
- $[[K]]_{II,\eta} = \pi(K)$
- $[[M::N]]_{II,\eta} = ([[M]]_{II,\eta}, [[N]]_{II,\eta})$

---

Computational soundness

Let $P$ be a set of state machines that does not generate encryption cycles and $II$ a secure and confusion-free encryption scheme. If a data value $s$ in $P$ is secret then $s$ is computationally secret.

(Still for symmetric encryption against passive adversaries; extension in progress.)

---

Conclusion

Work towards automated verification of security-critical software using first-order logic theorem provers which aims to be

- efficient, powerful
- intuitive, simple
- computationally faithful
- practically applicable

Limitations:

- give up (theoretical) completeness
- complexity theory is also „just“ a theoretical model

---

(Advertisement block)

Use verification in industrial projects with HypoVereinsbank, T-Systems, BMW, ...
Hide logic behind industrial notation UML:
Book: Jan Jürjens, Secure Systems Development with UML, Springer-Verlag, 2004
Summer School "Foundation of Security Analysis and Design", Bertinoro (6-11/9)

More information (slides, tool etc.):
http://www.jurjens.de/jan