On Reactive Simulatability

Matthias Berg / (Michael Backes)  
Saarland U and MPI-SWS  
CoSyProofs 2009
Building Systems on Open Networks

E-Government

On Reactive Simulatability
Cryptography: The Details

Crypto-Toolbox

- Encryption
- Hashfunction
- Signature
- Key establishment

Fact(p*q)  DL(g^x)

Prob[Attack] ≤ …

On Reactive Simulatability
Cryptography: The Details

Crypto-Toolbox

Encryption

Hashfunction

Signature

Key establishment

Proof

On Reactive Simulatability
Formal Methods: The Big Picture

Idealized Crypto
- Signature
- Encryption
- Hashfunction
- Key establishment

But can we justify?

Designed by CAD
Verified by CAV

On Reactive Simulatability
Idea: Sound Abstract Protocol Proofs

Formalize with given interface

Prove per Protocol

Abstract primitives uses Abstract protocol fulfils Abstract goals

Abstract protocol replaces primitives

Concrete primitives uses Concrete protocol fulfils Concrete goals

On Reactive Simulatability
Example

Only this per system

Abstract SecChan \( \rightarrow \) Pay via SecChan \( \rightarrow \) PaySys Integrity

SSL (?) \( \rightarrow \) Pay via SSL \( \rightarrow \) " with error prob

Abstraction

uses \( \rightarrow \) fulfils

replace primitives \( \rightarrow \) Abstraction

General defs

On Reactive Simulatability
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
  - Network characteristics? synchr./asynchr., reliable, secure, etc.
  - Power of the adversary? Passive/active, static/dynamic, secure function evaluation / reactive (!)
  - Realistic scheduling
  - Which other protocols may run concurrently?
  - ...
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time
  - Cryptographic issues: probabilism, error-probabilities, computational restrictions, etc.
  - Abstraction issues: Abstract transition functions, distributed-systems aspects, formal calculi, etc.
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time
- Mathematically rigorous definition of what a “good” abstraction is
  - Intuitive
  - Should fit to a variety of different abstractions/real protocol classes
  - Provable by convenient proof techniques
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time
- Mathematically rigorous definition of what a “good” abstraction is
- Not only hold in isolation but preserve security under composition
  - (Makes the definition “useful”)
  - Make modular analysis of larger protocols possible
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time
- Mathematically rigorous definition of what a “good” abstraction is
- Not only hold in isolation but preserve security under composition
- Should preserve essentially arbitrary security properties
  - Integrity, variants of confidentiality, non-interference, poly-time liveness
  - Tight links to properties shown for symbolic abstractions of crypto
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time.
- Mathematically rigorous definition of what a “good” abstraction is.
- Not only hold in isolation but preserve security under composition.
- Should preserve essentially arbitrary security properties.
- Abstractions should match the intuition for the requirements in mind.
  - Intuitive abstractions, easy to read for non-specialist, thus enabling convenient use in larger protocols.
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time.
- Mathematically rigorous definition of what a “good” abstraction is.
- Not only hold in isolation but preserve security under composition.
- Should preserve essentially arbitrary security properties.
- Abstractions should match the intuition for the requirements in mind.
- Abstractions should be based on the functionality of the protocol, not on its structure.
- Good abstractions for many of useful protocol classes should exist.
Overview: Reactive Simulatability Framework

- Precise system model allowing cryptographic and abstract operations
- Reactive simulatability with composition theorem
- Preservation theorems for security properties
- Concrete pairs of idealizations and secure realizations
- Sound symbolic abstractions (Dolev-Yao models) that are suitable for tool support
- Sound security proofs of security protocols: NSL, Otway-Rees, iKP, (parts of) Kerberos, etc.
- Detailed Proofs (Cryptographic bisimulations with static information flow analysis, … )
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time.
- Mathematically rigorous definition of what a “good” abstraction is.
- Not only hold in isolation but preserve security under composition.
- Should preserve essentially arbitrary security properties.
- Abstractions should match the intuition for the requirements in mind.
- Abstractions should be based on the functionality of the protocol, not on its structure.
- Good abstractions for many of useful protocol classes should exist.

On Reactive Simulatability
Idea: Define Security relative to an ideal task

||
| „implements“ |
| „as secure as“ |

\[ M_1 \geq M_2 \]

Real system

Ideal system (abstraction)

How to define that? What does “every attack” mean? “successfully converted”?

What are good ideal systems? What about concrete security properties, e.g., integrity or secrecy?
Reactive Simulatability – here blackbox

\[ \forall \quad \forall \quad \forall \]

Real system

\begin{align*}
\text{view}_{\text{real}}(H) & \approx \text{view}_{\text{ideal}}(H) \\
\text{Indistinguishability of random variables}
\end{align*}

Ideal system

On Reactive Simulatability
Reactive Simulatability Variants

- Standard simulatability: \( \forall A \ \forall H \ \exists A' \)
- Universal simulatability: \( \forall A \ \exists A \ \forall H \)
- Blackbox simulatability: \( \exists \text{Sim} \ \forall H \ \forall A \ A' = \text{Sim&A} \)
- Perfect / statistic / computational

On Reactive Simulatability
Indistinguishability [Yao_82]

- Families of random variables:
  \[ \left( v_k \right)_{k \in \mathbb{N}} \approx_{\text{poly}} \left( v'_k \right)_{k \in \mathbb{N}} \]

  \[ \approx_{\text{poly}} \iff \forall D \text{ (prob. poly. in first input)}:\]

  \[ \left| \Pr(D(1^k, v_k) = 1) - \Pr(D(1^k, v'_k) = 1) \right| \leq \frac{1}{\text{poly}(k)}. \]
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time
- Mathematically rigorous definition of what a “good” abstraction is
- Not only hold in isolation but preserve security under composition
- Should preserve essentially arbitrary security properties
- Abstractions should match the intuition for the requirements in mind
- Abstractions should be based on the functionality of the protocol, not on its structure.
- Good abstractions for many of useful protocol classes should exist
Base Lemmas of reactive simulatability

- Machine combination is defined and
  - is associative
  - retains poly-time (for strong version)
  - retains sub-machine views

- “As secure as” is transitive. E.g., with composition:

\[ \text{[PW00,PW01]} \]
Composition – One System

Given:

Then this holds:
Proof Idea (Single Composition)

On Reactive Simulatability
Composition – Multiple Systems

Given:

Also this holds:
General Composition Proof via Hybrid Systems

On Reactive Simulatability
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions realizaions at the same time.
- Mathematically rigorous definition of what a “good” abstraction is.
- Not only hold in isolation but preserve security under composition.
- Should preserve essentially arbitrary security properties.
- Abstractions should match the intuition for the requirements in mind.
- Abstractions should be based on the functionality of the protocol, not on its structure.
- Good abstractions for many of useful protocol classes should exist.
Cryptographic Idealization Layers

Symbolic abstractions

Dolev-Yao Model

Larger abstractions

VSS

Certified mail

[GM95]

[PSW00]

Credentia-

Small real abstractions

Secure channels

Auth/sigs as statement database

[PW00, PW01, CK02, BNP02, ...]

[BPW03 ...]

Related: [SM93, P93]

Low-level crypto
(not abstract)

Encryption as E(pk, 1^len)

Real auth/sig’s + integrity lookup

[LMMS98, PW00, C01, ...]

[LMMS98, C01, ...]

Normal cryptographic definitions

On Reactive Simulatability
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time.
- Mathematically rigorous definition of what a “good” abstraction is.
- Not only hold in isolation but preserve security under composition.
- Should preserve essentially arbitrary security properties.
- Abstractions should match the intuition for the requirements in mind.
- Abstractions should be based on the functionality of the protocol, not on its structure.
- Good abstractions for many of useful protocol classes should exist.
Recall Prior Result

- “as secure as” (reactive simulatability)
- for certain versions of \[ \text{puzzle symbols} \] and \[ \text{other symbols} \]
Specification Styles

- Is $\geq$ what people want?

- Often yes, in particular together with

  - E.g., secure channels (see also spi calculus), certified mail

- But not always ...

On Reactive Simulatability
Alternative: Property-based spec.

- E.g., “I want a tight roof on top”: integrity
  - Preserved by “≥”:

- In the RSIM framework: Preservation theorems for integrity, non-interference, poly-time liveness, etc.
Example: Ordered Channels over Unordered Ones

$\text{TH}_{\text{ord}} \quad = \quad \text{A} \quad = \quad \text{TH}_{\text{SecMess}}$

1. Application
2. Preservation theorem

Transitivity

$(\text{rec}_{u,v} \subseteq \text{send}_{u,v})$
What do we need for soundly abstracting?

- Precise system model that permits all “realistic” attacks.
- Capable of reasoning about abstractions/realizations at the same time
- Mathematically rigorous definition of what a “good” abstraction is
- Not only hold in isolation but preserve security under composition
- Should preserve essentially arbitrary security properties
- Abstractions should match the intuition for the requirements in mind
- Abstractions should be based on the functionality of the protocol, not on its structure.
- Good abstractions for many of useful protocol classes should exist

Now: the BPW model (sound Dolev-Yao style library)
Why Formal Methods?

- Automation if
  - Repetitive
  - Tedious
  - Prone to human errors
  - Critical application

- A top candidate: Distributed protocols

- Security variants for 20 years
Protocol Proof Tools

- Almost anything
- Much human interaction

- Special logic fragments for security
- Approximations: correct, not complete

- Fully automatic
- State exploration

On Reactive Simulatability
Automating Security Protocol Proofs

- Even simple protocol classes & properties undecidable
  - Robust protocol design helps
- Full arithmetic is out
- Probability theory just developing for reasoning about larger protocols

So how do current tools handle cryptography?
Dolev-Yao Model

• Idea [DY81]
  – Abstraction as term algebras, e.g., $D_x(E_x(E_x(m)))$
  – Cancellation rules, e.g., $D_x E_x = \varepsilon$
• Well-developed proof theories
  – Abstract data types
  – Equational 1st-order logic
• Important for security proofs:
  – Inequalities! (Everything that cannot be derived.)
  – Known as “initial model”

Important goal: Justify or replace
Dolev-Yao Model – Variants [BPW]

- Operators and equations [EG82, M83, EGS85 ...]
  - Enc, sigs, nonce, payload, pairing, ...
  - Inequalities assumed across operators!
- Untyped or typed
- Destructors explicit or implicit
- Abstraction from probabilism
  - Finite selection, counting, multisets
- Surrounding protocol language
  - Special-purpose, CSP, pi calculus, ...
  [any]
The BPW model – major challenges

- Recall: Term algebra, inequalities
- Major tasks:
  - Represent ideal and real library in the same way to higher protocols
  - Prevent honest users from stupidity with real crypto objects, but don’t restrict adversary
    - E.g., sending a bitstring that’s almost a signature
  - What imperfections are tolerable / must be allowed?
The BPW model - characteristics

• Characteristics
  – “library” of standard crypto primitives
  – tracks content (messages) and knowledge (who knows what)
  – $N$ honest users and adversary manipulate messages indirectly using handles

• Functionality
  – local functions for message construction and access
    (e.g. nonce & key generation, encryption/decryption, sign/verify, pairing/projection)
  – send functions for message transmission
    (user to adversary and vice versa)
  – adversary interface has additional capabilities
    (e.g. create garbage messages, invalid ciphertexts transform signatures)
The BPW model

No crypto outputs! Deterministic!

Commands, payloads, terms? handles

Payloads / test results, terms? handles

Term 1 Term 2 Term 3 Not globally known
For U: $T_{u,1}$ $T_{u,2}$ $T_{u,3}$
For V: - $T_{v,1}$ -
For A: - $T_{a,1}$ -

E
pk pk m

E
pk m

TH

On Reactive Simulatability
The BPW model

U

\( T_{u,4} \leftarrow \text{encrypt}(T_{u,1}, T_{u,3}) \)
send\((V, T_{u,4})\)

V

\text{received}(U, T_{v,2})

get\_type(T_{v,2})

\( T_{v,3} := \text{decrypt}(\ldots) \)

\[ \begin{array}{c|c|c|c|c}
\text{Term 1} & \text{Term 2} & \text{Term 3} & \text{Term 4} \\
\hline
\text{For U:} & T_{u,1} & T_{u,2} & T_{u,3} & \ldots \\
\text{For V:} & - & T_{v,1} & - & \\
\text{For A:} & - & T_{a,1} & - & E \\
\end{array} \]

\[ E \]

\[ \begin{array}{c|c|c|c|c}
\text{pk} & \text{pk} & m & \text{pk} & m \\
\hline
E & E & \text{pk} & \text{pk} & \\
& & m & m & \\
\end{array} \]

TH

On Reactive Simulatability
Main Differences to Standard Dolev-Yao

• Tolerable imperfections:
  – Lengths of encrypted messages cannot be kept secret
  – Adversary may include incorrect messages inside encryptions
  – Signature schemes can have memory
  – Slightly restricted key usage for symmetric encryption

Most imperfections avoidable for more restricted cases
Real Implementation

Commands, payloads, handles

U

pk
$c_1 \leftarrow E(pk, m)$
$c_2 \leftarrow E(pk, m)$

Bitstrings

V

Payloads / test results, handles

$c_1$

A

No crypto outputs!

Real system

On Reactive Simulatability
The Simulator

On Reactive Simulatability
The BPW model in Isabelle

APM/BRSIM proof & NSL case study

- 40 theories (boxes in graph)
- 150 definitions
- 1200 lemmas and theorems
- 18k lines of Isabelle/HOL
- 360 pages PDF documentation

Overall

<table>
<thead>
<tr>
<th>Module</th>
<th>Theories</th>
<th>Pages</th>
<th>Lines</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling &amp; verification tools</td>
<td>9</td>
<td>54</td>
<td>2.7k</td>
<td>7</td>
</tr>
<tr>
<td>DAG-based model + BRSIM</td>
<td>10</td>
<td>119</td>
<td>6k</td>
<td>16</td>
</tr>
<tr>
<td>Term-based model</td>
<td>17</td>
<td>96</td>
<td>4.8k</td>
<td>13</td>
</tr>
<tr>
<td>Term-based NSL</td>
<td>14</td>
<td>136</td>
<td>6.8k</td>
<td>18</td>
</tr>
<tr>
<td>APM + BRSIM</td>
<td>31</td>
<td>266</td>
<td>13.3k</td>
<td>35</td>
</tr>
<tr>
<td>APM-based NSL</td>
<td>9</td>
<td>83</td>
<td>4.2k</td>
<td>11</td>
</tr>
<tr>
<td>Total</td>
<td>90</td>
<td>754</td>
<td>37.8k</td>
<td>100</td>
</tr>
</tbody>
</table>

Abstract protocol model, RSIM proof, and NSL case study

On Reactive Simulatability
Bigger picture and some possible next steps

- Linking crypto and formal methods
  - Extended DY
  - "classic" DY
    - soundness
      - sign
      - pk
      - E
      - pk'
      - m
  - Encoding crypto techniques
    - <...>
    - xs4yuvls38
    - </...>