Calculi and Models for Security

Chiara Bodei, Massimo Bartoletti,
Pierpaolo Degano, Gian-Luigi Ferrari,
and Roberto Zunino

Dipartimento di Informatica,
Università di Pisa, Italy

Pisa, 17-28 Settembre 2007

Models for Security

What is security?

What is security model?

Focus on

- systems (protocols)
- wanted properties

Today, we provide some background

- models (in general)
- models based on process algebras

Models for Security

Two distinct approaches

Models for Security

Two distinct approaches

- Computational Complexity Theory
- Formal Models

Models for Security

Two distinct approaches

- Computational Complexity Theory
  - Algorithms and probability
  - In–depth, detailed view
- Formal Models

Models for Security

Two distinct approaches

- Computational Complexity Theory
  - Algorithms and probability
  - In–depth, detailed view
- Formal Models
  - Process algebras
  - Simpler semantics (abstraction)
  - (Semi)–automatic correctness proofs
  - Stronger assumptions

Models for Security

Two distinct approaches

- Computational Complexity Theory
  - Algorithms and probability
  - In–depth, detailed view
- Formal Models
  - Process algebras
  - Simpler semantics (abstraction)
  - (Semi)–automatic correctness proofs
  - Stronger assumptions
    e.g. perfect cryptography

Computational Model

Pros:

Computational Model

Pros:

- precise, in–depth view
  probability, complexity
- many properties can be expressed
  (“all”)

Computational Model

Pros:

- precise, in–depth view
  probability, complexity
- many properties can be expressed ("all")

Cons:

- too low–level for some purposes
- proving correctness is hard

Formal Models

Pros:

Formal Models

Pros:

- abstraction
  easy to understand/reason about
- (semi–)automatic proofs of correctness

Formal Models

Pros:

- abstraction
  easy to understand/reason about
- (semi–)automatic proofs of correctness

Cons:

- abstraction(!)
- strong assumptions needed

Formal Models

Pros:

- abstraction
  easy to understand/reason about
- (semi–)automatic proofs of correctness

Cons:

- abstraction(!)
- strong assumptions needed
  e.g. *perfect cryptography*
  fixed crypto algebra

The Dolev-Yao Adversary Model

An adversary can

- eavesdrop messages
- intercept messages
- reroute messages
- manipulate messages (“reasonably”)
- impersonate participants

Effectively, the adversary *is* the network.

This can be seen in both models.

Messages

A Formal Model for Messages

Values are terms in an algebra (possibly free)

$$
\begin{array}{rcll}
M, N & ::= & \{M\}_k & \text{encryption} \\
 & | & (M, N) & \text{pair} \\
 & | & k & \text{key} \\
 & | & n & \text{nonce}
\end{array}
$$

The adversary can construct/destruct terms only via some standard operations

Some Standard Operations

Really simple rules:

$$
\begin{aligned}
\mathsf{M}, \mathsf{N} &\longmapsto (\mathsf{M}, \mathsf{N}) \\
(\mathsf{M}, \mathsf{N}) &\longmapsto \mathsf{M} \\
(\mathsf{M}, \mathsf{N}) &\longmapsto \mathsf{N}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{M}, \mathsf{k} &\longmapsto \{\mathsf{M}\}_{\mathsf{k}} \\
\{\mathsf{M}\}_{\mathsf{k}},\ \mathsf{k} &\longmapsto \mathsf{M}
\end{aligned}
$$

Some Standard Operations

Really simple rules:

$$
\begin{aligned}
M, N &\mapsto (M, N) \\
(M, N) &\mapsto M \\
(M, N) &\mapsto N
\end{aligned}
\qquad
\begin{aligned}
M, k &\mapsto \{M\}_k \\
\{M\}_k, \textcircled{k} &\mapsto M
\end{aligned}
$$

The adversary must know the key to decrypt a term. Otherwise, it is *semantically infeasible*. In other words, the cryptosystem is perfect.

Computational Model

Messages = bit strings

Turing Machines for crypto algorithms

- key generation
- encryption, decryption, etc.
- tractable complexity w.r.t. η (security parameter)

Recall complexity classes $\mathcal{P}, \mathcal{NP}$

Actually, probabilistic TM

Keys are $\sim \eta^c$ bits long

Protocols

Computational Model

Network of Turing Machines

- protocol participants
- hostile network (adversary)

Focus on

computational complexity vs. attack probability

Preliminary Definitions

A function $f(\eta)$ is negligible iff it is asymptotically smaller than any rational function.

$$\forall c > 0.\exists \eta_0.\forall \eta > \eta_0.|f(\eta)| < \eta^{-c}$$

Example: $2^{-\eta}$ is negligible, η^{-2} is not

Significant Attacks

Intuitively:

If an adversary wins only with negligible probability, it is not significant.

"luck always wins"

If an adversary is not in $\mathcal{P}$, it is not significant.

"(extreme) power always wins"

(not completely true for some ZK protocols)

Attacks to Algorithms

Naïve attack

Given $\{m\}_k$, deduce m.

Complexity/probability must be taken into account.

Attacks to Algorithms

Naïve attack

Given $\{m\}_k$, deduce m.

Complexity/probability must be taken into account.

Still, not enough!

Discovering the first half of m is still dangerous.

Other Well-known Attack Types

How to distinguish $\{m_0\}_k$ from $\{m_1\}_k$?

Chosen Plaintext Attack

The adversary can use the encryption machine then must guess

Chosen Ciphertext Attack

The adversary can use the encryption and the decryption machine (not on m_i) then must guess

Negligible advantage

$$Adv = \mathbb{P}[\text{informed guess}] - \mathbb{P}[\text{blind guess}] = p_{\text{i.g.}} - 1/2$$

Protocol Computational Security

Similar definitions: e.g.

- indistinguishability from ideal behaviour ($\sim$ without the adversary)
- negligible advantage (e.g. for secrecy)
- execution traces “not too different” from ideal ones
- ...

Protocol Formal Security

Differences from the computational approach:

- no Turing machines
- protocol source code
- no bit strings
- terms in an algebra

Properties

- indistinguishability (w.r.t. formal semantics)
- reachability (eaiser to prove)
- no probability/complexity

Bridging the Gap

Does formal security imply computational security?

- Abadi, Rogaway
  terms equivalence
- Jürjens
  protocol equivalence (passive adversary)
- Pfitzmann, Backes, Waidner, Canetti
  simulatability, universal composability

Modelling Protocols

Many formal methods use *process algebras* (calculi) to specify the protocol logic

- π calculus
- SPI
- applied pi
- LySa
- . . .

π calculus

The π Calculus

A programming language core featuring:

- concurrency (parallelism)
- send/receive primitives
- message passing
- creation of new channels
- creation of new “tags” (e.g. keys)
- mobility ($\sim$ network reconfiguration)

The π Calculus

Warning: many, many variants!
E.g.

- what is a message?
  - an atomic name
  - ...
  - a generic term (e.g. $f(g(x), y)$)
- many ways to define the semantics

Suggestion: focus on the main concepts rather than the details of the particular variant we present here.

Basics

P, Q processes

Semantics through transitions

$$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow \cdots$$

Sometimes, we shall annotate the transitions with some label, to make some event *observable*

Example:

$$P_1 \rightarrow P_2 \xrightarrow{\textbf{boom}} P_3 \rightarrow \cdots$$

We can then answer questions such as “does P_1 make the bomb explode?”

Syntax: nil

$$\mathbf{0} \qquad \text{nil process}$$

- stopped/terminated
- no further interaction

Of course

$$\mathbf{0} \not\rightarrow$$

Syntax: output (send)

$$\overline{a}\langle x\rangle.Q$$

- a is the channel ($\sim$ network address)
- x is the object being sent (message)
- Q is the continuation process (what to do next)

Let's make the message observable:

$$\overline{a}\langle x\rangle.Q \xrightarrow{\overline{a}\langle x\rangle} Q$$

Syntax: input (receive)

$$a(x).Q$$

- a is the channel ($\sim$ network address)
- x is a *variable* for the object being received
- Q is the continuation process (what to do next)
- Q usually depends on x.

Examples:

$$a(x).\overline{b}\langle x\rangle.\overline{c}\langle x\rangle.0$$
$$a(x).\overline{x}\langle b\rangle.0$$

In the second one, x is used as a channel.

Syntax: parallel composition

$$P \mid Q$$

Some *structural congruence* rules:

$$P \mid Q \equiv Q \mid P$$
$$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

Commutativity $+$ associativity

$$P \mid 0 \equiv P$$

Stopped processes are immaterial

Summing up: networks as “sets of processes”

Syntax: parallel composition

$$P \mid Q$$

Processes P and Q can

- evolve independently

  $$\text{if } P \xrightarrow{\alpha} P' \text{ then } P \mid Q \xrightarrow{\alpha} P' \mid Q$$

  non-deterministic: “who goes first?”

- interact (communicate)

  $$\overline{a}\langle x \rangle.P \mid a(y).Q \longrightarrow P \mid Q\{x \mapsto y\}$$

  The channel a must be the same.
  The variable y is bound to the message x,
  through a substitution on Q.

Example

$$\overline{a}\langle a\rangle.0 \mid a(y).\overline{c}\langle y\rangle.0$$

$$\longrightarrow$$

$$0 \mid \overline{c}\langle a\rangle.0$$

$$\xrightarrow{\overline{c}\langle a\rangle}$$

$$0 \mid 0 \equiv 0$$

Example

$$\overline{a}\langle a\rangle.0 \;\mid\; a(y).\overline{c}\langle y\rangle.0$$

$$\rightarrow$$

$$0 \;\mid\; \overline{c}\langle a\rangle.0$$

$$\xrightarrow{\overline{c}\langle a\rangle}$$

$$0 \;\mid\; 0 \equiv 0$$

But also:

$$\overline{a}\langle a\rangle.0 \;\mid\; a(y).\overline{c}\langle y\rangle.0$$

$$\xrightarrow{\overline{a}\langle a\rangle}$$

$$0 \;\mid\; a(y).\overline{c}\langle y\rangle.0 \quad \text{(stuck)}$$

Syntax: restriction

$$(\nu x)P$$

creates a new name and binds it to x in P.

new name $\sim$ new channel, or new tag, or new key

A low-level semantics:

$$(\nu x)P \rightarrow P\{x \mapsto c\}$$

where c is a globally fresh name

Example: restriction

$$(\nu z)(\overline{z}\langle a\rangle.0 \;\mid\; z(y).\overline{y}\langle z\rangle.0)$$

$$\longrightarrow$$

$$\overline{c}\langle a\rangle.0 \;\mid\; c(y).\overline{y}\langle c\rangle.0$$

$$\longrightarrow$$

$$0 \;\mid\; \overline{a}\langle c\rangle.0$$

$$\xrightarrow{\overline{a}\langle c\rangle}$$

$$0 \;\mid\; 0 \equiv 0$$

Example: restriction

$$(\nu z)(\overline{z}\langle a\rangle.0 \mid z(y).\overline{y}\langle z\rangle.0) \mid Q$$

$$\longrightarrow$$

$$\overline{c}\langle a\rangle.0 \mid c(y).\overline{y}\langle c\rangle.0 \mid Q$$

$$\longrightarrow$$

$$0 \mid \overline{a}\langle c\rangle.0 \mid Q$$

$$\xrightarrow{\overline{a}\langle c\rangle}$$

$$0 \mid 0 \mid Q \equiv Q$$

Note that there is no way Q can communicate on z.

Syntax: match

$$[x = y]P$$

behaves as P if $x = y$.
Otherwise it is stuck.

Semantics:

$$[x = x]P \rightarrow P$$

Summary of the first part

What is the computational power of the calculus, as seen so far?

Is it possible for a process to diverge?

$$P \longrightarrow \longrightarrow \longrightarrow \cdots \text{ (infinite trace)}$$

Summary of the first part

What is the computational power of the calculus, as seen so far?

Is it possible for a process to diverge?

$$P \longrightarrow \longrightarrow \longrightarrow \cdots \text{ (infinite trace)}$$

No. Each communication makes a process syntactically smaller.

Is the calculus *Turing-complete*?

Summary of the first part

What is the computational power of the calculus, as seen so far?

Is it possible for a process to diverge?

$$P \longrightarrow \longrightarrow \longrightarrow \cdots \text{ (infinite trace)}$$

No. Each communication makes a process syntactically smaller.

Is the calculus *Turing-complete*? No.

We must introduce some kind of recursion in the calculus.

Syntax: replication

A useful recursive construct

$$!P$$

Intuition:

$$!P \equiv P \mid P \mid P \mid \cdots \text{ (ad infinitum)}$$

Formally,

$$!P \equiv P \mid !P$$

Useful to model *servers*

$$!\, a(x).Q$$

Roughly mimicks the $\texttt{accept}/\texttt{fork}$ loop.

Example: replication

$$P = !\,(\nu z)(\overline{a}\langle z\rangle.0)$$

$$\equiv\longrightarrow$$

$$\overline{a}\langle c_1\rangle.0 \;\mid\; P$$

$$\equiv\longrightarrow$$

$$\overline{a}\langle c_1\rangle.0 \;\mid\; \overline{a}\langle c_2\rangle.0 \;\mid\; P$$

$$\xrightarrow{\overline{a}\langle c_2\rangle}$$

$$\overline{a}\langle c_1\rangle.0 \;\mid\; 0 \;\mid\; P$$

P keeps generating fresh names, sending them over channel a

Example: replication vs. recursion

With some (slight) extensions, we can write a factorial function:

$$
\begin{aligned}
! \, fact(x, ret).\mathtt{if}\ x = 0 \\
\quad \mathtt{then}\ \overline{ret}\langle 1 \rangle.0 \\
\quad \mathtt{else}\ (\nu z)(\ \ & \overline{fact}\langle x - 1, z \rangle. \\
& z(y). \\
& \overline{ret}\langle x \times y \rangle.0 \quad)
\end{aligned}
$$

x is the argument, ret is the return channel. Note the fresh return channel z for the recursive call (invocation). Its result is y.

Restriction: alternative semantics

Local, compositional semantics

$$((\nu x).P)|Q \equiv (\nu x).(P|Q) \text{ if } x \notin \mathrm{free}(Q) \cdots$$

We need alpha-conversion (renaming of x)

Scope enlargement rules stop at replication.

$$!\,(\nu x)P \not\equiv (\nu x)!\,P$$

Scope extrusion (when output is performed).

Other classical definitions

$$P + Q$$

Non deterministic choice

$$P + Q \rightarrow P$$
$$P + Q \rightarrow Q$$

Who chooses?

External/Angelic vs. Internal/Demonic

Example: coffee machine

$$\mathsf{coin}.(\overline{\mathsf{tea}} + \overline{\mathsf{coffee}}) \qquad \text{angelic}$$
$$\mathsf{coin}.\overline{\mathsf{tea}} + \mathsf{coin}.\overline{\mathsf{coffee}} \qquad \text{demonic}$$

Security: the adversary chooses (worst case)

Other classical definitions

(Too) many equivalences between processes

- traces
  $a + a.b \equiv a.b$
- bisimulation
  $a + a.b \not\equiv a.b$
  $a.b + b.a \equiv a|b$
- observational equivalence
- barbed equivalence
- testing equivalence
- ...

A simple property: (un)reachability

Reachability, e.g. $\neg(P \longrightarrow^* \xrightarrow{\langle \mathsf{secret} \rangle})$

- secrecy
  no secret is disclosed
- authentication
  no end before begin
- forward secrecy
  no old secret is disclosed

A simple property: (un)reachability

Reachability, unlike equivalences

- defines strong attacks
- community consensus
- easier to check automatically

Many techniques/tools exist:

- model checking
- static analysis
  - type systems
  - control flow analysis (tomorrow)

Modelling Protocols in π calculus

Steps:

- define the term algebra
  often, the free algebra
- define the protocol participants
- allow for an unbound number of parallel sessions
  (use replaciation)
- define the adversary
- put everything in parallel

Modelling Protocols in π calculus

How to handle network scheduling?
(non determinism)

Modelling Protocols in π calculus

How to handle network scheduling?
(non determinism)

Security: the adversary chooses (worst case)

"err on the safe side"

Not different from the computational models

Dolev-Yao Formal Adversary

Message rerouting $\iff$ only one channel

$$\overline{a}\langle x\rangle.P \quad \text{vs} \quad \langle x\rangle.P$$
$$a(x).P \quad \text{vs} \quad (x).P$$

Private channels vs. global channel

Restriction (νz) still useful

- key generation
- nonce generation
- ...

Example

Wide mouthed frog example

We now show the actual specification for a tool (Rewrite).

Example

Dolev-Yao adversary.

```
 !.( in W . in Z .
      ! . out W . out Z .
      out enc(W, Z) . out dec(W, Z) . ()
   | new Nonce . out Nonce . ()
   )
```

Decription is in the term algebra, but it does not need to be. Alternative:

$$\text{decrypt } x \text{ as } \{y\}_k \text{ in } P_y$$

Example

```
! . new AS . new BS .                 # Unbounded sessions
  (                                   # The server S
    in X .
    out enc(dec(X,AS),BS) . ()
  |                                   # Participant A
    new Key .
    out enc(Key,AS) .
    out enc(msg,Key) . ()             # msg is the secret
  |                                   # Participant B
    in N . in Key1 .
    let Key2 = dec(Key1,BS) .
    let Mess2 = dec(N,Key2) .
    out hash(Mess2) . ()                    )
```

Modelling Protocols in π calculus

Reachability result

$$\neg(Proto \mid Adv \rightarrow^* \xrightarrow{\langle \mathsf{msg} \rangle})$$

Can be automatically checked

“Hard” to do by hand

- cost problems
- confidence problems

Open Discussion

How to model

- random nonces
- signatures
- public key crypto
- exclusive or
- secure sessions (e.g. SSL)
- “toss a coin”
- quantitative aspects (DoS: denial of service)
- zero-knowledge protocols

Tomorrow

- automatic security proofs
- techniques
- tools
- static analysis
- control flow analysis