A Calculus for Secure Mobility

Bruno Blanchet
CNRS, École Normale Supérieure
and Max-Planck-Institut für Informatik

Benjamin Aziz
Computer Science Department,
University College Cork

December 2003

Introduction

Our goal is to combine:

- cryptography
- mobility: processes/programs being sent on the network
- locations: protections by firewalls for instance

to represent all security aspects of network applications, such as Java applets.

Overview

Design choices

The calculus (syntax, part of the semantics)

Examples

Observational equivalence

Representing cryptography

Many different variants of the spi-calculus [Abadi, Gordon, I&C'99]
= pi calculus + cryptographic primitives.

We focus on the applied pi calculus [Abadi, Fournet, POPL'01]
which is not limited to a small set of built-in primitives:
cryptographic primitives can be defined by equational theories.

Example: symmetric encryption

$$\mathbf{decrypt}(\mathbf{encrypt}(x, y), y) = x$$

optionally, we can add $\mathbf{decrypt}(M, N) = \mathbf{wrong}$ when $M \neq \mathbf{encrypt}(L, N)$
for all L.

Representing cryptography

A more complex example: Diffie-Hellman key agreements use modular exponentiation and can be modeled using the equational theory of [Meadows, Narendran, WITS'02]

$$x * (y * z) = (x * y) * z$$
$$x * y = y * x$$
$$x * 1 = x$$
$$x * x^{-1} = 1$$
$$exp(x, 1) = x$$
$$exp(exp(x, y), z) = exp(x, y * z)$$

We can represent data structures, public-key encryption, signatures, hash functions, macs, Diffie-Hellman key agreements, XOR, ...

Representing mobility

- Processes should be sent and received explicitly, as in current real networks.

  Excludes the ambient calculus [Cardelli, Gordon, TCS'00] and variants, in which processes move by themselves.

  Closer to the seal calculus [Vitek, Castagna, IPL'99]

- Processes should be manipulated by cryptographic functions, for example, to encrypt them before sending them on the network.

$\Rightarrow$ processes as data, higher-order calculus

Processes as data

We use special functions:
pack converts a process into a piece of data
exec converts a piece of data into a process

- A packed process is $\mathbf{pack}((M_1, \ldots, M_n), \lambda x_1, \ldots, x_n.P)$ with $fn(P) = \emptyset$ and $fv(P) \subseteq \{x_1, \ldots, x_n\}$.
  – $(M_1, \ldots, M_n)$ represents the data of the packed process.
  – $\lambda x_1, \ldots, x_n.P$ represents its code.

- When the packed process is executed, it runs $P\{M_1/x_1, \ldots, M_n/x_n\}$:

  $$\mathbf{exec}(\mathbf{pack}((M_1, \ldots, M_n), \lambda x_1, \ldots, x_n.P)) \rightarrow P\{M_1/x_1, \ldots, M_n/x_n\}$$

- When the packed process is sent in the clear on the network, it reveals its contents, in particular its data $(M_1, \ldots, M_n)$.

Processes as data (continued)

The definition of packed processes ensures separation of code and data, which is important for security.

The following processes have the same effet when they are executed, but reveal different pieces of data:

$$\mathbf{pack}((s, k, a), \lambda x, y, z.\overline{z}^{\uparrow}(\mathbf{encrypt}(x, y)))$$

reveals s, k, a when it is sent in clear over the network, whereas

$$\mathbf{pack}((\mathbf{encrypt}(s, k), a), \lambda x, z.\overline{z}^{\uparrow}(x))$$

only reveals $\mathbf{encrypt}(s, k)$ and a, so it does not reveal s when the adversary does not have k.

Processes as data (continued)

- **getdata** allows the adversary to obtain data from a packed process:

$$\mathbf{getdata}(\mathbf{pack}((M_1, \ldots, M_n), \lambda\vec{x}.P)) = (M_1, \ldots, M_n)$$

- The adversary can also compare the code of processes (by building a packed process with the same data and its own code).

- We can easily add a function to obtain the code of a process as a term, which would make it possible to modify received processes.

Syntax

Terms represent messages (data).
Processes represent code.

$M, N ::=$	terms
x, y, z	variable
a, b, c, k	name
$f(M_1, \ldots, M_n)$	function application
$\mathbf{pack}(\vec{M}, \lambda\vec{x}.P)$	packed process
(with $fn(P) = \emptyset, fv(P) \subseteq \vec{x}$)	

Syntax

$\eta ::=$	communication target
M	down
$\uparrow$	up
$\star$	local
$P, Q ::=$	processes
$0 \quad P \mid Q \quad !P \quad (\nu a)P$	
$M^{\eta}(\lambda x).P$	input
$\overline{M}^{\eta}(M').P$	output
$M[P]$	location
$\mathbf{exec}(M)$	execute packed process
$if \ M = N \ then \ P \ else \ Q$	conditional

Key part of the semantics: Communications with locations

Semantics: a structural equivalence relation $\equiv$
and a reduction relation $\rightarrow$.

Locations $M[P]$ communicate internally and with locations immediately inside or outside them:

- local: $a^\star(\lambda y).P' \mid \overline{a}^\star(M).Q' \rightarrow P'\{M/y\} \mid Q'$
- down: $c[a^\uparrow(\lambda y).P' \mid R] \mid \overline{a}^c(M).Q' \rightarrow c[P'\{M/y\} \mid R] \mid Q'$
- up: $a^c(\lambda y).P' \mid c[\overline{a}^\uparrow(M).Q' \mid R] \rightarrow P'\{M/y\} \mid c[Q' \mid R]$

Similar to the boxed ambients [Bugliesi, Castagna, Crafa, TACS'01]

Example: firewall

Locations can be used to model firewalls: when the adversary does not have c, it cannot send messages through $c[\ldots]$.

- A perfect firewall is defined by

  $$(\nu c)c[P]$$

  Nothing goes through this firewall.

- A firewall that leaves some channels open can be represented by

  $$(\nu c)(c[P] \mid !a^{\star}(\lambda x).\overline{a}^{c}(x) \mid !b^{c}(\lambda x).\overline{b}^{\star}(x))$$

  The process forwards messages on channel a from outside to inside $c[\ldots]$ and on channel b from inside to outside $c[\ldots]$.

Example: applet

Processes sent as messages can be used to model applets (such as Java applets).

Java applets are often signed for security:

$$\begin{array}{ll} C \to S: & \text{“Send me your applet”} \\ S \to C: & \{\mathbf{pack}(M, \lambda y.P)\}_{sk_S} \end{array}$$

$(\nu sk_S)let\ pk_S = \mathbf{pk}(sk_S)\ in\ \overline{c}^{\star}(pk_S).$

$((C) \quad \overline{c}^{\star}(\mathsf{SendApplet}).c^{\star}(\lambda x).\mathbf{exec}(\mathbf{checksign}(x, pk_S))$

$|\,(S) \quad c^{\star}(\lambda x).if\ x = \mathsf{SendApplet}\ then\ \overline{c}^{\star}(\mathbf{sign}(\mathbf{pack}(M, \lambda y.P), sk_S)))$

Example: applet with a sandbox

A well-placed sandbox can protect against malicious applets:

$$c^\star(\lambda x).(\nu sb)(sb[\mathbf{exec}(x)] \mid P) \mid Q$$

where P models the Java standard library. Its communications with the applet model method calls and returns.

The applet can communicate to the external world only through P. So P can restrict these communications for security.

The process Q is protected from the applet.
(It is outside of the sandbox.)

Example: certified email system (very simplified!)
Abadi, Glew, Horne, Pinkas, WWW2002

The receiver R should obtain the message if and only if the sender S has a certificate that R has read the message.

$S \to R$:	$\mathbf{encrypt}(m, k), \mathbf{pencrypt}(k, pk_{TTP})$
$R \to TTP$:	$\mathbf{pencrypt}(k, pk_{TTP})$ on a secure channel (SSL)
$TTP \to R$:	k on the secure channel (SSL)
$TTP \to S$:	certificate that R has received m

The (full) protocol is correct. In an implementation,

$S \to R$: <APPLET "http://TTP/applet.jar">, . . .

The applet connects with TTP and displays the message.
Malicious applet $\Rightarrow$ broken protocol! (The applet never gives the message to R, while S receives his certificate.)

Observational equivalence

P and Q are observationally equivalent,
$P \approx Q$, when the adversary cannot distinguish P from Q.

$\approx$ is the largest symmetric relation $\mathcal{R}$ between closed processes such that $P \mathcal{R} Q$ implies:

- if P emits on channel a, then Q emits on a;
- if $P \rightarrow^* P'$ then there exists Q' such that $Q \rightarrow^* Q'$ and $P' \mathcal{R} Q'$;
- $C[P] \mathcal{R} C[Q]$ for all closing evaluation contexts $C[]$.
  $(C ::= [] \quad (\nu a)C \quad P \mid C \quad C \mid P \quad M[C])$

Secrecy (non-interference)

Observational equivalence can be used to define various security properties, for example secrecy:

Let P be a process with free variables $x_1, \ldots, x_n$.

P preserves the secrecy of $x_1, \ldots, x_n$ if and only if
for all σ and σ' substitutions of domain $x_1, \ldots, x_n$,

$$P\sigma \approx P\sigma'$$

The adversary can observe the code of processes

The adversary can observe the code of packed processes, by testing equality with processes he builds

$$c^{\eta}(\lambda x).\mathit{if}\ \ x = \mathbf{pack}(\mathbf{getdata}(x), \lambda\vec{y}.P)\ \ \mathit{then}\ \overline{a}^{\star}(x)\ \ \mathit{else}\ 0$$

tests equality between the code of x and P.

Important consequence: for the equivalence, the language behaves like a first-order language.

In contrast, in most higher-order languages, the adversary can observe processes only by running them.

Observational equivalence as labeled bisimilarity

Observational equivalence proofs are difficult, because we have to prove properties for all contexts.

Labeled bisimilarity makes the proofs much easier by avoiding this quantification.

Agents

Agents represent processes as well as the current knowledge of the adversary.

$$A = (\nu \vec{a})(\{M_1/x_1, \ldots, M_n/x_n\}, P)$$

The adversary has the terms $M_1, \ldots, M_n$, but sometimes cannot inspect the whole structure of these terms, because it does not a priori have the names $\vec{a}$.

$$(\nu a, b, k)(\{\mathbf{encrypt}((a, b), k)/x\}, P)$$

The adversary cannot see that x is the encryption of a pair.

The structural equivalence and reduction of agents extend naturally those of processes.

Static equivalence $A \approx_{\mathsf{s}} B$

A and B give statically indistinguishable knowledge to the adversary.

Intuitively, $A \approx_{\mathsf{s}} B$ when

- equality tests cannot differentiate A from B:
  $M = N$ with the values of the variables given by $A \Leftrightarrow$
  $M = N$ with the values of the variables given by B;

- M is equal to a name with the values of the variables given by $A \Leftrightarrow$ it is with the values of the variables given by B.

  The adversary can detect whether a term is a name by using it as a channel.

Labeled reduction

Labeled reduction represents the reduction of a process in interaction with a context.

$$a^{\eta}(\lambda x).P \xrightarrow{a^{\eta}(\lambda M)} P\{M/x\} \qquad (M \text{ and } \eta \text{ closed}) \qquad \text{(In)}$$

$$\overline{a}^{\eta}(M).P \xrightarrow{\overline{a}^{\eta}(x)} \{M/x\}, P \qquad (\eta \text{ closed}) \qquad \text{(Out)}$$

$$\frac{P \xrightarrow{\overline{a}^{\uparrow}(x)} A}{c[P] \xrightarrow{c[\overline{a}^{\uparrow}(x)]} c[] \circ A} \qquad \text{(Loc Out)}$$

with $c[] \circ (\nu \vec{a})(\sigma, P) = (\nu \vec{a})(\sigma, c[P])$ if $c \notin \vec{a}$

Similar rule for input inside a location, etc.

Labeled bisimilarity

Labeled bisimilarity ($\approx_{\mathsf{l}}$) is the largest symmetric relation $\mathcal{R}$ on closed agents such that $A \,\mathcal{R}\, B$ implies:

- $A \approx_{\mathsf{s}} B$;
- if $A \to A'$ then there exists B' such that $B \to^* B'$ and $A' \,\mathcal{R}\, B'$;
- if $A \xrightarrow{\alpha} A'$, then there exists B' such that $B \to^* \xrightarrow{\alpha} \to^* B'$ and $A' \,\mathcal{R}\, B'$.

Comparison between observational equivalence and labeled bisimilarity

Theorem 1 *Labeled bisimilarity equals observational equivalence:* $P \approx_{\mathsf{l}} Q$ *if and only if* $P \approx Q$.

This result can be used to prove security properties of the previous examples.

Conclusion

The calculus can:

- represent generic cryptographic primitives
- represent communication of code over a network,
  and manipulation of code by cryptographic operations.
- represent location-based security (firewalls, sandboxes, . . .)