Analyzing Security Protocols with Secrecy Types and Logic Programs

Martín Abadi
University of California, Santa Cruz

Bruno Blanchet
Ecole Normale Supérieure
Max-Planck Institut für Informatik

January 2002

Introduction

The design of cryptographic protocols is difficult and error-prone. Their verification is therefore particularly important.

Programming language methods
(typing, abstract interpretation, logic programming, ...)
can handle infinite-state protocols, often with little effort.
$\Rightarrow$ interesting alternative to model-checking and theorem proving.

We extend two language-based lines of research and clarify their relations:

- one based on a typed process calculus

- the other on (untyped) logic programs

Starting point

Two superficially quite different approaches:

– Types:
Process calculus with communication and cryptographic primitives (such as the spi-calculus).
Type system to prove properties of these processes.

– Logic programming:
Representation of protocols by Horn clauses.
Inference algorithm to prove properties of protocols.

We focus on secrecy properties.

Similarities between both systems

Both systems make common approximations:

- Tests $if\ M = N\ then\ P\ else\ Q$:
  - "then" branch typed only if M and N have compatible types
    "else" branch always typed
  - "then" branch considered only if M and N are unifiable
    "else" branch always considered

- Name creation (e.g. creation of fresh keys):
  - the type of the fresh name depends on the type environment
  - a new name is a function of previous inputs

Scope of both systems

Till now, the two systems differ in scope:

- Most previous type systems are specific to some cryptographic and communication primitives.

- The logic programming tools are more flexible and handle more general cryptographic primitives.

Results

- A general and flexible type system to prove secrecy properties of protocols, that handles fairly arbitrary cryptographic primitives.

- A translation from the process calculus to Horn clauses

- The equivalence of the two systems:
  – The logic programming protocol checker corresponds to a particular instance of the type system (correctness)
  – It can prove all secrecy properties provable in any instance of the general type system (relative completeness)

Process calculus

Extension of the pi calculus (roughly like the applied pi calculus).

$M, N ::=$	terms
$\quad x, y, z$	variable
$\quad a, b, c, k, s$	name
$\quad f(M_1, \ldots, M_n)$	constructor application
$P, Q ::=$	processes
$\quad \overline{M}\langle N \rangle.P$	output
$\quad M(x).P$	input
$\quad \mathit{let}\ x = g(M_1, \ldots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q$	destructor application
$\quad \mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$	conditional
$\quad 0 \quad P \mid Q \quad !P \quad (\nu a)P$	

Constructors and destructors

Two kinds of operations:

- Constructors f are used to build terms
  $f(M_1, \ldots, M_n)$

- Destructors g manipulate terms
  $\mathit{let}\ x = g(M_1, \ldots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q$
  Destructors are defined by equations $g(M_1, \ldots, M_n) = M$.

Examples of constructors and destructors

- Constructor: Shared-key encryption $s\_encrypt(M, N)$.
  Destructor: Decryption $s\_decrypt(M', N)$

$$s\_decrypt(s\_encrypt(M, N), N) = M.$$

- Constructors: Public-key encryption $p\_encrypt(M, N)$.
  Public key generation $pk(N)$.
  Destructor: Decryption $p\_decrypt(M', N)$

$$p\_decrypt(p\_encrypt(M, pk(N)), N) = M.$$

Examples of constructors and destructors (continued)

- Constructor: Signature $sign(M, N)$.
  Destructors: Signature checking $checksign(M', N')$

$$checksign(sign(M, N), pk(N)) = M.$$

Message extraction $getmessage(M')$

$$getmessage(sign(M, N)) = M.$$

- Constructor: One-way hash function $H(M)$.

Constructors and destructors can also be used to represent data structures.

Definition of secrecy

S finite set of names (initial knowledge of the adversary).

A S-adversary is any process Q such that $fn(Q) \subseteq S$ and Q has no free variable.

P preserves the secrecy of M from S

$\Leftrightarrow$

$P \mid Q$ does not output M on c for any S-adversary Q and any $c \in S$.

Example

- $\overline{a}\langle pk(k)\rangle.\overline{a}\langle p\_encrypt(s, pk(k))\rangle$ preserves the secrecy of s from $\{a\}$.

  No $\{a\}$-adversary can output s on a.

- $\overline{a}\langle k\rangle.\overline{a}\langle p\_encrypt(s, pk(k))\rangle$ does not preserve the secrecy of s from $\{a\}$.

  The adversary $a(x).a(y).let\ z = p\_decrypt(y, x)\ in\ \overline{a}\langle z\rangle$ outputs s on a.

Our generic type system

- The set of types depends on each instance of the type system.

- The type system is parametrized by a set of functions that manipulate types.

  $conveys(T)$ = set of types of data that can be conveyed on a channel of type T

  $O_f(T_1, \ldots, T_n)$ = type of the result of applying the constructor f to data of types $T_1, \ldots, T_n$.

  $O_g(T_1, \ldots, T_n)$ = set of possible types of the result of applying the destructor g to data of types $T_1, \ldots, T_n$.

The type rules: Constructors

$O_f(T_1, \ldots, T_n)$ = type of the result of applying the constructor f to data of types $T_1, \ldots, T_n$.

$$\frac{\forall i \in \{1, \ldots, n\}, E \vdash M_i : T_i \qquad O_f(T_1, \ldots, T_n) \text{ is defined}}{E \vdash f(M_1, \ldots, M_n) : O_f(T_1, \ldots, T_n)}$$

The type rules: Destructors

$O_g(T_1, \ldots, T_n)$ = set of possible types of the result of applying the destructor g to data of types $T_1, \ldots, T_n$.

$$\frac{\forall i \in \{1, \ldots, n\}, E \vdash M_i : T_i \qquad \forall T \in O_g(T_1, \ldots, T_n), (E, x : T) \vdash P \qquad E \vdash Q}{E \vdash let\ x = g(M_1, \ldots, M_n)\ in\ P\ else\ Q}$$

Compatibility between g and O_g:

If $g(M_1, \ldots, M_n) = M$ and for all $i \in \{1, \ldots, n\}, E \vdash M_i : T_i$, then there exists $T \in O_g(T_1, \ldots, T_n)$ such that $E \vdash M : T$.

Typing the adversary

T_{Public} = set of types of data that the adversary can have.

- The adversary can send and receive any public message on any public channel.
  If $T \in T_{\text{Public}}$, then $conveys(T) = T_{\text{Public}}$.

- The adversary can apply constructors.
  If for all $i \in \{1, \dots, n\}$, $T_i \in T_{\text{Public}}$,
  then $O_f(T_1, \dots, T_n)$ is defined and $O_f(T_1, \dots, T_n) \in T_{\text{Public}}$.

- The adversary can apply destructors.
  If for all $i \in \{1, \dots, n\}$, $T_i \in T_{\text{Public}}$ and $T \in O_g(T_1, \dots, T_n)$,
  then $T \in T_{\text{Public}}$.

Secrecy theorem

"Well-typed processes preserve their secrets"

Let S be the set of public names in environment E (the set of names whose type is in T_{Public}).

Let P be a process without free variable.

If P is well-typed in environment E,
and s is a secret name in environment E
(a name whose type is not in T_{Public}),
then P preserves the secrecy of s from S.

Example: A simple instance of this type system

T ::=	types
Public	public data
Secret	secret data
$T_1 \times \ldots \times T_n$	tuple
$\mathsf{EK}^{\mathsf{Public}}[T]$	public encryption key
$\mathsf{DK}^{\mathsf{Public}}[T]$	decryption key (with public encryption key)
...	

Subtyping: $\mathsf{EK}^{\mathsf{Public}}[T] \leq$ Public, $\mathsf{DK}^{\mathsf{Public}}[T] \leq$ Secret, ...

Asymmetric encryption with public encryption key

$$
\begin{tikzcd}
M : T', T' \leq T \arrow[dr] & & M : T_P, T_P \leq \mathsf{Public} \arrow[dl, "p\_encrypt(M, K),\, K : \mathsf{EK}^{\mathsf{Public}}[T]"] \\
& M' : \mathsf{Public} \arrow[d, "p\_decrypt(M', K'),\, K' : \mathsf{DK}^{\mathsf{Public}}[T]"] & \\
& T' \text{ or } T_P &
\end{tikzcd}
$$

if $T' \leq T$ or $T' \leq \mathsf{Public}$, then $O_{p\_encrypt}(T', \mathsf{EK}^{\mathsf{Public}}[T]) = \mathsf{Public}$

$O_{p\_decrypt}(\mathsf{Public}, \mathsf{DK}^{\mathsf{Public}}[T]) = \{T' \mid T' \leq T\} \cup \{T_P \mid T_P \leq \mathsf{Public}\}$

Protocol checker

In the protocol checker, the actions of the protocol and of the attacker are represented by Horn clauses:

- Terms: $M ::= f(M_1, \ldots, M_n) \mid a[M_1, \ldots, M_n] \mid x$
  A name a is represented by a function $a[M_1, \ldots, M_n]$ of the inputs that precede its creation.

- Facts: $F ::= \text{attacker}(M) \mid \text{message}(M, M')$
  attacker(M): the attacker has the term M
  message(M, M'): the message M' is sent on the channel M

- Rules: $F_1 \wedge \ldots \wedge F_n \Rightarrow F$

Protocol checker

- Secrecy criterion:
  When attacker(M) is not derivable from the Horn clauses representing the protocol, the protocol preserves the secrecy of M.

- This checker can prove secrecy properties, or find attacks.

- It is flexible and efficient in practice.

The checker as an instance of the type system

We define a translation from our process calculus to Horn clauses. The checker can then be seen as an instance of the general type system

- Types: $T ::= f(T_1, \ldots, T_n) \mid a[T_1, \ldots, T_n]$
- $conveys(T) = \{T' \mid \text{message}(T, T') \text{ is derivable from the Horn clauses representing the process in question}\}$
- $O_f(T_1, \ldots, T_n) = f(T_1, \ldots, T_n)$
- $O_g =$ natural extension of g to types.
- $T_{\text{Public}} = \{T \mid \text{attacker}(T) \text{ is derivable from the Horn clauses representing the process in question}\}$

Equivalence between the checker and the type system

Theorem 1
S finite set of names, $s \in \mathit{fn}(P_0)$.
If attacker(s) cannot be derived from the Horn clauses representing P_0, the attacker, and its initial knowledge S,
then P_0 preserves the secrecy of s from S.

Theorem 2
A secrecy property can be inferred from Theorem 1

$\Leftrightarrow$

it can be inferred from the secrecy theorem for the type system.

Conclusion

Contributions:

- A general type system
  for proving secrecy properties of protocols
  with a generic treatment of many cryptographic primitives.

- A tight relation between two superficially quite different approaches to protocol analysis:
  - a type system
  - a logic programming tool.