SE547: Lecture 4

Overview

Cryptographic protocols

Spi calculus

Proverif

Secrecy

Authenticity

Cryptographic protocols

The pi-calculus contains enough features (processes, communication) to model communications protocols.

What features need to be added to model cryptographic protocols?

What properties could we check of these cryptographic protocols?

Spi calculus

History:

Pi-calculus (Milner)

+ Dolev-Yao model of cryptographic protocols

= Spi-calculus (Abadi and Gordon).

Other models (all late 1990s-early 2000s):

Strand Spaces (Guttman).

Communicating Sequential Processes for crypto (Roscoe, Lowe, Schneider...)

Multiset Rewriting (Scedrov, Cervesato, Mitchell...)

...

Spi calculus

Take the pi-calculus:

P, Q, R ::=
- $\mathbf{0}$
- out $x\ y; P$
- in $x\ (y); P$
- $P \mid Q$
- $!P$
- new $(x); P$

Spi calculus

Add messages as primitives:

P, Q, R ::=
- **0**
- out $M\ N; P$
- in $M\ (y); P$
- $P \mid Q$
- $!P$
- new $(x); P$
- ... other stuff will go here ...

L, M, N ::=
- x
- ... other stuff will go here ...

Spi calculus

Add lists as primitives:

P, Q, R ::=
...
split L is (x, y); P

L, M, N ::=
...
()
(M,N)

Example program:

!in c (x);
split x is $(head,tail)$;
out d $(head)$; out c $(tail)$; **0**

Spi calculus

Dynamic semantics of lists:

split (M, N) is (x, y); $P \rightarrow P[M/x, N/y]$

What is the dynamic semantics of:

Example | out c (*fred*, (*wilma*, ())); **0**

where:

Example =
!in c (x);
split x is (*head*,*tail*);
out d (*head*); out c (*tail*); **0**

Spi calculus

Add message tags as primitives (just 2 tags are good enough):

P, Q, R ::=
  ...
  case L is inl(x) P is inr(y) Q

L, M, N ::=
  ...
  inl (M)
  inr (M)

Example program:

!in c (x);
case x is inl(y)
  out d (y); **0**
is inr(z)
  out e (z); **0**

Spi calculus

Dynamic semantics of message tags:

$$\text{case inl}(M) \text{ is inl}(x)\ P \text{ is inr}(y)\ Q \rightarrow P[M/x]$$
$$\text{case inr}(N) \text{ is inl}(x)\ P \text{ is inr}(y)\ Q \rightarrow Q[N/y]$$

What is the dynamic semantics of:

Example | out c (inl(*fred*)); out c (inr(*wilma*)); **0**

where:

Example =
  !in c (x);
  case x is inl(y)
    out d (y); **0**
  is inr(z)
    out e (z); **0**

Spi calculus

Add symmetric-key cryptography:

P, Q, R ::=
  ...
  decrypt M is $\{ x \}_N$; P

L, M, N ::=
  ...
  $\{ M \}_N$

Example program:

!in c (x);
decrypt x is $\{ y \}_{key}$;
out d (y); **0**

Spi calculus

Dynamic semantics of crypto:

$$\text{decrypt } \{\ M\ \}_N \text{ is } \{\ x\ \}_N; P \rightarrow P[M/x]$$

What is the dynamic semantics of:

$$\text{Example} \mid \text{out } c\ \{\ msg\ \}_{key}; \mathbf{0}$$

where:

Example =
!in c (x);
decrypt x is $\{\ y\ \}_{key}$;
out d (y); **0**

Spi calculus

Add nonces:

P, Q, R ::=
...
check M is N; P

Example program:

!in c (x);
check x is *nonce*;
out d (x); **0**

Spi calculus

Dynamic semantics of nonces:

check N is N; $P \rightarrow P$

What is the dynamic semantics of:

Example | out c ($nonce$); out c ($other$); **0**

where:

Example =
!in c (x);
check x is $nonce$;
out d (x); **0**

Spi calculus

Derived forms:

- Pattern matching.
- Lists of any length (not just zero or two).
- Any number of tags (not just two).

Example:

new (*nonce*);
out *net* (*nonce*);
in *net* { *msg*, *nonce′* }$_{key}$;
check *nonce* is *nonce′*; **0**

Spi calculus

Andrew Secure RPC:

(1) $A \rightarrow B : A, \{\ \text{msg1}(Na)\ \}Kab$
(2) $B \rightarrow A : \{\ \text{msg2}(Na, Nb)\ \}Kab$
(3) $A \rightarrow B : \{\ \text{msg3}(Nb)\ \}Kab$
(4) $B \rightarrow A : \{\ \text{msg4}(SK, SN)\ \}Kab$

What does this look like in spi?

Proverif

Proverif (Blanchet) is a spi-calculus based protocol verification tool.

(It's written in O'Caml, so you need to get ocaml from www.ocaml.org if you want to recompile proverif.)

To run proverif:

analyzer -in pi *filename*

for example:

analyzer -in pi pi-simple-secr

produces output:

RESULT goal unreachable: attacker:s[]

Proverif is trying to find an attack, so this is good news for the protocol!

Proverif

Proverif syntax is like the spi-calculus, but includes constant declarations, for example:

```
fun hostA/0.
private fun kAS/0.
fun hostB/0.
private fun kBS/0.
fun hostI/0.
fun kIS/0.
```

and function declarations, for example:

```
private reduc
   key(hostA) = kAS;
   key(hostB) = kBS;
   key(hostI) = kIS.
```

Proverif

The syntax is also slightly different, for example:

new s; out(net, sencrypt(s,kAB)); 0.
in(net, m1); let s = sdecrypt(m1, kAB) in 0.

rather than:

new (s); out *net* $\{s\}_{kAB}$; **0**
in *net* (*m1*); decrypt *m1* is $\{s\}_{kAB}$; **0**

Proverif

Some examples of secrecy:

Niave cryptography.

Wide Mouth Frog.

Andrew RPC.

Some examples of failed secrecy:

Publish a secret to anyone.

Example from homework 1.

Secrecy

A protocol P violates the secrecy of M if:

$P \rightarrow^* Q \mid \text{out } net\ M; \mathbf{0}$

A protocol is *safe* for secrecy of M otherwise.

A protocol is *robustly safe* for secrecy of M if:

for any O where $M \notin O$, $P \mid O$ is safe for secrecy of M

Secrecy

Examples:

1. out *net M*; **0**
2. new (*kAB*); out *net* { *M* }$_{kAB}$; **0**
3. new (*kAB*); out *net* { *M* }$_{kAB}$; out *net kAB*; **0**

Which ones are safe for secrecy of *M*?

Which ones are robustly safe for secrecy of *M*?

Secrecy

In proverif:

query s.

checks if the protocol is robustly safe for secrecy of s.

1. RESULT goal unreachable: attacker:s[] *the protocol robustly preserves secrecy* pi-simple-secr.
2. RESULT goal reachable: attacker:s[] *there is an attacker which violates secrecy* pi-stupid-secr.
3. ... or the tool may run forever ... pi-flawed-secr.

Authenticity

We want to verify:

> A generates M for B
> (1) $A \rightarrow B$: $\{ M \} Kab$
> B believes A generated M for B

we need to turn this into a formal statement!

Authenticity

One formulation of authenticity is as *correspondences*. Add session markers:

> A begins generated (A, B, M)
> (1) $A \rightarrow B$: { M }Kab
> B ends generated (A, B, M)

Now check correspondence:

> For any session marker L, the number of end Ls is less than or equal to the number of begin Ls.

In the presence of an attacker, does the above protocol satisfy correspondence?

Authenticity

Extending spi for authenticity:

P, Q, R ::=
...
begin L; P
end L; P

Example processes:

Sender =
new (m);
begin generated (A,B,m);
out c { m }$_{key}$; **0**
Receiver =
!in c (x);
decrypt x is { y }$_{key}$;
end generated (A,B,m); **0**

Authenticity

Record a *trace* of a process, generated by:

new (x); P -gen(x)-> P
begin (L); P -begin(L)-> P
end (L); P -end(L)-> P

For example, what are the traces of Sender | Receiver:

Sender =
  new (m);
  begin generated (A,B,m);
  out c { m }$_{key}$; **0**
Receiver =
  !in c (x);
  decrypt x is { y }$_{key}$;
  end generated (A,B,m); **0**

Authenticity

A process P is *safe* for authenticity if:

> for any trace of P, the number of end Ls is less than or equal to the number of begin Ls.

A process P is *robustly safe* for authenticity if:

> for any O, $P \mid O$ is safe for authenticity

Is Sender | Receiver safe? Robustly safe?

> Sender =
> new (m);
> begin generated (A,B,m);
> out c { m }$_{key}$; **0**
> Receiver =
> !in c (x);
> decrypt x is { y }$_{key}$;
> end generated (A,B,m); **0**

Authenticity

In proverif:

authquery generated/3.

checks if the protocol is robustly safe for authenticity (for session marker 'generated').

For example, pi-simple-auth.

Proverif also supports *many-to-one* correspondences, which check the weaker property:

For any session marker L, any end Ls is preceded by a begin Ls.

Note that begin L; end L; end L; **0** satisfies this weaker property.

Authenticity

Other failed examples of authenticity:

Wide Mouth Frog.

Andrew RPC.

Publish a secret to anyone.

Example from homework 1.

These can be fixed, for example:

Wide Mouth Frog.

Next week

Homework sheet 4.

Types for cryptographic protocols.