Proverif

- secrecy
- correspondence
- observational equivalence(indistinguishability)

- bound names: in a process
- free names: known to all processes
- variables

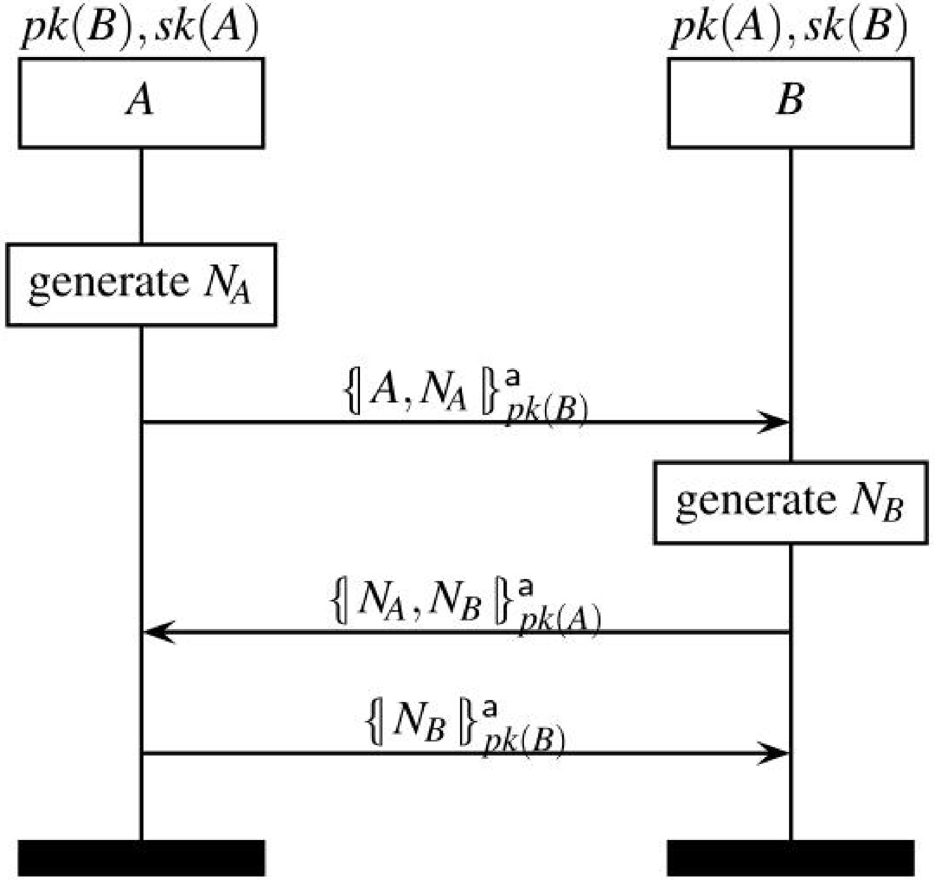
Needham-Shroeder public key protocol

$$
\begin{array}{llcllll}
(1) & A & \rightarrow & S & : & \mathrm{(A,\ B)} \\
(2) & S & \rightarrow & A & : & \mathrm{sign((B,\ pk(skB)),\ skS)} \\
(3) & A & \rightarrow & B & : & \mathrm{aenc((Na,\ A),\ pk(skB))} \\
(4) & B & \rightarrow & S & : & \mathrm{(B,\ A)} \\
(5) & S & \rightarrow & B & : & \mathrm{sign((A,\ pk(skA)),\ skS)} \\
(6) & B & \rightarrow & A & : & \mathrm{aenc((Na,\ Nb),\ pk(skA))} \\
(7) & A & \rightarrow & B & : & \mathrm{aenc(Nb,\ pk(skB))}
\end{array}
$$

Needham-Shroeder public key

- Simplified Version

$$
\begin{array}{lcll}
A & \rightarrow & B & : \quad \text{aenc((Na,pk(skA)), pk(skB))} \\
B & \rightarrow & A & : \quad \text{aenc((Na,Nb), pk(skA))} \\
A & \rightarrow & B & : \quad \text{aenc(Nb, pk(skB))}
\end{array}
$$



Tamarin

Symbolic analysis tool for systems in presence of a Dolev-Yao style network adversary

- **Modeling** protocol & adversary done using multiset rewriting
  - Specifies transition system; induces set of traces
- **Property** specification using fragment of first-order logic
  - Specifies “good” traces
- Tamarin tries to
  - provide proof that all system traces are good, or
  - construct a counterexample trace of the system (attack)

Modeling in Tamarin

- Basic ingredients:
  - **Terms** (think "messages")
  - **Facts** (think "sticky notes on the fridge") the predicate on terms
  - Special facts: **Fr(t)**, **In(t)**, **Out(t)**, **K(t)**

sent out the message t on the public channel

- State of system is a multiset of facts
  - **Initial state** is the empty multiset
  - **Rules** specify the transition rules ("moves")
- Rules are of the form:
  - `l --> r`
  - `l --[ a ]-> r`

Terms - Cryptographic Messages

Constants

We distinguish between two types of constants:

- *Public constants* model publicly known atomic messages such as agent identities and labels. We use the notation `'ident'` to denote public constants in Tamarin.
- *Fresh constants* model random values such as secret keys or random nonces. We use the notation `~'ident'` to denote fresh constants in Tamarin. Note that fresh *constants* are rarely used in protocol specifications. A fresh *variable* is almost always the right choice.

Terms - Cryptographic Messages

- Function Symbols
  - built-in function symbols
  - user-defined function symbols
- such like
  - pair, fst, snd
  - message theories: hashing, asymmetric-encryption, signing,diffie-hellman, bilinear-pairing, xor, and multiset
  - functions: f1/a1, ..., fn/an

Terms - Cryptographic Messages

`diffie-hellman:` This theory models Diffie-Hellman groups. It defines the function symbols `inv/1`, `1/0`, and the symbols `^` and `*`. We use `g ^ a` to denote exponentiation in the group and `*`, `inv` and `1` to model the (multiplicative) abelian group of exponents (the integers modulo the group order). The set of defined equations is:

```
(x^y)^z  = x^(y*z)
x^1      = x
x*y      = y*x
(x*y)*z  = x*(y*z)
x*1      = x
x*inv(x) = 1
```

Terms - Equational theories

- Equational theories can be used to model properties of functions
- equations: lhs1 = rhs1, ..., lhsn = rhsn

Facts - Model Specification

Facts are of the form `F(t1,...,tn)` for a fact symbol `F` and terms `ti`. They have a fixed arity (in this case `n`).

SPECIAL FACTS

there is a fresh rule that produces unique **Fr**(...) facts;
and there is a set of rules for adversary knowledge derivation,
which consume **Out**(...) facts and produce **In**(...) facts.

Two types of Facts

Linear Facts:they might appear in one state but not in the next.
Persistent Facts:some facts in our models will never be removed from the state once they are introduced

Rules - Model Specification

- We use multiset rewriting to specify the concurrent execution of the protocol and the adversary.
- A rewrite rule in Tamarin has a name and three parts, each of which is a sequence of facts: one for the rule's left-hand side, one labelling the transition (which we call 'action facts'), and one for the rule's right-hand side. For example:

```
rule MyRule1:
  [ ] --[ L('x') ]-> [ F('1','x'), F('2','y') ]

rule MyRule2:
  [ F(u,v) ] --[ M(u,v) ]-> [ H(u), G('3',h(v)) ]
```

Property Specification

- Trace Properties
  - A trace property is a set of traces.
  - We define a set of traces using first-order logic formulas over action facts and timepoints.
- Observational Equivalence

Property Specification

The syntax for specifying security properties is defined as follows:

- `All` for universal quantification, temporal variables are prefixed with #
- `Ex` for existential quantification, temporal variables are prefixed with #
- `==>` for implication
- `&` for conjunction
- `|` for disjunction
- `not` for negation
- `f @ i` for action constraints, the sort prefix for the temporal variable 'i' is optional
- `i < j` for temporal ordering, the sort prefix for the temporal variables 'i' and 'j' is optional
- `#i = #j` for an equality between temporal variables 'i' and 'j'
- `x = y` for an equality between message variables 'x' and 'y'
- `Pred(t1,..,tn)` as syntactic sugar for instantiating a predicate Pred for the terms `t1` to `tn`

Property Specification

To specify a property about a protocol to be verified, we use the keyword `lemma` followed by a name for the property and a guarded first-order formula. This expresses that the property must hold for all traces of the protocol. For instance, to express the property that the fresh value `~n` is distinct in all applications of the fictitious rule (or rather, if an action with the same fresh value appears twice, it actually is the same instance, identified by the timepoint), we write

Tamarin Syntax

a Tamarin input file consists of:

- Comments(C-like, single line: //, multiline: /* */)
- starts with *theory* followed by the *theory's name*
- the keyword *begin*
- *cryptographic primitives* declarations (the protocol process uses)
- *multiset rewriting rules* declarations (modeling the protocol)
- *properties* declarations (*lemmas* to be proven)
- *the keyword end*

Example

```
C -> S: aenc(k, pkS)
C <- S: h(k)
```

Example Begining and Cryptographic primitives

```
theory FirstExample
begin

builtins: hashing, asymmetric-encryption  //Tamarin's built-in functins
/*
  the built-ins give us:
 --unary function h, denoting a cryptographic hash function
 --a binary function aenc denoting the asymmetric encryption algorithm,
 --a binary function adec denoting the asymmetric decryption algorithm, and
 --a unary function pk denoting the public key corresponding to a private key.
*/
```

Example Multiset Rewriting Rules declarations

i. first, generate a fresh name ~ltk, which is the new private key(Long-term key)
ii. non-deterministically choose a public name A for the agent for whom we are generating the key-pair.
iii. Afterward, generate the fact !Ltk($A, ~ltk) (the exclamation mark ! denotes that the fact is persistent).which denotes the association between agent A and its private key ~ltk,
iv. generate the fact !Pk($A, pk(~ltk)), which associates agent A and its public key pk(~ltk).

```
// Registering a public key
rule Register_pk:
    [ Fr(~ltk) ]  //premise( if all the facts are present in current state)
  -->
    [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)) ]  //conclusion(add these facts to the state,delete the above)
```

In Tamarin, the sort of a variable is expressed using prefixes:

- ~x denotes x:fresh
- $x denotes x:pub
- #i denotes i:temporal
- m denotes m:msg

Example Multiset Rewriting Rules declarations

```
rule Get_pk:
    [ !Pk(A, pubkey) ]
  -->
    [ Out(pubkey) ]

rule Reveal_ltk:
    [ !Ltk(A, ltk) ]
  --[ LtkReveal(A) ]->
    [ Out(ltk) ]
```

modeling that the adversary can access any public key existing

[!Ltk] --[LtkReveal(A)]-> [Out] modeling dynamic compromise of long-term key

the action facts is " --[LtkReveal(A)]->", showing that the Long-term key of A has been compromised.

action facts shown on the traces, other facts shown in the states

Example Multiset Rewriting Rules declarations

```
C -> S: aenc(k, pkS)
C <- S: h(k)
```

C1 -> S1: aenc(k, pkS)
C2 <- S1: h(k)

```
// Start a new thread executing the client role, choosing the server
// non-deterministically.
rule Client_1:
    [ Fr(~k)          // choose fresh key
    , !Pk($S, pkS)    // lookup public-key of server
    ]
  -->
    [ Client_1( $S, ~k )    // Store server and key for next step of thread
    , Out( aenc(~k, pkS) )  // Send the encrypted session key to the server
    ]

rule Client_2:
    [ Client_1(S, k)   // Retrieve server and session key from previous step
    , In( h(k) )       // Receive hashed session key from network
    ]
  --[ SessKeyC( S, k ) ]-> // State that the session key 'k'
    []                     // was setup with server 'S' by client
```

Example Multiset Rewriting Rules declarations

C1 -> S1: aenc(k, pkS)
C2 <- S1: h(k)

```
// A server thread answering in one-step to a session-key setup request from
// some client.
rule Serv_1:
    [ !Ltk($S, ~ltkS)                             // lookup the private-key
    , In( request )                               // receive a request
    ]
  --[ AnswerRequest($S, adec(request, ~ltkS)) ]-> // Explanation below
    [ Out( h(adec(request, ~ltkS)) ) ]            // Return the hash of the
                                                  // decrypted request.

```

the action facts here is for allowing the formalization of the authentication property for the client , because security properties are defined over traces of the action facts of a protocol execution, using actions fact to record this knowledge.

In Tamarin, the sort of a variable is expressed using prefixes:

- ~x denotes x:fresh
- $x denotes x:pub
- #i denotes i:temporal
- m denotes m:msg

Example Properties declarations

- Security properties are defined over traces of the action facts of a protocol execution

```

lemma Client_session_key_secrecy:
  " /* It cannot be that a  */
    not(
      Ex S k #i #j.   //存在这样的服务端，和密钥 k，在两个时刻 i 和 j
        /* client has set up a session key 'k' with a server'S' */
        SessKeyC(S, k) @ #i   //在 i 时刻，客户端与服务端 S 建立通信密钥 k
        /* and the adversary knows 'k' */
      & K(k) @ #j   //在 j 时刻，攻击者知道通信密钥 k
        /* without having performed a long-term key reveal on 'S'. */
      & not(Ex #r. LtkReveal(S) @r) //但是攻击者从未攻破服务器
    )
  "
```

Example Properties declarations

```
lemma Client_auth:
  " /* For all session keys 'k' setup by clients with a server 'S' */
    ( All S k #i.  SessKeyC(S, k) @ #i
       ==>
         /* there is a server that answered the request */
       ( (Ex #a. AnswerRequest(S, k) @ a)
         /* or the adversary performed a long-term key reveal on 'S'
            before the key was setup. */
       | (Ex #r. LtkReveal(S) @ r & r < i)
       )
    )
  "
```

Example Properties declarations

```
lemma Client_auth_injective:
  " /* For all session keys 'k' setup by clients with a server 'S' */
    ( All S k #i.  SessKeyC(S, k) @ #i
       ==>
         /* there is a server that answered the request */
       ( (Ex #a. AnswerRequest(S, k) @ a
           /* and there is no other client that had the same request */
           & (All #j. SessKeyC(S, k) @ #j ==> #i = #j)
       )
         /* or the adversary performed a long-term key reveal on 'S'
            before the key was setup. */
       | (Ex #r. LtkReveal(S) @ r & r < i)
       )
    )
  "
```

Example Properties declarations

```
lemma Client_session_key_honest_setup:
  exists-trace
  " Ex S k #i.
        SessKeyC(S, k) @ #i
      & not(Ex #r. LtkReveal(S) @ r)
  "

end
```