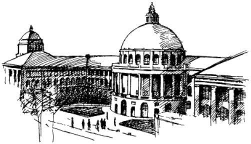
Symbolic verification of cryptographic protocols using Tamarin

Part 2 : Symbolic Verification

David Basin
ETH Zurich



Summer School on Verification Technology, Systems & Applications
Nancy France
August 2018

Outline

1. Formal Models
2. Term Rewriting
3. Rewriting-based Protocol Syntax
4. The Dolev-Yao-Style Adversary
5. Protocol Semantics

1. Formal Models
2. Term Rewriting
3. Rewriting-based Protocol Syntax
4. The Dolev-Yao-Style Adversary
5. Protocol Semantics

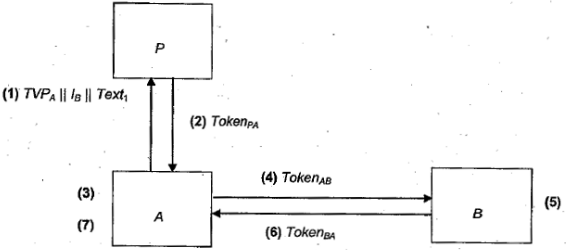


Figure 5 — Mechanism 5 — Four-pass authentication

The form of the token ($Token_{PA}$), sent by P to A, is:

$$Token_{PA} = Text_4 \,||\, e_{K_{AP}}(TVP_A \,||\, K_{AB} \,||\, I_B \,||\, Text_3) \,||\, e_{K_{BP}}(TN_P \,||\, K_{AB} \,||\, I_A \,||\, Text_2)$$

The form of the token ($Token_{AB}$), sent by A to B, is:

$$Token_{AB} = Text_6 \,||\, e_{K_{BP}}(TN_P \,||\, K_{AB} \,||\, I_A \,||\, Text_2) \,||\, e_{K_{AB}}(TN_A \,||\, I_B \,||\, Text_5)$$

The form of the token ($Token_{BA}$), sent by B to A, is:

$$Token_{BA} = Text_8 \,||\, e_{K_{AB}}(TN_B \,||\, I_A \,||\, Text_7)$$

The choice of using either time stamps or sequence numbers in this mechanism depends on the capabilities of the entities involved as well as on the environment.

Real-world protocol specifications: IKE RFC

```
When using encryption for authentication, Main Mode is defined as
follows.

     Initiator                          Responder
    -----------                        -----------
     HDR, SA                     -->
                                 <--    HDR, SA
     HDR, KE, [ HASH(1), ]
       <IDii_b>PubKey_r,
         <Ni_b>PubKey_r          -->
                                        HDR, KE, <IDir_b>PubKey_i,
                                 <--            <Nr_b>PubKey_i
     HDR*, HASH_I                -->
                                 <--    HDR*, HASH_R

Aggressive Mode authenticated with encryption is described as
follows:

     Initiator                          Responder
    -----------                        -----------
     HDR, SA, [ HASH(1),] KE,
       <IDii_b>Pubkey_r,
        <Ni_b>Pubkey_r           -->
                                        HDR, SA, KE, <IDir_b>PubKey_i,
                                 <--         <Nr_b>PubKey_i, HASH_R
     HDR, HASH_I                 -->
```

Real-world protocol specifications: IKE RFC

```
Harkins & Carrel            Standards Track                    [Page 9]

RFC 2409                          IKE                     November 1998

     For signatures:           SKEYID = prf(Ni_b | Nr_b, g^xy)
     For public key encryption: SKEYID = prf(hash(Ni_b | Nr_b), CKY-I |
   CKY-R)
     For pre-shared keys:      SKEYID = prf(pre-shared-key, Ni_b |
   Nr_b)

   The result of either Main Mode or Aggressive Mode is three groups of
   authenticated keying material:

      SKEYID_d = prf(SKEYID, g^xy | CKY-I | CKY-R | 0)
      SKEYID_a = prf(SKEYID, SKEYID_d | g^xy | CKY-I | CKY-R | 1)
      SKEYID_e = prf(SKEYID, SKEYID_a | g^xy | CKY-I | CKY-R | 2)

   and agreed upon policy to protect further communications. The values
   of 0, 1, and 2 above are represented by a single octet. The key used
   for encryption is derived from SKEYID_e in an algorithm-specific
   manner (see appendix B).

   To authenticate either exchange the initiator of the protocol
   generates HASH_I and the responder generates HASH_R where:

    HASH_I = prf(SKEYID, g^xi | g^xr | CKY-I | CKY-R | SAi_b | IDii_b )
    HASH_R = prf(SKEYID, g^xr | g^xi | CKY-R | CKY-I | SAi_b | IDir_b )

   For authentication with digital signatures, HASH_I and HASH_R are
   signed and verified; for authentication with either public key
   encryption or pre-shared keys, HASH_I and HASH_R directly
   authenticate the exchange.  The entire ID payload (including ID type,
   port, and protocol but excluding the generic header) is hashed into
   both HASH_I and HASH_R.
```

- A **language** is **formal** when it has a well-defined syntax and semantics. Additionally there is often a deductive system for determining the truth of statements.
- **Examples:**

What are formal models?

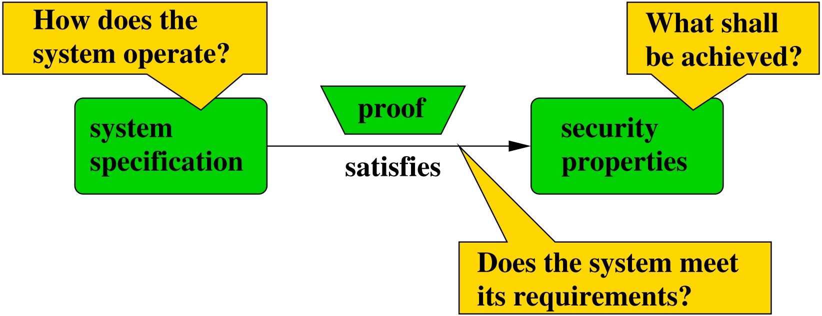
- A **language** is **formal** when it has a well-defined syntax and semantics. Additionally there is often a deductive system for determining the truth of statements.
- **Examples:** propositional logic, first-order logic.
- A **model** (or **construction**) is **formal** when it is specified in a formal language.
- Standard protocol notation is not formal.
- We will see how to formalize such notations.

Formal modeling and analysis of protocols

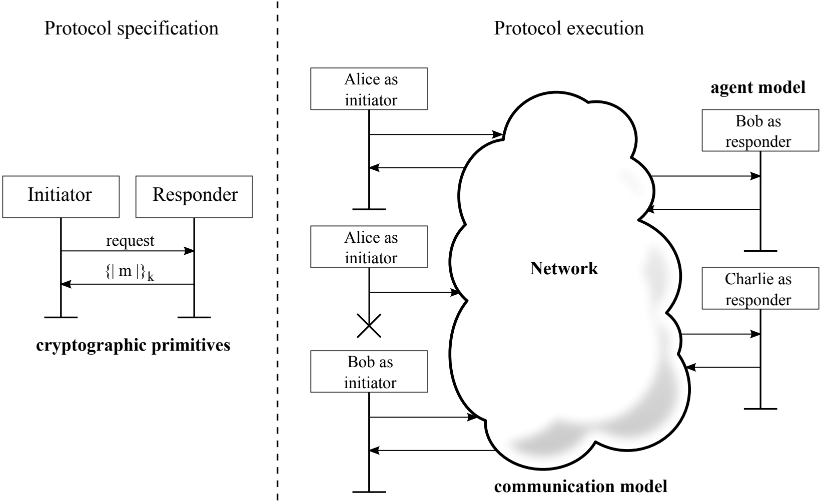
Goal: formally model protocols and their properties and provide a mathematically sound means to reason about these models.

Basis: suitable abstraction of protocols.

Analysis: with formal methods based on mathematics and logic, e.g., theorem proving.


How does the system operate?
What shall be achieved?
proof
system specification
satisfies
security properties
Does the system meet its requirements?

From message sequence charts to protocol execution



1 Formal Models

2 Term Rewriting

3 Rewriting-based Protocol Syntax

4 The Dolev-Yao-Style Adversary

5 Protocol Semantics

Term Rewriting is

- a useful and flexible formalism in general.
    - Programming languages
    - Automated deduction
    - Rewriting logic
- used for representing messages and protocols in Tamarin.

Example: $senc(m, k)$ represents the symmetric encryption of m with key k

Definition (Signature)

An unsorted signature Σ is a set of function symbols, each having an arity $n \geq 0$. We call function symbols of arity 0 constants.

Definition (Signature)

An unsorted signature Σ is a set of function symbols, each having an arity $n \geq 0$. We call function symbols of arity 0 constants.

Example (Peano notation for natural numbers)

$\Sigma = \{0, s, +\}$, where 0 is a constant, s has arity 1 and represents the successor function, and $+$ has arity 2 and represents addition. Note that for binary operators we sometimes will use infix notation.

Definition (Term Algebra)

Let Σ be a signature, $\mathcal{X}$ a set of variables, and $\Sigma \cap \mathcal{X} = \emptyset$. We call the set $\mathcal{T}_\Sigma(\mathcal{X})$ the term algebra over Σ. It is the least set such that:

- $\mathcal{X} \subseteq \mathcal{T}_\Sigma(\mathcal{X})$.
- If $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{X})$ and $f \in \Sigma$ with arity n, then $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(\mathcal{X})$.

The set of ground terms $\mathcal{T}_\Sigma$ consists of terms built without variables, i.e., $\mathcal{T}_\Sigma := \mathcal{T}_\Sigma(\emptyset)$.

Definition (Term Algebra)

Let Σ be a signature, $\mathcal{X}$ a set of variables, and $\Sigma \cap \mathcal{X} = \emptyset$. We call the set $\mathcal{T}_\Sigma(\mathcal{X})$ the term algebra over Σ. It is the least set such that:

- $\mathcal{X} \subseteq \mathcal{T}_\Sigma(\mathcal{X})$.
- If $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{X})$ and $f \in \Sigma$ with arity n, then $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(\mathcal{X})$.

The set of ground terms $\mathcal{T}_\Sigma$ consists of terms built without variables, i.e., $\mathcal{T}_\Sigma := \mathcal{T}_\Sigma(\emptyset)$.

Exercise: constants are included in $\mathcal{T}_\Sigma$ and $\mathcal{T}_\Sigma(\mathcal{X})$.

Definition (Term Algebra)

Let Σ be a signature, $\mathcal{X}$ a set of variables, and $\Sigma \cap \mathcal{X} = \emptyset$. We call the set $\mathcal{T}_\Sigma(\mathcal{X})$ the term algebra over Σ. It is the least set such that:

- $\mathcal{X} \subseteq \mathcal{T}_\Sigma(\mathcal{X})$.
- If $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{X})$ and $f \in \Sigma$ with arity n, then $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(\mathcal{X})$.

The set of ground terms $\mathcal{T}_\Sigma$ consists of terms built without variables, i.e., $\mathcal{T}_\Sigma := \mathcal{T}_\Sigma(\emptyset)$.

Exercise: constants are included in $\mathcal{T}_\Sigma$ and $\mathcal{T}_\Sigma(\mathcal{X})$.

Example (Peano notation for natural numbers (continued))

$s(0) \in \mathcal{T}_\Sigma$
$s(s(0)) + s(X) \in \mathcal{T}_\Sigma(\mathcal{X})$
$+s(0)+ \notin \mathcal{T}_\Sigma(\mathcal{X})$

Cryptographic Messages

We generally denote variables with upper case names $X, Y, \ldots$, and function symbols (inc. constants) with lower case names $a, b, \ldots$

Definition (Messages)

A message is a term in $\mathcal{T}_\Sigma(\mathcal{X})$, where $\Sigma = \mathcal{A} \cup \mathcal{F} \cup \mathit{Func} \cup \{\mathit{pair}, \mathit{pk}, \mathit{aenc}, \mathit{senc}\}$. We call

$\mathcal{X}$	the set of variables A, B, X, Y, Z, ...,
$\mathcal{A}$	the set of agents a, b, c, ...,
$\mathcal{F}$	the set of fresh values na, nb, k (nonces, keys, ...),
Func	the set of user-defined functions (hash, exp, ...),
$\mathit{pair}(t_1, t_2)$	pairing, also denoted by $\langle t_1, t_2 \rangle$,
$\mathrm{pk}(t)$	public key,
$\mathit{aenc}(t_1, t_2)$	asymmetric encryption, also denoted by $\{t_1\}_{t_2}$,
$\mathit{senc}(t_1, t_2)$	symmetric encryption, also denoted by $\{\mid t_1 \mid\}_{t_2}$.

Definition (Free Algebra)

In the free algebra every term is interpreted by itself (syntactically).

Example (Equational theory for symmetric cryptography)

$\Sigma = \mathcal{A} \cup \mathcal{F} \cup \{senc, sdec\}$, with *senc* and *sdec* of arity 2.
(E: $sdec(senc(M, K), K) = M$)

- $t_1 =_{\text{free}} t_2$ iff $t_1 =_{\text{syntactic}} t_2$.
- $a \neq_{\text{free}} b$ for different constants a and b.
- For above example: $sdec(senc(X, Y), Y) \neq_{\text{free}} X$.

This is too coarse as we clearly want to identify those two terms.
Hence, we will need to reason modulo equations.

Definition (Equation)

An equation is a pair of terms, written: $t = t'$, and a set of equations is called an equational theory (Σ, E).

An equation can be oriented as $t \rightarrow t' \in \vec{E}$ or as $t \leftarrow t' \in \overleftarrow{E}$. Equations are usually oriented left to right for use in simplification.

Definition (Equation)

An equation is a pair of terms, written: $t = t'$, and a set of equations is called an equational theory (Σ, E).

An equation can be oriented as $t \rightarrow t' \in \vec{E}$ or as $t \leftarrow t' \in \overleftarrow{E}$. Equations are usually oriented left to right for use in simplification.

Example (Peano natural numbers (continued))

The equations E defining the Peano natural numbers are:
$X + 0 = X$
$X + s(Y) = s(X + Y)$

Rewriting $s(s(0)) + s(0)$ using $\vec{E}$ yields the equational derivation:
$s(s(0)) + s(0) =$

Definition (Equation)

An equation is a pair of terms, written: $t = t'$, and a set of equations is called an equational theory (Σ, E).

An equation can be oriented as $t \rightarrow t' \in \vec{E}$ or as $t \leftarrow t' \in \overleftarrow{E}$. Equations are usually oriented left to right for use in simplification.

Example (Peano natural numbers (continued))

The equations E defining the Peano natural numbers are:
$X + 0 = X$
$X + s(Y) = s(X + Y)$

Rewriting $s(s(0)) + s(0)$ using $\vec{E}$ yields the equational derivation:
$s(s(0)) + s(0) = s(s(s(0)) + 0) =$

Definition (Equation)

An equation is a pair of terms, written: $t = t'$, and a set of equations is called an equational theory (Σ, E).

An equation can be oriented as $t \rightarrow t' \in \vec{E}$ or as $t \leftarrow t' \in \overleftarrow{E}$. Equations are usually oriented left to right for use in simplification.

Example (Peano natural numbers (continued))

The equations E defining the Peano natural numbers are:
$X + 0 = X$
$X + s(Y) = s(X + Y)$

Rewriting $s(s(0)) + s(0)$ using $\vec{E}$ yields the equational derivation:
$s(s(0)) + s(0) = s(s(s(0)) + 0) = s(s(s(0)))$.

Example (Equations E)

$$\{\{M\}_K\}_{(K)^{-1}} = M \qquad ((K)^{-1})^{-1} = K$$
$$\{|\{|M|\}_K|\}_K = M \qquad \exp(\exp(B,X),Y) = \exp(\exp(B,Y),X)$$

Definition (Congruence, Equivalence, Quotient)

Set of equations E induces a congruence relation $=_E$ on terms and thus the equivalence class $[t]_E$ of a term modulo E. The quotient algebra $\mathcal{T}_\Sigma(\mathcal{X})/_{=_E}$ interprets each term by its equivalence class.

- Two terms are semantically equal iff that is a consequence of E.

Example (Equations E)

$$\begin{aligned} \{\{M\}_K\}_{(K)^{-1}} &= M & ((K)^{-1})^{-1} &= K \\ \{|\{|M|\}_K|\}_K &= M & \exp(\exp(B,X),Y) &= \exp(\exp(B,Y),X) \end{aligned}$$

Definition (Congruence, Equivalence, Quotient)

Set of equations E induces a congruence relation $=_E$ on terms and thus the equivalence class $[t]_E$ of a term modulo E. The quotient algebra $\mathcal{T}_\Sigma(\mathcal{X})/_{=_E}$ interprets each term by its equivalence class.

- Two terms are semantically equal iff that is a consequence of E.
- For the above example equations:
  - $a \neq_E b$ for any distinct constants a and b

Example (Equations E)

$$
\begin{aligned}
\{\{M\}_K\}_{(K)^{-1}} &= M & ((K)^{-1})^{-1} &= K \\
\{|\{|M|\}_K|\}_K &= M & \exp(\exp(B, X), Y) &= \exp(\exp(B, Y), X)
\end{aligned}
$$

Definition (Congruence, Equivalence, Quotient)

Set of equations E induces a congruence relation $=_E$ on terms and thus the equivalence class $[t]_E$ of a term modulo E. The quotient algebra $\mathcal{T}_\Sigma(\mathcal{X})/_{=_E}$ interprets each term by its equivalence class.

- Two terms are semantically equal iff that is a consequence of E.
- For the above example equations:
  - $a \neq_E b$ for any distinct constants a and b
  - If $m_1 \neq_E m_2$ then also $h(m_1) \neq_E h(m_2)$

Example (Equations E)

$$\{\{M\}_K\}_{(K)^{-1}} = M \qquad ((K)^{-1})^{-1} = K$$
$$\{\!|\{\!|M|\!\}_K|\!\}_K = M \qquad \exp(\exp(B,X),Y) = \exp(\exp(B,Y),X)$$

Definition (Congruence, Equivalence, Quotient)

Set of equations E induces a congruence relation $=_E$ on terms and thus the equivalence class $[t]_E$ of a term modulo E. The quotient algebra $\mathcal{T}_\Sigma(\mathcal{X})/_{=_E}$ interprets each term by its equivalence class.

- Two terms are semantically equal iff that is a consequence of E.
- For the above example equations:
  - $a \neq_E b$ for any distinct constants a and b
  - If $m_1 \neq_E m_2$ then also $h(m_1) \neq_E h(m_2)$
  - $\{\{M\}_{(K)^{-1}}\}_K =_E M$

Example (Equations E)

$$\begin{aligned} \{\{M\}_K\}_{(K)^{-1}} &= M & ((K)^{-1})^{-1} &= K \\ \{\!|\{\!|M|\!\}_K|\!\}_K &= M & \exp(\exp(B,X),Y) &= \exp(\exp(B,Y),X) \end{aligned}$$

Definition (Congruence, Equivalence, Quotient)

Set of equations E induces a congruence relation $=_E$ on terms and thus the equivalence class $[t]_E$ of a term modulo E. The quotient algebra $\mathcal{T}_\Sigma(\mathcal{X})/_{=_E}$ interprets each term by its equivalence class.

- Two terms are semantically equal iff that is a consequence of E.
- For the above example equations:
  - $a \neq_E b$ for any distinct constants a and b
  - If $m_1 \neq_E m_2$ then also $h(m_1) \neq_E h(m_2)$
  - $\{\{M\}_{(K)^{-1}}\}_K =_E M$
  - $\{\!|\{\!|M|\!\}_{\exp(\exp(g,Y),X)}|\!\}_{\exp(\exp(g,X),Y)} =_E M$

Definition (Substitution)

A substitution σ is a function $\sigma : \mathcal{X} \to \mathcal{T}_\Sigma(\mathcal{X})$ where $\sigma(x) \neq x$ for finitely many $x \in \mathcal{X}$.
We write substitutions in postfix notation and homomorphically extend them to a mapping $\sigma : \mathcal{T}_\Sigma(\mathcal{X}) \to \mathcal{T}_\Sigma(\mathcal{X})$ on terms:

$$f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$$

Definition (Substitution)

A substitution σ is a function $\sigma : \mathcal{X} \to \mathcal{T}_\Sigma(\mathcal{X})$ where $\sigma(x) \neq x$ for finitely many $x \in \mathcal{X}$.
We write substitutions in postfix notation and homomorphically extend them to a mapping $\sigma : \mathcal{T}_\Sigma(\mathcal{X}) \to \mathcal{T}_\Sigma(\mathcal{X})$ on terms:

$$f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$$

Example (Applying a substitution)

Given substitution $\sigma = \{X \mapsto senc(M, K)\}$ and the term $t = sdec(X, K)$ we can apply the substitution and get $t\sigma = sdec(senc(M, K), K)$.

Definition (Substitution composition)

We denote with $\sigma\tau$ the composition of substitutions σ and τ, i.e., $\tau \circ \sigma$.

Example (Substitution composition)

For substitutions $\sigma = [x \mapsto f(y), y \mapsto z]$ and $\tau = [y \mapsto a, z \mapsto g(b)]$ we have $\sigma\tau = [x \mapsto f(a), y \mapsto g(b), z \mapsto g(b)]$.

Definition (Position)

A position p is a sequence of positive integers. The subterm $t|_p$ of a term t at position p is obtained as follows.

- If $p = [\,]$ is the empty sequence, then $t|_p = t$.
- If $p = [i] \cdot p'$ for a positive integer i and a sequence p', and $t = f(t_1, \ldots, t_n)$ for $f \in \Sigma$ and $1 \leq i \leq n$ then $t|_p = t_i|_{p'}$, else $t|_p$ does not exist.

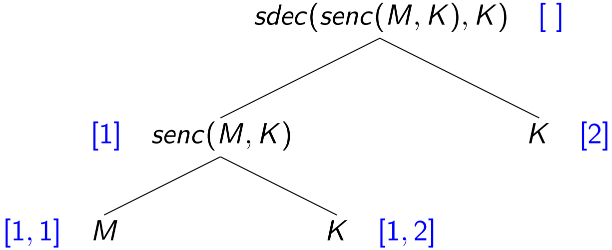
Example (Position in a term)

For the term $t = sdec(senc(M, K), K)$ we have five subterms:

$t|_{[\,]} = t$
$t|_{[1]} = senc(M, K)$
$t|_{[1,1]} = M$
$t|_{[1,2]} = K$
$t|_{[2]} = K$

Graphical representation of positions in a term

Tree of subterms of $sdec(senc(M, K))$ and their positions.



Definition (Matching)

A term t matches another term l if there is a subterm of t, i.e., $t|_p$, such that there is a substitution σ so that $t|_p = l\sigma$. We call σ the matching substitution.

Definition (Matching)

A term t matches another term l if there is a subterm of t, i.e., $t|_p$, such that there is a substitution σ so that $t|_p = l\sigma$. We call σ the matching substitution.

Definition (Application of a rule)

A rule (oriented equation) $l \to r$ is applicable on a term t, when t matches l.

The result of such a rule application is the term $t[r\sigma]_p$, where σ is the matching substitution.

Definition (Unification)

We say that $t \stackrel{?}{=} t'$ is unifiable in (Σ, E) for $t, t' \in \mathcal{T}_{\Sigma}(\mathcal{X})$, if there is a substitution σ such that $t\sigma =_E t'\sigma$ and we call σ a unifier.

Definition (Unification)

We say that $t \stackrel{?}{=} t'$ is unifiable in (Σ, E) for $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, if there is a substitution σ such that $t\sigma =_E t'\sigma$ and we call σ a unifier.

For syntactic unification ($E = \emptyset$) there is a most general unifier for two unifiable terms, and it is decidable whether they are unifiable.

Definition (Unification)

We say that $t \stackrel{?}{=} t'$ is unifiable in (Σ, E) for $t, t' \in \mathcal{T}_{\Sigma}(\mathcal{X})$, if there is a substitution σ such that $t\sigma =_E t'\sigma$ and we call σ a unifier.

For syntactic unification ($E = \emptyset$) there is a most general unifier for two unifiable terms, and it is decidable whether they are unifiable.

Unification modulo theories ($E \neq \emptyset$) is much more complicated: undecidable in general, or potentially (infinitely) many unifiers.

Definition (Unification)

We say that $t \stackrel{?}{=} t'$ is unifiable in (Σ, E) for $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, if there is a substitution σ such that $t\sigma =_E t'\sigma$ and we call σ a unifier.

For syntactic unification ($E = \emptyset$) there is a most general unifier for two unifiable terms, and it is decidable whether they are unifiable.

Unification modulo theories ($E \neq \emptyset$) is much more complicated: undecidable in general, or potentially (infinitely) many unifiers.

This is no good for automated analysis: we need to restrict ourselves.

Definition (Termination)

$(\Sigma, \vec{E})$ has infinite computations if there is a function $a : \mathbb{N} \to \mathcal{T}_{\Sigma}(\mathcal{X})$ such that

$$a(0) \to_{\vec{E}} a(1) \to_{\vec{E}} a(2) \to_{\vec{E}} \dots \to_{\vec{E}} a(n) \to_{\vec{E}} a(n+1) \dots$$

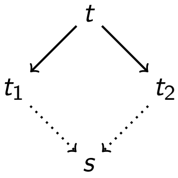
We say $(\Sigma, \vec{E})$ it is terminating when it does not have infinite computations.

Example (Termination)

For $E = \{a = b\}$, $\vec{E}$ is terminating.
For $E = \{a = b, b = a\}$, $\vec{E}$ is not terminating.

Definition (Confluence)

Confluence is the property that guarantees the order of applying equalities is immaterial, formally:
$\forall t, t_1, t_2 . t \rightarrow^* t_1 \wedge t \rightarrow^* t_2 \Rightarrow \exists s . t_1 \rightarrow^* s \wedge t_2 \rightarrow^* s$



Example (Confluence)

For $E = \{a = b, a = c\}$, we have that $\vec{E}$ is not confluent, as b and c are reachable from a, but not joinable.
For $E = \{a = b, a = c, b = c\}$, then $\vec{E}$ is confluent.

Tamarin supports (see Tamarin manual for details)

- any user-defined equational theory that is convergent (confluent and terminating) with finite variant property
- special built-in theories: Diffie-Hellman exponentiation, bilinear pairing, multisets, XOR (soon...)

Example (Tamarin Syntax)

```
functions: h/1, senc/2, sdec/2
equations: sdec(senc(m,k),k) = m
builtins: diffie-hellman, bilinear-pairing, multiset

/* There are also other convenient builtins:
   hashing, asymmetric-encryption, symmetric-encryption,
   signing, revealing-signing */
```

Plan

1. Formal Models
2. Term Rewriting
3. Rewriting-based Protocol Syntax
4. The Dolev-Yao-Style Adversary
5. Protocol Semantics

In Tamarin, protocols are modeled using rewrite rules operating on multisets of facts:

$$l \xrightarrow{a} r$$

where l, a, and r are multisets of facts, l is called the left hand side, r the right hand side, and a the actions of the rule.

The rule's left and right sides specify which facts are consumed or produced when executing the rule, the actions are recorded as event labels on the trace and are used to specify properties.

Multiset rewriting in Tamarin: example

Example

- rule 1: $\xrightarrow{\text{Init()}} \text{A}('5'), \text{C}('3')$ ('x' is a constant)
- rule 2: $\text{A}(x) \xrightarrow{\text{Step}(x)} \text{B}(x)$

or in Tamarin syntax:

```
rule 1: [ ] --[ Init() ]-> [ A('5'), C('3') ]
rule 2: [ A(x) ] --[ Step(x) ]-> [ B(x) ]

// A rule without action:
rule 3: [ C(x) ] --> [ D(x) ]
```

Definition (Fresh terms)

Agents generate fresh terms using fresh facts, denoted by Fr. These fresh terms represent randomness being used, are assumed unguessable and unique, i.e., can represent nonces.

There is a countable supply of fresh terms, each as argument of a fresh fact, usable in rules.

In Tamarin, fresh variables are prefixed with a $\sim$, e.g., `~r`.

Definition (Public terms)

We define public terms to be terms known to all participants of a protocol. These include all agent names and all constants.

In Tamarin, public variables are prefixed with a $, e.g., `$X`.

Communication and persistent facts

Messages are sent and received via Out (output to the network) and In (input from the network) facts, respectively.

Example (Input and Output)

```
rule 3: [ Key(x), In(y) ] --> [ Out( senc(y,x) ) ]
```

Messages are sent and received via Out (output to the network) and In (input from the network) facts, respectively.

Example (Input and Output)

```
rule 3: [ Key(x), In(y) ] --> [ Out( senc(y,x) ) ]
```

Facts can be linear or persistent.

- Linear facts can only be consumed once
- Persistent facts can be consumed infinitely often.

Persistent facts are marked with a ! in Tamarin, e.g.:

```
rule key-reveal:
 [ !Ltk(~k) ] --[ Reveal(~k) ]-> [ Out(~k) ]
```

By default, facts are linear.

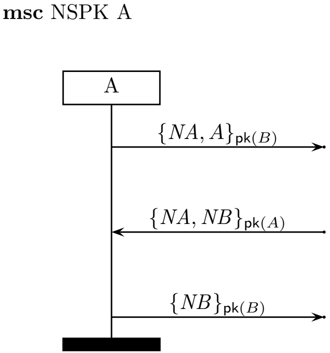
Protocol rules must be well-formed.

Definition (Well-formedness)

For a protocol rule $l \xrightarrow{a} r$ to be well-formed, the following conditions must hold.

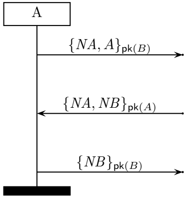
1. In and Fr, only occur in l.
2. Out only occurs in r.
3. Every variable in r or a that is not public must occur in l.
4. All occurrences of the same fact have the same arity, and the same persistence.

Graphical:



Protocol rules

msc NSPK A

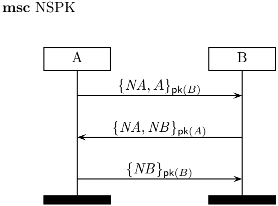


$$[\mathrm{St\_A\_1}(A, tid, skA, pk(skB)),\quad \mathrm{Fr}(NA)] \rightarrow$$
$$[\mathrm{St\_A\_2}(A, tid, skA, pk(skB), NA),\quad \mathrm{Out}(\{NA, A\}_{pk(skB)})]$$

$$[\mathrm{St\_A\_2}(A, tid, skA, pk(skB), NA),\quad \mathrm{In}(\{NA, NB\}_{pk(skA)})] \rightarrow$$
$$[\mathrm{St\_A\_3}(A, tid, skA, pk(skB), NA, NB)]$$

$$[\mathrm{St\_A\_3}(A, tid, skA, pk(skB), NA, NB)] \rightarrow$$
$$[\mathrm{St\_A\_4}(A, tid, skA, pk(skB), NA, NB),\quad \mathrm{Out}(\{NB\}_{pk(skB)})]$$

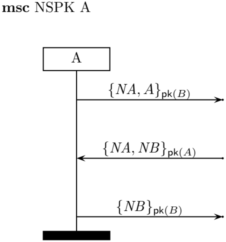
Be careful: pattern matching!



Generate longterm keys and public keys.

$$[\mathsf{Fr}(\mathit{skR})] \rightarrow [!\mathsf{Ltk}(R, \mathit{skR}), \mathsf{Out}(\mathit{pk}(\mathit{skR}))]$$

Initialization of protocol roles



$$[\mathsf{Fr}(id), !\mathsf{Ltk}(A, skA), !\mathsf{Ltk}(B, skB)] \xrightarrow{\mathsf{Create(A,id)}} [\mathsf{St\_A\_1}(A, id, skA, pk(skB))]$$

Plan

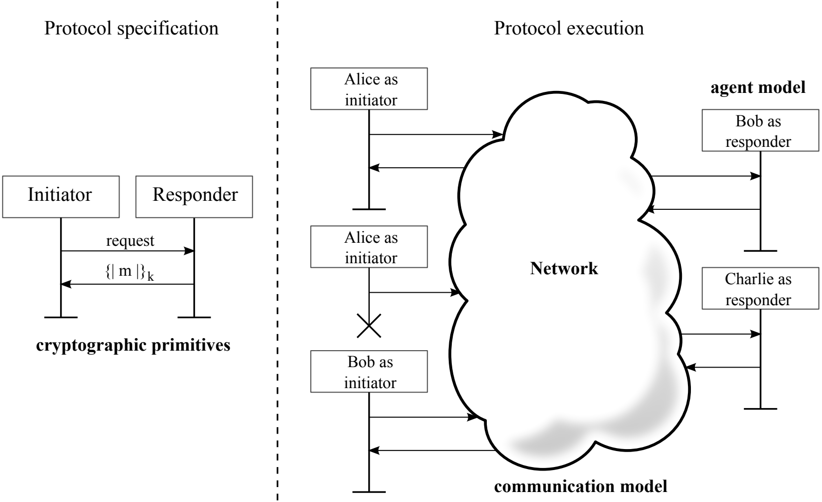
1 Formal Models

2 Term Rewriting

3 Rewriting-based Protocol Syntax

4 The Dolev-Yao-Style Adversary

5 Protocol Semantics


Protocol specification
Initiator
Responder
request
{| m |}k
cryptographic primitives
Protocol execution
Alice as initiator
Alice as initiator
Bob as initiator
agent model
Bob as responder
Charlie as responder
Network
communication model

Danny Dolev & Andrew C. Yao

On the Security of Public Key Protocols (IEEE Trans. Inf. Th., 1983)

- Consider a public key system wherre for every user X
  - there is a public encryption function E_X
    — every user can apply this function.
  - and a private decryption function D_X
    — only X can apply this function.
  - These functions have the property that $E_X D_X = D_X E_X = 1$.
- The **Dolev-Yao adversary**:
  - Controls the network (read, intercept, send)
  - Is also a user, called Z
  - Can apply E_X for any X
  - Can apply D_Z

Definition (Adversary Knowledge)

We represent the adversary knowing a term t by a fact $\mathsf{K}(t)$. The set of the adversary's knowledge is $\mathcal{K}$ and contains persistent facts of the form $\mathsf{K}(t)$.

Definition (Adversary Knowledge Derivation)

The adversary can use the following inference rules on the state:

$$\frac{\mathsf{Fr}(x)}{\mathsf{K}(x)} \qquad \frac{\mathsf{Out}(x)}{\mathsf{K}(x)} \qquad \frac{\mathsf{K}(x)}{\mathsf{In}(x)}$$

$$\frac{\mathsf{K}(t_1)\ldots\mathsf{K}(t_k)}{\mathsf{K}(f(t_1,...,t_k))}\ \forall f \in \Sigma(\text{k-ary})$$

N.B. terms are used modulo the equational theory. So, given $\mathsf{K}(< t_1, t_2 >)$ the operator *fst* can be applied, and result is $\mathsf{K}(t_1)$.

Example

Given $\mathsf{K}(x), \mathsf{K}(\{\!|b, n|\!\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})$ where *prf* is some (constructible) function.

$$
\dfrac{\dfrac{}{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{K}(\{\!|b, n|\!\}_k)} \quad \dfrac{}{\mathsf{K}(k)}}{\mathsf{K}(\{\!|\{\!|b, n|\!\}_k|\!\}_k)}}{\mathsf{K}(b, n)}\,E}{\mathsf{K}(snd(b, n))}}{\mathsf{K}(n)}\,E \qquad \dfrac{}{\mathsf{K}(x)}}{\mathsf{K}(prf(n, x))}}{\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})}
$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{|b,n|\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{|m|\}_{prf(n,x)})$ where *prf* is some (constructible) function.

$$\dfrac{\overline{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\mathsf{K}(\{|b,n|\}_k)} \quad \overline{\mathsf{K}(k)}}{\mathsf{K}(\{|\{|b,n|\}_k|\}_k)}}{\mathsf{K}(b,n)}\,E}{\mathsf{K}(snd(b,n))}}{\mathsf{K}(n)}\,E \qquad \overline{\mathsf{K}(x)}}{\mathsf{K}(prf(n,x))}}{\mathsf{K}(\{|m|\}_{prf(n,x)})}$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{\!|b, n|\!\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})$ where prf is some (constructible) function.

$$\dfrac{\dfrac{}{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{K}(\{\!|b, n|\!\}_k)} \quad \dfrac{}{\mathsf{K}(k)}}{\mathsf{K}(\{\!|\{\!|b, n|\!\}_k|\!\}_k)}}{\mathsf{K}(b, n)}\,E}{\mathsf{K}(snd(b, n))}}{\mathsf{K}(n)}\,E \qquad \dfrac{}{\mathsf{K}(x)}}{\mathsf{K}(prf(n, x))}}{\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})}$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{\!|b, n|\!\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})$ where prf is some (constructible) function.

$$
\frac{\dfrac{}{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{K}(\{\!|b, n|\!\}_k)} \quad \dfrac{}{\mathsf{K}(k)}}{\mathsf{K}(\{\!|\{\!|b, n|\!\}_k|\!\}_k)}}{\mathsf{K}(b, n)}\ E}{\mathsf{K}(snd(b, n))}}{\mathsf{K}(n)}\ E \qquad \dfrac{}{\mathsf{K}(x)}}{\mathsf{K}(prf(n, x))}}{\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})}
$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{\!|b, n|\!\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})$ where prf is some (constructible) function.

$$\dfrac{\dfrac{}{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{K}(\{\!|b, n|\!\}_k)} \quad \dfrac{}{\mathsf{K}(k)}}{\mathsf{K}(\{\!|\{\!|b, n|\!\}_k|\!\}_k)}}{\mathsf{K}(b, n)}\,E}{\mathsf{K}(snd(b, n))}}{\mathsf{K}(n)}\,E \qquad \dfrac{}{\mathsf{K}(x)}}{\mathsf{K}(prf(n, x))}}{\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})}$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{|b, n|\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{|m|\}_{prf(n,x)})$ where prf is some (constructible) function.

$$\dfrac{\overline{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\mathsf{K}(\{|b,n|\}_k)} \quad \overline{\mathsf{K}(k)}}{\mathsf{K}(\{|\{|b,n|\}_k|\}_k)}}{\mathsf{K}(b,n)}\,E}{\mathsf{K}(snd(b,n))}}{\mathsf{K}(n)}\,E \qquad \overline{\mathsf{K}(x)}}{\mathsf{K}(prf(n,x))}}{\mathsf{K}(\{|m|\}_{prf(n,x)})}$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{\!|b, n|\!\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})$ where prf is some (constructible) function.

$$\dfrac{\dfrac{}{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{K}(\{\!|b, n|\!\}_k)} \quad \dfrac{}{\mathsf{K}(k)}}{\mathsf{K}(\{\!|\{\!|b, n|\!\}_k|\!\}_k)}}{\mathsf{K}(b, n)}\,E}{\mathsf{K}(snd(b, n))}}{\mathsf{K}(n)}\,E \qquad \dfrac{}{\mathsf{K}(x)}}{\mathsf{K}(prf(n, x))}}{\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})}$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{\!|b, n|\!\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})$ where *prf* is some (constructible) function.

$$
\dfrac{\dfrac{}{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{K}(\{\!|b, n|\!\}_k)} \quad \dfrac{}{\mathsf{K}(k)}}{\mathsf{K}(\{\!|\{\!|b, n|\!\}_k|\!\}_k)}}{\mathsf{K}(b, n)}\,E}{\mathsf{K}(snd(b, n))}}{\mathsf{K}(n)}\,E \qquad \dfrac{}{\mathsf{K}(x)}}{\mathsf{K}(prf(n, x))}}{\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})}
$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{\!|b, n|\!\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})$ where prf is some (constructible) function.

$$\dfrac{\dfrac{}{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{K}(\{\!|b, n|\!\}_k)} \quad \dfrac{}{\mathsf{K}(k)}}{\mathsf{K}(\{\!|\{\!|b, n|\!\}_k|\!\}_k)}}{\mathsf{K}(b, n)}\,E}{\mathsf{K}(snd(b, n))}}{\mathsf{K}(n)}\,E \qquad \dfrac{}{\mathsf{K}(x)}}{\mathsf{K}(prf(n, x))}}{\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})}$$

Example

Given $\mathsf{K}(x), \mathsf{K}(\{\!|b, n|\!\}_k), \mathsf{K}(k), \mathsf{K}(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})$ where prf is some (constructible) function.

$$\dfrac{\overline{\mathsf{K}(m)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\mathsf{K}(\{\!|b, n|\!\}_k)} \quad \overline{\mathsf{K}(k)}}{\mathsf{K}(\{\!|\{\!|b, n|\!\}_k|\!\}_k)}}{\mathsf{K}(b, n)}\,E}{\mathsf{K}(snd(b, n))}}{\mathsf{K}(n)}\,E \qquad \overline{\mathsf{K}(x)}}{\mathsf{K}(prf(n, x))}}{\mathsf{K}(\{\!|m|\!\}_{prf(n,x)})}$$

Definition (Adversary Knowledge Derivation as rewrite rules)

$$[\mathsf{Fr}(x)] \rightarrow [\mathsf{K}(x)]$$

$$[\mathsf{Out}(x)] \rightarrow [\mathsf{K}(x)]$$

$$[\mathsf{K}(x)] \xrightarrow{\mathsf{K}(x)} [\mathsf{In}(x)]$$

$$[\mathsf{K}(t_1), \ldots, \mathsf{K}(t_k)] \rightarrow [\mathsf{K}(f(t_1, \ldots, t_k))] \quad \forall f \in \Sigma(\text{k-ary})$$

Note: the adversary deriving a message and then sending it (via In) is annotated with the action fact K (identical to its state fact of the same name!); we use this for our reasoning later.

1. Formal Models
2. Term Rewriting
3. Rewriting-based Protocol Syntax
4. The Dolev-Yao-Style Adversary
5. Protocol Semantics

Outlook

We will define a trace semantics for protocols in terms of labeled transition systems.

Definition (Multiset)

A multiset is a set of elements, each imbued with a multiplicity. Instead of stating an explicit multiplicity, we may also simply write elements multiple times.
We use $\setminus^\sharp$ for the multiset difference, and $\cup^\sharp$ for the union.

Definition (Labeled multiset rewriting)

A labeled multiset rewriting rule is a triple, l, a, r, each of which is a multisets of facts, and written as:

$$l \xrightarrow{a} r$$

Definition (State)

A state is a multiset of facts.

Example (State)

$$\mathsf{St\_R\_1}(A, id, k_1, k_2), \mathsf{Out}(k_1), \mathsf{Out}(k_2), \mathsf{Out}(k_2)$$

Definition (Ground substitutition)

A substitution is called ground when each variable is mapped to a ground term.

Definition (Ground instances)

We call the ground instances of a term t all those terms $t\sigma$ that are ground for some (ground) substitution.
A fact F is ground if all its terms are ground. The multiset of all ground facts is $\mathcal{G}^\sharp$.
For a rule, its ground instances are those where all facts are ground, and we use

$$ginsts(R)$$

for the set of all ground instances of the set of rules R.

Definition (Fresh rule)

We define a special rule that creates fresh facts. This is the only rule allowed to produce fresh facts and has no precondition:

$$[] \rightarrow [\mathsf{Fr}(N)]$$

Note that each created nonce N is fresh, and thus unique.

Definition (Steps)

For a multiset rewrite system R we define the labeled transition relation step, $steps(R) \subseteq \mathcal{G}^\sharp \times ginsts(R) \times \mathcal{G}^\sharp$, as follows:

$$\frac{l \xrightarrow{a} r \in ginsts(R), \qquad l \subseteq^\sharp S, \qquad S' = (S \setminus^\sharp l) \cup^\sharp r}{(S, l \xrightarrow{a} r, S') \in steps(R)}$$

Definition (Execution)

An execution of R is an alternating sequence

$$S_0, (l_1 \xrightarrow{a_1} r_1), S_1, \ldots, S_{k-1}(l_k \xrightarrow{a_k} r_k), S_k$$

of states and multiset rewrite rule instances with

(1) $S_0 = \emptyset$

(2) $\forall i : (S_{i-1}, l_i \xrightarrow{a_i} r_i, S_i) \in steps(R)$

(3) Fresh names are unique, i.e., for n fresh, and $(l_i \xrightarrow{a_i} r_i) = (l_j \xrightarrow{a_j} r_j) = ([] \rightarrow [\mathrm{Fr}(n)])$ it holds that $i = j$.

Definition (Trace)

The trace of an execution

$$S_0, (l_1 \xrightarrow{a_1} r_1), S_1, \ldots, S_{k-1}(l_k \xrightarrow{a_k} r_k), S_k$$

is defined by the sequence of the multisets of its action labels, i.e.:

$$a_1; a_2; \ldots; a_k$$

Two parts:

- State transition
- Trace event

Two parts:

- State transition
- Trace event

Example (Transition example)

$$[\mathrm{St\_I\_2}(A, 17, k), \mathrm{In}(m)] \xrightarrow{\mathrm{Recv}(A,m)} [\mathrm{St\_I\_3}(A, 17, k, m)]$$

Agent state changes, and In fact is consumed, while Recv action is added to trace.

- David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. *Symbolically Analyzing Security Protocols using TAMARIN*, SIGLOG News, 2017.
- Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. *Automated analysis of Diffie-Hellman Protocols and Advanced Security Properties*, Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF) 2012.
- Gavin Lowe. *A hierarchy of authentication specifications*. In Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97), pages 31–43. IEEE CS Press, 1997.
- Peter Ryan, Steve Schneider, Michael Goldsmith, Gawin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols*, Addison-Wesley, 2000.