Model-Checking Security Protocols
Part II: Tamarin

David Basin
Information Security Lab
ETH Zurich

Learning Objectives for Part II

- Learn how to model security protocols in Tamarin
  - ★ Specification language based on multiset rewriting
  - ★ Not all aspects explained in detail.
    You can consult additional material for background theory
  - ★ Emphasis instead on main, high-level ideas
- Learn how to formalize security properties
- Learn how to construct proofs

A (Semi-)Formal Alice & Bob Language

Actions :

$$
\begin{array}{llll}
A & \rightarrow & B & : \quad \{NA, A\}_{\mathsf{pk}(B)} \\
B & \rightarrow & A & : \quad \{NA, NB\}_{\mathsf{pk}(A)} \\
A & \rightarrow & B & : \quad \{NB\}_{\mathsf{pk}(B)}
\end{array}
$$

A (Semi-)Formal Alice & Bob Language

Protocol : *NSPK*

Actions :

$$
\begin{array}{llll}
A & \rightarrow & B & : \ \{NA, A\}_{\mathrm{pk}(B)} \\
B & \rightarrow & A & : \ \{NA, NB\}_{\mathrm{pk}(A)} \\
A & \rightarrow & B & : \ \{NB\}_{\mathrm{pk}(B)}
\end{array}
$$

First, let's give it a name.

A (Semi-)Formal Alice & Bob Language

Protocol : *NSPK*
Types :
  Agent A, B;
  Number NA, NB;
  Function pk;

Actions :
$$A \rightarrow B : \{NA, A\}_{\mathrm{pk}(B)}$$
$$B \rightarrow A : \{NA, NB\}_{\mathrm{pk}(A)}$$
$$A \rightarrow B : \{NB\}_{\mathrm{pk}(B)}$$

Specify the types of all identifiers.
NB: we do not necessarily consider types in the analysis!

A (Semi-)Formal Alice & Bob Language

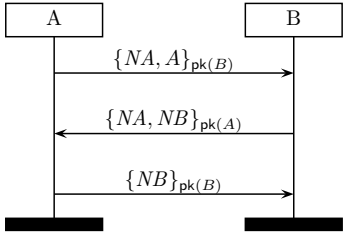
Protocol : $NSPK$
Types :
  Agent A, B;
  Number NA, NB;
  Function pk;
Knowledge :
  A : $A, B, \mathsf{pk}, \mathsf{sk}(A)$;
  B : $B, \mathsf{pk}, \mathsf{sk}(B)$;
Actions :

$$
\begin{array}{llll}
A & \rightarrow & B & : \{NA, A\}_{\mathsf{pk}(B)} \\
B & \rightarrow & A & : \{NA, NB\}_{\mathsf{pk}(A)} \\
A & \rightarrow & B & : \{NB\}_{\mathsf{pk}(B)}
\end{array}
$$

Initial knowledge of each role uses terms of type Agent.
Other variables (like NA and NB): values freshly created by the agent who first uses it.

Message Sequence Charts

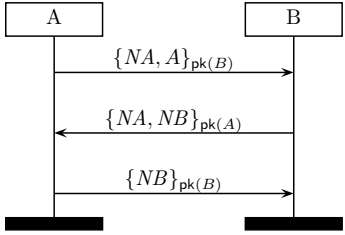
msc NSPK



Towards the Meaning of an A&B Specification

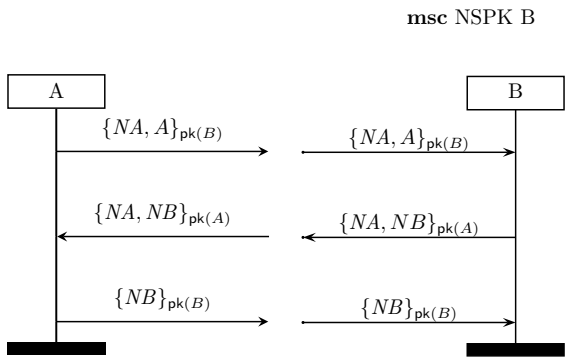
Split a message sequence chart into individual roles

msc NSPK



Towards the Meaning of an A&B Specification

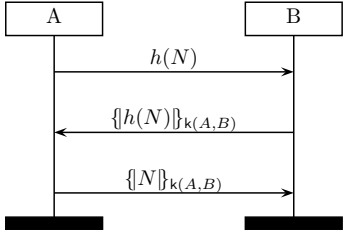
Split a message sequence chart into individual roles



Towards the Meaning of an A&B Specification

Nontrivial for some protocols:

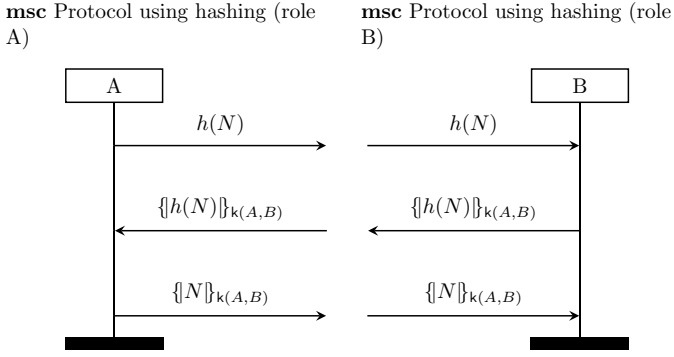
msc Protocol using hashing



Here, $\mathsf{k}(A, B)$ is a shared key of A and B and N is fresh.

Towards the Meaning of an A&B Specification

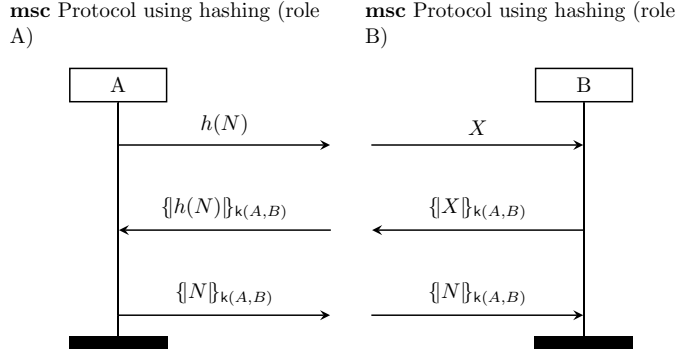
Nontrivial for some protocols:



This is wrong: B cannot check the format of the first message... before receiving the third!

Towards the Meaning of an A&B Specification

Nontrivial for some protocols, better view:



What remains to be done: B must check that $X = h(N)$ after receiving third message.

Technical Background: Multisets

Definition (Multiset)

A multiset m over a set X is a set of elements, each imbued with a multiplicity, i.e., $m\colon X \to \mathbb{N}$, where $m(x)$ denotes the multiplicity of x.

- We use $\subseteq^\sharp$ for multiset inclusion, $\cup^\sharp$ for multiset union, and $\setminus^\sharp$ for multiset difference. (Exercise: define these formally).
- We denote by $X^\sharp$ the set of finite multisets with elements from X.

Technical Background: Multisets

Definition (Multiset)

A multiset m over a set X is a set of elements, each imbued with a multiplicity, i.e., $m\colon X \to \mathbb{N}$, where $m(x)$ denotes the multiplicity of x.

- We use $\subseteq^\sharp$ for multiset inclusion, $\cup^\sharp$ for multiset union, and $\setminus^\sharp$ for multiset difference. (Exercise: define these formally).
- We denote by $X^\sharp$ the set of finite multisets with elements from X.

Example (Multisets)

Let $A = [0 \mapsto 1, 1 \mapsto 3, 2 \mapsto 2] = [0, 1, 1, 1, 2, 2]$
and $B = [0 \mapsto 2, 1 \mapsto 1, 2 \mapsto 1] = [0, 0, 1, 2]$. Then

- $A \cup^\sharp B = [0 \mapsto 3, 1 \mapsto 4, 2 \mapsto 3] = [0, 0, 0, 1, 1, 1, 1, 2, 2, 2]$.
- $A \setminus^\sharp B = [1 \mapsto 2, 2 \mapsto 1] = [1, 1, 2]$.

- Instead of expliciting stating a multiplicity, we may simply write elements multiple times.

Technical Background: Multiset Rewriting (MSR)

Definition (Facts)

We assume a set of untyped fact symbols, written Σ_{fact}, each with an arity $k \geq 0$. Then

$$\mathsf{F}(t_1, \ldots, t_k)$$

for $\mathsf{F} \in \Sigma_{fact}$ with arity k and $t_1, \ldots, t_k \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{N})$ is called a fact.
Here $\mathcal{X}$ and $\mathcal{N}$ are sets of variables and names, e.g., for agents or nonces.

Technical Background: Multiset Rewriting (MSR)

Definition (Facts)

We assume a set of untyped fact symbols, written Σ_{fact}, each with an arity $k \geq 0$. Then

$$F(t_1, \ldots, t_k)$$

for $F \in \Sigma_{fact}$ with arity k and $t_1, \ldots, t_k \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{N})$ is called a fact.
Here $\mathcal{X}$ and $\mathcal{N}$ are sets of variables and names, e.g., for agents or nonces.

Definition (Labeled multiset rewriting)

- A labeled multiset rewriting rule is a triple written

$$l \xrightarrow{a} r$$

  where l and r are multisets of facts, called state facts and a is a multiset of facts, called action facts or events.
- A labeled multiset rewrite system is a set of multiset rewriting rules.

Modeling Security Protocols Using MSR

We have different kinds of rules in our protocol models:

Adversary rules Determine which messages the adversary can derive from his knowledge.
The adversary fact $\mathrm{K}(t)$ represents the knowledge of a term t.
Can be augmented by **equations** that express how functions are computed.

Example (Adversary deduction)

Standard symmetric encryption and decryption is defined by

$$\mathit{decrypt}(\mathit{encrypt}(m, k), k) = m$$

so given $\mathrm{K}(\mathit{encrypt}(\mathit{msg}, \mathit{key}))$ and $\mathrm{K}(\mathit{key})$, the adversary can derive $\mathrm{K}(\mathit{msg})$ by applying *decrypt* to the two known terms.

Multiset rewrite rules are applied modulo the given equations.

Modeling Security Protocols Using MSR (cont.)

Protocol rules Formalize the roles of the given protocol. These define the sending and receiving of messages and use agent state facts to track each roles' progress.

Modeling Security Protocols Using MSR (cont.)

Protocol rules Formalize the roles of the given protocol. These define the sending and receiving of messages and use agent state facts to track each roles' progress.

Infrastructure rules Formalize the generation of cryptographic keys, e.g., to model a public-key infrastructure (PKI).

Modeling Security Protocols Using MSR (cont.)

Protocol rules Formalize the roles of the given protocol. These define the sending and receiving of messages and use agent state facts to track each roles' progress.

Infrastructure rules Formalize the generation of cryptographic keys, e.g., to model a public-key infrastructure (PKI).

Fresh rule Generates unique fresh values X marked as fresh facts $\mathrm{Fr}(X)$. These can be used as nonces or thread identifiers.

Adversary Rules

Definition (Adversary knowledge derivation)

The adversary can use the following inference rules:

$$[\mathsf{K}(x)] \xrightarrow{\mathsf{K}(x)} [\mathsf{In}(x)] \qquad [\mathsf{Out}(x)] \to [\mathsf{K}(x)] \qquad [\mathsf{Fr}(x)] \to [\mathsf{K}(x)]$$

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

- The first two rules model the interface with the protocol:
  - ★ Out facts mark messages sent by the protocol and
  - ★ In facts mark messages to be received by the protocol.

Adversary Rules

Definition (Adversary knowledge derivation)

The adversary can use the following inference rules:

$$[\mathsf{K}(x)] \xrightarrow{\mathsf{K}(x)} [\mathsf{In}(x)] \qquad [\mathsf{Out}(x)] \to [\mathsf{K}(x)] \qquad [\mathsf{Fr}(x)] \to [\mathsf{K}(x)]$$

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

- The first two rules model the interface with the protocol:
  - ★ Out facts mark messages sent by the protocol and
  - ★ In facts mark messages to be received by the protocol.
- The third rule models that the adversary can generate fresh values.

Adversary Rules

Definition (Adversary knowledge derivation)

The adversary can use the following inference rules:

$$[\mathsf{K}(x)] \xrightarrow{\mathsf{K}(x)} [\mathsf{In}(x)] \qquad [\mathsf{Out}(x)] \to [\mathsf{K}(x)] \qquad [\mathsf{Fr}(x)] \to [\mathsf{K}(x)]$$

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

- The first two rules model the interface with the protocol:
  - ★ Out facts mark messages sent by the protocol and
  - ★ In facts mark messages to be received by the protocol.
- The third rule models that the adversary can generate fresh values.
- The final rule models the adversary's inference capabilities.
  Note that all derivations are modulo the underlying equational theory.

Adversary Rules

Definition (Adversary knowledge derivation)

The adversary can use the following inference rules:

$$[\mathsf{K}(x)] \xrightarrow{\mathsf{K}(x)} [\mathsf{In}(x)] \qquad [\mathsf{Out}(x)] \to [\mathsf{K}(x)] \qquad [\mathsf{Fr}(x)] \to [\mathsf{K}(x)]$$

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

- The first two rules model the interface with the protocol:
  - ★ Out facts mark messages sent by the protocol and
  - ★ In facts mark messages to be received by the protocol.
- The third rule models that the adversary can generate fresh values.
- The final rule models the adversary's inference capabilities.
  Note that all derivations are modulo the underlying equational theory.

A protocol model may include additional adversary rules, e.g., for compromising other agents by learning their long-term keys.

Fresh Rule

Definition (Fresh and public values)

Let FV and PV be two countably infinite and disjoint sets of fresh values and public values, s.t. $FV \cup PV \subseteq \mathcal{N}$. Rules defined using terms in $\mathcal{T}_\Sigma(\mathcal{X}, \mathcal{N})$.

Definition (Fresh rule)

We define a special rule for the creation of fresh values. This rule has no precondition and it is the only rule allowed to produce such facts:

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

Our semantics will ensure that each created nonce N is unique.

Protocol Rules

- A protocol consists of a set of roles.
- Each role consists of a set of protocol rules, specifying the sending and receiving of messages, and the use of fresh values.
- The roles use agent state facts to track their progress.

Protocol Rules

- A protocol consists of a set of roles.
- Each role consists of a set of protocol rules, specifying the sending and receiving of messages, and the use of fresh values.
- The roles use agent state facts to track their progress.

Definition (Agent state fact)

An agent state fact for role R is a fact

$$\text{St\_R\_s}(A, id, k_1, \ldots, k_n)$$

where $\text{St\_R\_s} \in \Sigma_{fact}$ and

- $s \in \mathbb{N}$ is the number of the protocol step within the role,
- A is the name of the agent executing the role,
- id the thread identifier for this instantiation of role R, and
- $k_i \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{N})$ are terms in the agent's knowledge.

Communication

- Messages are sent and received via Out and In facts.
- Any rule with such a fact may also have a matching Send and Recv action. This is convenient for reasoning.

Example (Rule examples)

Receive rule example:

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

Send rule example:

$$[\text{St\_I\_3}(A, id, k, m)] \xrightarrow{\text{Send}(A,\{m\}_k)} [\text{St\_I\_4}(A, id, k, m), \text{Out}(\{m\}_k)]$$

Infrastructure Rules

Definition (Key generation for a PKI)

Generate (long-term) private and public keys.

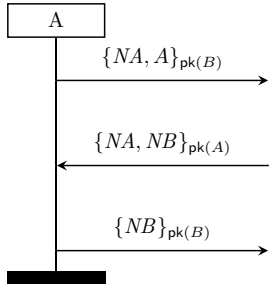
$$[\,\mathrm{Fr}(sk)\,] \rightarrow [\,\mathrm{Ltk}(A, sk), \mathrm{Pk}(A, pk(sk)), \mathrm{Out}(pk(sk))\,]$$

- The premise requires a fresh value sk.
- The fact $\mathrm{Ltk}(A, sk)$ records sk as A's private key.
- The fact $\mathrm{Pk}(A, pk(sk))$ records $pk(sk)$ as A's public key.
- The fact $\mathrm{Out}(pk(sk))$ publishes the public key $pk(sk)$.

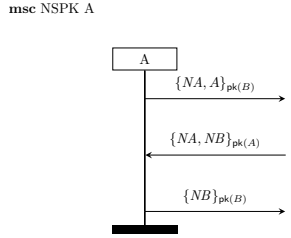
Role Syntax

Graphical:

msc NSPK A

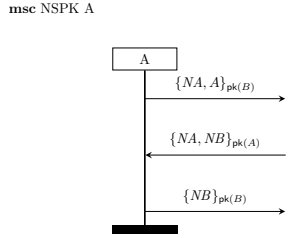


Role Specification Rules



$$[\mathsf{St\_A\_1}(A, tid, skA, B, pkB),\ \ \mathsf{Fr}(NA)] \xrightarrow{\mathsf{Send}(A,\{NA,A\}_{pkB})} [\mathsf{St\_A\_2}(A, tid, skA, B, pkB, NA),\ \ \mathsf{Out}(\{NA, A\}_{pkB})]$$

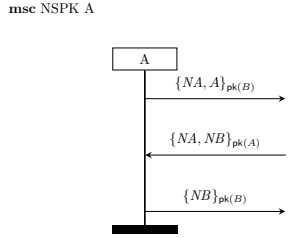
Role Specification Rules



$$[\mathsf{St\_A\_1}(A, tid, skA, B, pkB),\ \ \mathsf{Fr}(NA)] \xrightarrow{\mathsf{Send}(A, \{NA, A\}_{pkB})} [\mathsf{St\_A\_2}(A, tid, skA, B, pkB, NA),\ \ \mathsf{Out}(\{NA, A\}_{pkB})]$$

$$[\mathsf{St\_A\_2}(A, tid, skA, B, pkB, NA),\ \ \mathsf{In}(\{NA, NB\}_{pk(skA)})] \xrightarrow{\mathsf{Recv}(A, \{NA, NB\}_{pk(skA)})} [\mathsf{St\_A\_3}(A, tid, skA, B, pkB, NA, NB)]$$

Role Specification Rules

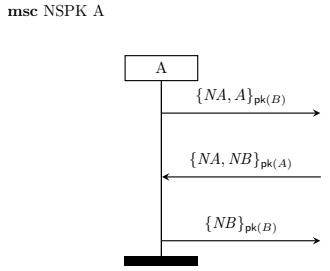


$$[\mathsf{St\_A\_1}(A, tid, skA, B, pkB),\ \ \mathsf{Fr}(NA)] \xrightarrow{\mathsf{Send}(A,\{NA,A\}_{pkB})} [\mathsf{St\_A\_2}(A, tid, skA, B, pkB, NA),\ \ \mathsf{Out}(\{NA, A\}_{pkB})]$$

$$[\mathsf{St\_A\_2}(A, tid, skA, B, pkB, NA),\ \ \mathsf{In}(\{NA, NB\}_{pk(skA)})] \xrightarrow{\mathsf{Recv}(A,\{NA,NB\}_{pk(skA)})} [\mathsf{St\_A\_3}(A, tid, skA, B, pkB, NA, NB)]$$

$$[\mathsf{St\_A\_3}(A, tid, skA, B, pkB, NA, NB)] \xrightarrow{\mathsf{Send}(A,\{NB\}_{pkB})} [\mathsf{St\_A\_4}(A, tid, skA, B, pkB, NA, NB),\ \ \mathsf{Out}(\{NB\}_{pkB})]$$

Initialization of Threads



For each role R there must be an initialization rule which creates a thread with a fresh identifier id, playing role R and owned by agent A:

$$[\,\mathsf{Fr}(id), \mathsf{Ltk}(A, skA), \mathsf{Pk}(B, pkB)\,] \xrightarrow{\mathsf{Create\_R}(A,id)} [\,\mathsf{St\_R\_1}(A, id, skA, B, pkB), \mathsf{Ltk}(A, skA), \mathsf{Pk}(B, pkB)\,]$$

The thread's knowledge is also initialized, here with A's private key skA and B's name and public key pkB.

Ground Instances

Definition (Ground instances)

- An instance of an object X (such as a term, fact, or rewrite rule) is the result of applying a substitution σ to all terms in X, written $X\sigma$.
- A ground instance of X is one where all the resulting terms have no variables, i.e., are ground.

Ground Instances

Definition (Ground instances)

- An instance of an object X (such as a term, fact, or rewrite rule) is the result of applying a substitution σ to all terms in X, written $X\sigma$.
- A ground instance of X is one where all the resulting terms have no variables, i.e., are ground.
- For a multiset rewrite system R (i.e., a set of MSR rules), we use $\mathit{ginsts}(R)$ to denote the set of all ground instances of rules in R.

Ground Instances

Definition (Ground instances)

- An instance of an object X (such as a term, fact, or rewrite rule) is the result of applying a substitution σ to all terms in X, written $X\sigma$.
- A ground instance of X is one where all the resulting terms have no variables, i.e., are ground.
- For a multiset rewrite system R (i.e., a set of MSR rules), we use $\mathit{ginsts}(R)$ to denote the set of all ground instances of rules in R.
- We denote by $\mathcal{G}$ the set of ground facts. Recall that $\mathcal{G}^\sharp$ then denotes the set of finite multisets of elements from $\mathcal{G}$.

State

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

State

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

Definition (State)

A state is a finite multiset of ground facts (i.e., an element of $\mathcal{G}^\sharp$).

Example (State)

$$[\text{St\_R\_1}(alice, 17, k_1, k_2), \text{Out}(k_1), \text{Out}(k_2), \text{Out}(k_2), \text{K}(m)]$$

Labeled Multiset Rewriting

Case for Linear Facts

Definition (Labeled multiset rewriting step)

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

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

Labeled Multiset Rewriting

Case for Linear Facts

Definition (Labeled multiset rewriting step)

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

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

- We distinguish between linear and persistent facts.
  (In Tamarin, the latters' names start with an exclamation mark.)
- The rule above is for linear facts, which are "consumed" by the rewriting step, i.e., removed from the state.

Labeled Multiset Rewriting

Case for Linear Facts

Definition (Labeled multiset rewriting step)

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

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

- We distinguish between linear and persistent facts.
  (In Tamarin, the latters' names start with an exclamation mark.)
- The rule above is for linear facts, which are "consumed" by the rewriting step, i.e., removed from the state.
- Persistent facts are not consumed by rewriting steps and can be reused arbitrarily often. See appendix slide for details.

Rewriting Step Example #1

Example (Rewriting step with linear facts)

Using the rewrite rule

$$[\,\mathsf{St\_I\_1}(A, ID, K), \mathsf{Fr}(N)\,] \xrightarrow{\mathsf{Send}(A, \{|N|\}_K)} [\,\mathsf{St\_I\_2}(A, ID, K, N), \mathsf{Out}(\{|N|\}_K)\,]$$

we rewrite the state

$$S = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_1}(alice, 17, k), \mathsf{Fr}(n), \mathsf{K}(m)\,]$$

Rewriting Step Example #1

Example (Rewriting step with linear facts)

Using the rewrite rule

$$[\,\mathsf{St\_I\_1}(A, ID, K), \mathsf{Fr}(N)\,] \xrightarrow{\mathsf{Send}(A, \{|N|\}_K)} [\,\mathsf{St\_I\_2}(A, ID, K, N), \mathsf{Out}(\{|N|\}_K)\,]$$

we rewrite the state

$$S = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_1}(alice, 17, k), \mathsf{Fr}(n), \mathsf{K}(m)\,]$$

to the successor state

$$S' = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_2}(alice, 17, k, n), \mathsf{Out}(\{|n|\}_k), \mathsf{K}(m)\,].$$

The matching substitution instantiating the rule is

$$\sigma = [A \mapsto alice, ID \mapsto 17, K \mapsto k, N \mapsto n]$$

Rewriting Step Example #1

Example (Rewriting step with linear facts)

Using the rewrite rule

$$[\,\mathsf{St\_I\_1}(A, ID, K), \mathsf{Fr}(N)\,] \xrightarrow{\mathsf{Send}(A, \{|N|\}_K)} [\,\mathsf{St\_I\_2}(A, ID, K, N), \mathsf{Out}(\{|N|\}_K)\,]$$

we rewrite the state

$$S = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_1}(alice, 17, k), \mathsf{Fr}(n), \mathsf{K}(m)\,]$$

to the successor state

$$S' = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_2}(alice, 17, k, n), \mathsf{Out}(\{|n|\}_k), \mathsf{K}(m)\,].$$

The matching substitution instantiating the rule is

$$\sigma = [A \mapsto alice, ID \mapsto 17, K \mapsto k, N \mapsto n]$$

Note: The linear agent-state and Fr facts are consumed.

Rewriting Step Example #2

Example (Rewriting step with persistent facts)

Using the adversary rewrite rule

$$[\,\mathsf{K}(X), \mathsf{K}(Y)\,] \to [\,\mathsf{K}(\langle X, Y\rangle)\,]$$

we rewrite the state

$$S' = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_2}(\mathit{alice}, 17, k, n), \mathsf{Out}(\{|n|\}_k), \mathsf{K}(m)\,].$$

Rewriting Step Example #2

Example (Rewriting step with persistent facts)

Using the adversary rewrite rule

$$[\,\mathsf{K}(X), \mathsf{K}(Y)\,] \to [\,\mathsf{K}(\langle X, Y\rangle)\,]$$

we rewrite the state

$$S' = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_2}(\mathit{alice}, 17, k, n), \mathsf{Out}(\{|n|\}_k), \mathsf{K}(m)\,].$$

to the successor state

$$S'' = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_2}(\mathit{alice}, 17, k, n), \mathsf{Out}(\{|n|\}_k), \mathsf{K}(m), \mathsf{K}(\langle m, m\rangle)\,].$$

The matching substitution instantiating the rule is

$$\sigma = [X \mapsto m, Y \mapsto m].$$

Rewriting Step Example #2

Example (Rewriting step with persistent facts)

Using the adversary rewrite rule

$$[\,\mathsf{K}(X), \mathsf{K}(Y)\,] \to [\,\mathsf{K}(\langle X, Y\rangle)\,]$$

we rewrite the state

$$S' = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_2}(\mathit{alice}, 17, k, n), \mathsf{Out}(\{|n|\}_k), \mathsf{K}(m)\,].$$

to the successor state

$$S'' = [\,\mathsf{Fr}(n'), \mathsf{St\_I\_2}(\mathit{alice}, 17, k, n), \mathsf{Out}(\{|n|\}_k), \mathsf{K}(m), \mathsf{K}(\langle m, m\rangle)\,].$$

The matching substitution instantiating the rule is

$$\sigma = [X \mapsto m, Y \mapsto m].$$

Note: The persistent fact $\mathsf{K}(m)$ can be used twice and it remains in state.

Executions

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

Executions

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 such that

1. The initial state is empty:

$$S_0 = \emptyset^\sharp$$

Executions

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 such that

1. The initial state is empty:
$$S_0 = \emptyset^\sharp$$
2. Corresponds to a transition sequence, i.e., for all i:
$$(S_{i-1},\ l_i \xrightarrow{a_i} r_i,\ S_i) \in \mathit{steps}(R)$$

Executions

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 such that

1. The initial state is empty:
$$S_0 = \emptyset^\sharp$$
2. Corresponds to a transition sequence, i.e., for all i:
$$(S_{i-1},\ l_i \xrightarrow{a_i} r_i,\ S_i) \in \mathit{steps}(R)$$
3. Fresh names are unique, i.e., for all n, i, and j:
$$(l_i \xrightarrow{a_i} r_i) = (l_j \xrightarrow{a_j} r_j) = ([] \to [\mathrm{Fr}(n)]) \implies i = j.$$

Trace

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.,

$$a_1, a_2, \ldots, a_k$$

Trace

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.,

$$a_1, a_2, \ldots, a_k$$

We will express the protocols' security properties in a first-order language interpreted over traces.

Claim Events

Properties of claim events

- Claim events are part of the protocol rules as actions.
- Their only effect is to record facts or claims in the protocol trace.
- They cannot be observed, modified, or generated by the adversary.

Claim Events

Properties of claim events

- Claim events are part of the protocol rules as actions.
- Their only effect is to record facts or claims in the protocol trace.
- They cannot be observed, modified, or generated by the adversary.

Expressing properties using claim events

- Properties of traces *tr* are expressed using claim events and other events (e.g., adversary knowledge K) occuring in *tr*.
- Properties may be formulated from the point of view of a given role *R* by including the role identifier *R* in the claim, thus yielding guarantees for that role.
- We illustrate with secrecy and variants of authentication, though the approach is more general.

Security Property Specification

Definition (Frequently used events)

$\mathsf{Send}(A, t), \mathsf{Recv}(A, t), \mathsf{Create\_R}(A, id)$	protocol events
$\mathsf{Claim\_}\texttt{claimtype}(A, t)$	claim event
$\mathsf{Honest}(A), \mathsf{Rev}(A)$	honesty and reveal events
$\mathsf{K}(t)$	adversary knowledge

Security Property Specification

Definition (Frequently used events)

$\mathsf{Send}(A,t), \mathsf{Recv}(A,t), \mathsf{Create\_R}(A,id)$	protocol events
$\mathsf{Claim\_claimtype}(A,t)$	claim event
$\mathsf{Honest}(A), \mathsf{Rev}(A)$	honesty and reveal events
$\mathsf{K}(t)$	adversary knowledge

Definition (Property specification language)

We formulate security properties in first-order logic over the predicates

$\mathsf{F}@i$	timestamped event
$t = u$	term equality
$i = j$	timepoint equality
$i < j$	timepoint inequality

The predicate $\mathsf{F}@i$ holds on trace $tr = a_1, \ldots, a_n$ if $\mathsf{F} \in a_i$.

Role Instrumentation for Secrecy

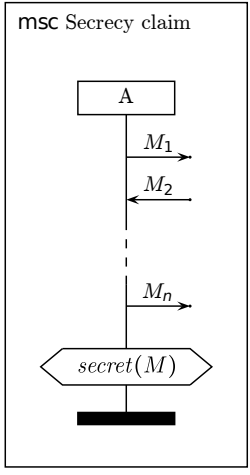
Definition (Secrecy, informally)

The intruder cannot discover the data (e.g., key, nonce, etc.) that is intended to be secret.

Role instrumentation

- Insert the claim event Claim_secret(A, M) into role A to claim that the message M used in the role remains secret.
- Position: At the role's end.
- **Example:** in NSPK, nonces *NA* and *NB* should remain secret.

Note: In the graphs, where the executing role is clear from the context, we abbreviate Claim_claimtype(A, t) to *claimtype*(t) inside a hexagon.



Formalization of Secrecy

Definition (Secrecy, first attempt)

The secrecy property consists of all traces tr satisfying

$$\forall A, M, i.\ \text{Claim\_}\texttt{secret}(A, M)@i \Rightarrow \neg(\exists j.\ \mathsf{K}(M)@j)$$

Formalization of Secrecy

Definition (Secrecy, first attempt)

The secrecy property consists of all traces tr satisfying

$$\forall A, M, i.\ \text{Claim\_}\texttt{secret}(A, M)@i \Rightarrow \neg(\exists j.\ \mathsf{K}(M)@j)$$

- Can only require M to remain secret if A is honest and runs the protocol with another honest agent, i.e.,
- Trivially broken whenever A or B is instantiated with a compromised agent, since then the adversary will of course know M.

Formalization of Secrecy

Definition (Secrecy, first attempt)

The secrecy property consists of all traces tr satisfying

$$\forall A, M, i.\ \text{Claim\_}\texttt{secret}(A, M)@i \Rightarrow \neg(\exists j.\ \mathsf{K}(M)@j)$$

- Can only require M to remain secret if A is honest and runs the protocol with another honest agent, i.e.,
- Trivially broken whenever A or B is instantiated with a compromised agent, since then the adversary will of course know M.
- This definition is fine for a passive adversary, who only observes network traffic, but does not act as a protocol participant.

Compromised Agent

Definition (Compromised Agent)

A compromised agent is under full adversary control, i.e., sharing its long-term secrets with the adversary. We model this using the rewrite rule

$$[\,\mathsf{Ltk}(A, skA)\,] \xrightarrow{\mathsf{Rev}(A)} [\,\mathsf{Ltk}(A, skA), \mathsf{Out}(skA)\,].$$

Compromised Agent

Definition (Compromised Agent)

A compromised agent is under full adversary control, i.e., sharing its long-term secrets with the adversary. We model this using the rewrite rule

$$[\,\mathrm{Ltk}(A, skA)\,] \xrightarrow{\mathrm{Rev}(A)} [\,\mathrm{Ltk}(A, skA), \mathrm{Out}(skA)\,].$$

- The Out state fact publishes A's private key sk.
- The Rev event in the trace can be used in property specifications.

Compromised Agent

Definition (Compromised Agent)

A compromised agent is under full adversary control, i.e., sharing its long-term secrets with the adversary. We model this using the rewrite rule

$$[\,\mathrm{Ltk}(A, skA)\,] \xrightarrow{\mathrm{Rev}(A)} [\,\mathrm{Ltk}(A, skA), \mathrm{Out}(skA)\,].$$

- The Out state fact publishes A's private key sk.
- The Rev event in the trace can be used in property specifications.

Given an agent's private key, the adversary can perform all the agent's send and receive steps. (Requires certain sanity conditions on the rules defining roles, which are omitted for conciseness.)

Formalization of Secrecy

Definition (Honesty)

The action label Honest(A) indicates that the agent A is presumed to be honest, i.e., not sharing information with the attacker. An agent is honest in a trace tr when $\mathrm{Rev}(A) \notin tr$.

Formalization of Secrecy

Definition (Honesty)

The action label Honest(A) indicates that the agent A is presumed to be honest, i.e., not sharing information with the attacker. An agent is honest in a trace tr when $\mathrm{Rev}(A) \notin tr$.

Rule instrumentation with honesty events:

- A claim event in a rule's action label must be accompanied by an Honest(B) event for all parties B that are expected to be honest.
- Usually, these are the owner and intended communication partners of thread making the claim.

Formalization of Secrecy

Definition (Honesty)

The action label Honest(A) indicates that the agent A is presumed to be honest, i.e., not sharing information with the attacker. An agent is honest in a trace tr when $\mathrm{Rev}(A) \notin tr$.

Rule instrumentation with honesty events:

- A claim event in a rule's action label must be accompanied by an Honest(B) event for all parties B that are expected to be honest.
- Usually, these are the owner and intended communication partners of thread making the claim.

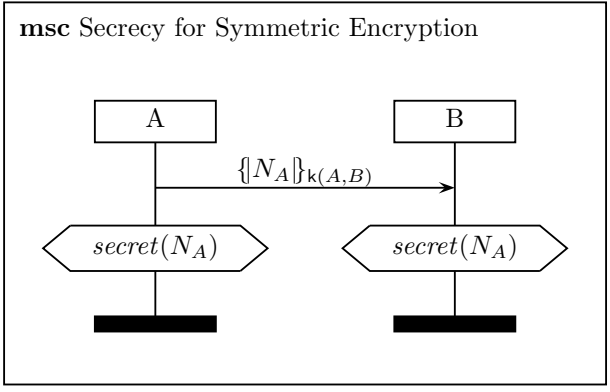
Definition (Secrecy)

The secrecy property consists of all traces tr satisfying

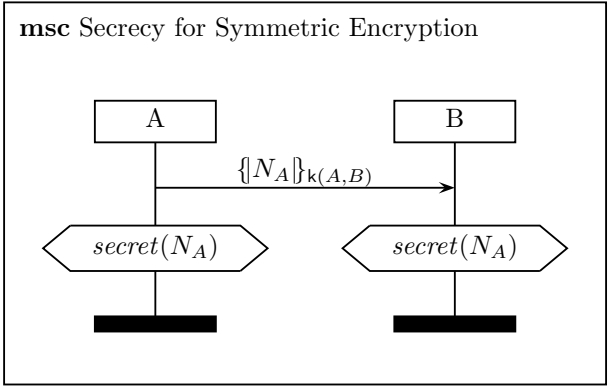
$$\forall A\ M\ i.\ (\mathrm{Claim\_secret}(A, M)@i) \Rightarrow \neg(\exists j.\ \mathrm{K}(M)@j) \vee (\exists B\ k.\ \mathrm{Rev}(B)@k \wedge \mathrm{Honest}(B)@i)$$

This definition guarantees the secrecy of M unless the adversary has compromised some agent that is required to be honest.

Secrecy Example #1

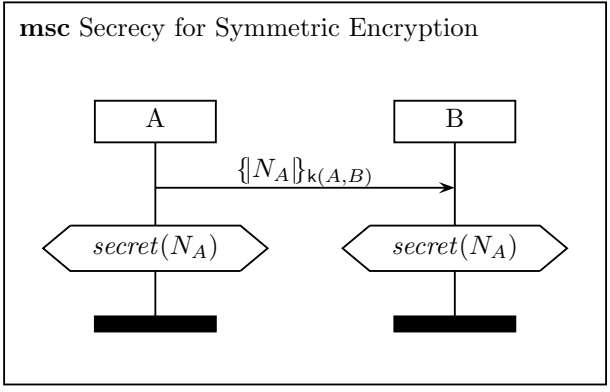


Secrecy Example #1



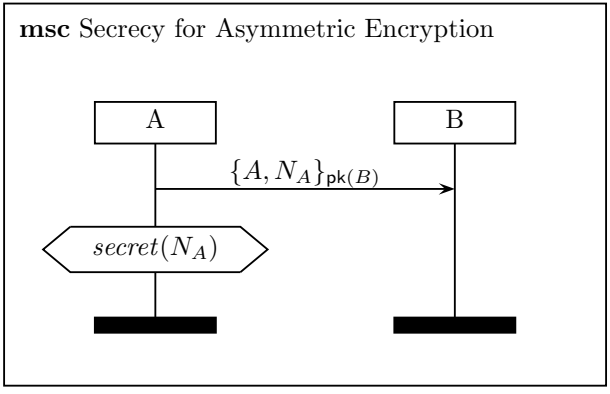
- This is fine: secrecy holds for both A and B.

Secrecy Example #1

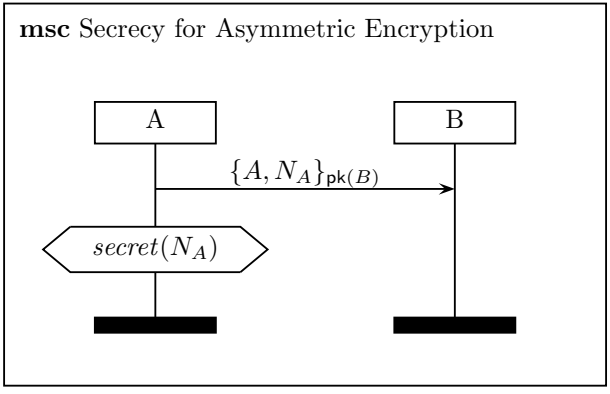


- This is fine: secrecy holds for both A and B.
- We omit the obvious annotations $\mathsf{Honest}(A), \mathsf{Honest}(B)$ in message sequence charts for 2-party protocols.

Secrecy Example #2

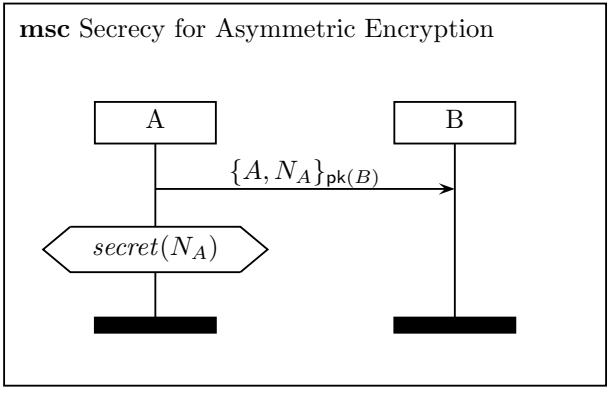


Secrecy Example #2



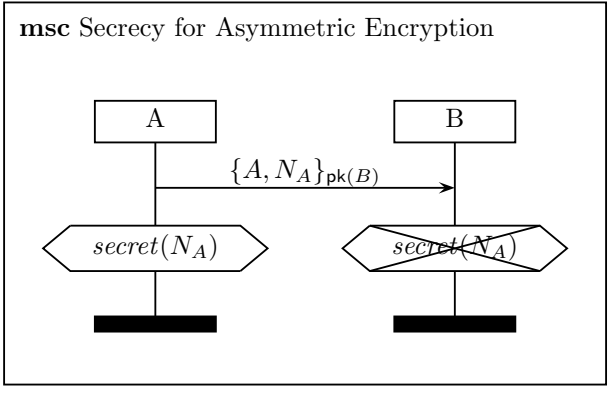
- Secrecy holds for A: she knows that only B can decrypt message.

Secrecy Example #2



- Secrecy holds for A: she knows that only B can decrypt message.
- Does it hold for B as well?

Secrecy Example #2



- Secrecy fails for B: he does not know who encrypted the message!

Authentication

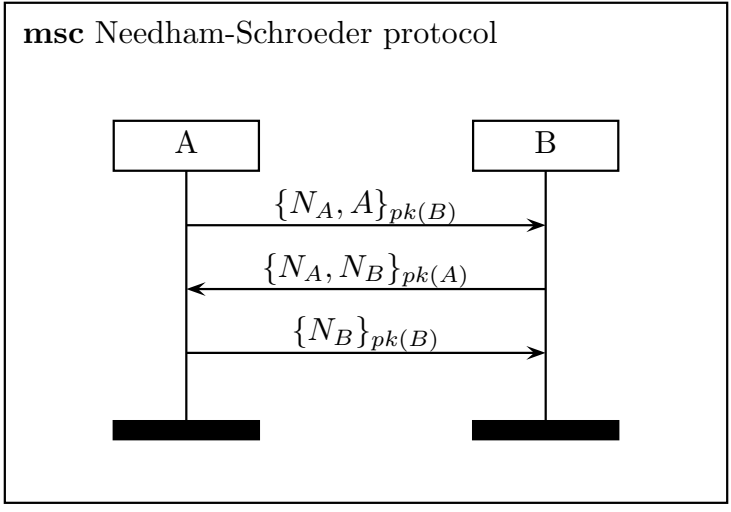
Which authentication are you talking about?

- No unique definition of authentication, but many different forms.
- Considerable effort has been devoted to specifying and classifying, semi-formally or formally, different forms of authentication.

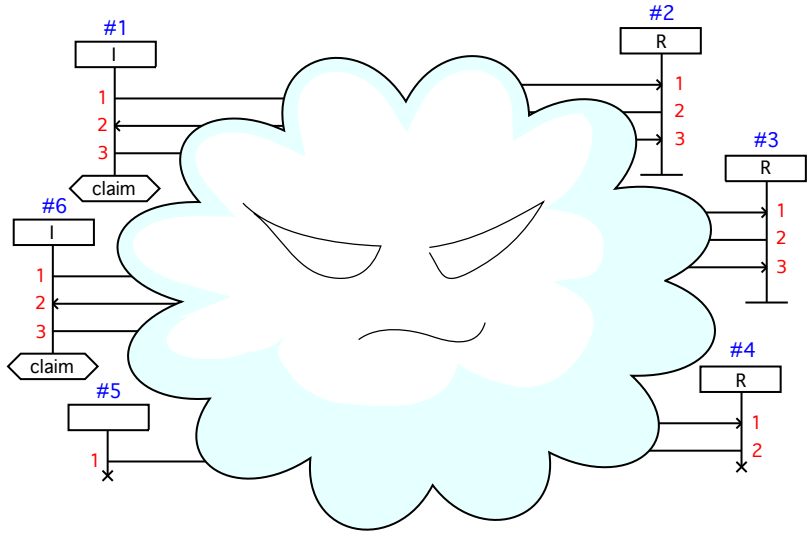
Examples

- aliveness, weak agreement, non-injective agreement, injective agreement, weak and strong authentication, synchronization, and matching histories.

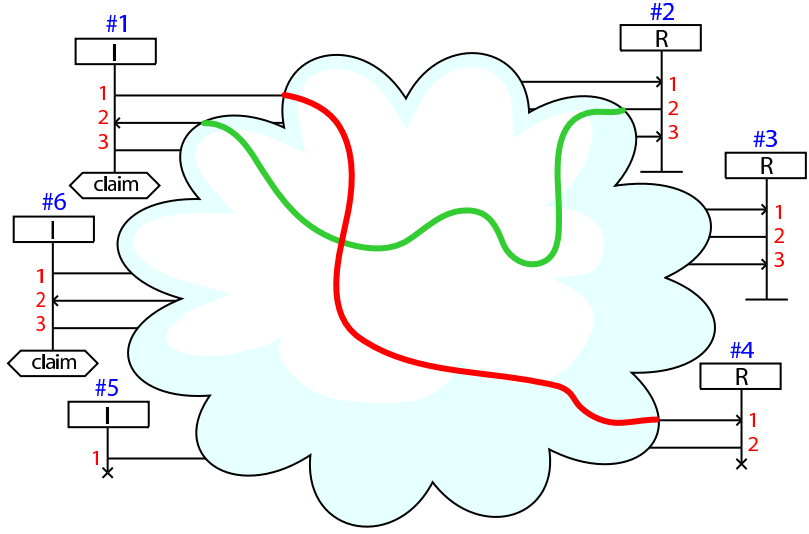
A Perfect (Picture of the) World



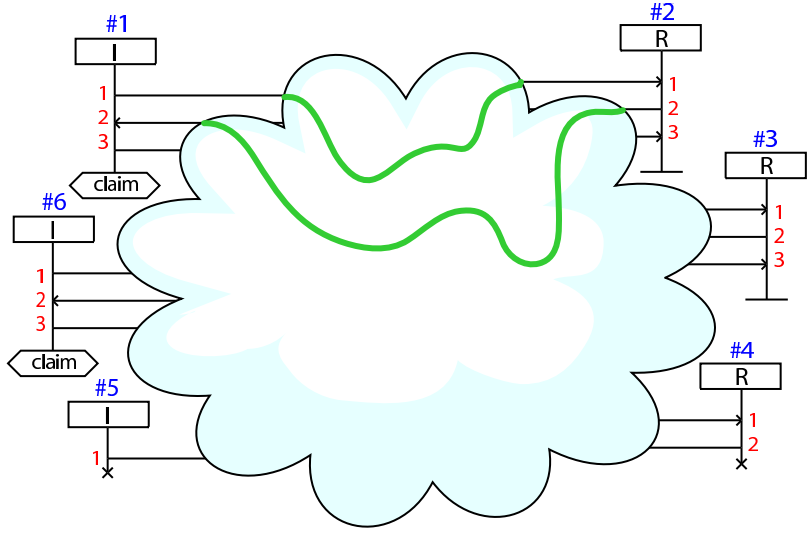
A More Realistic Picture



Failed Authentication



Successful Authentication



A Hierarchy of Authentication Specifications (1)

[Gavin Lowe, 1997]

Gavin Lowe has defined the following hierarchy of increasingly stronger authentication properties[1]:

Aliveness A protocol guarantees to an agent a in role A aliveness of another agent b if, whenever a completes a run of the protocol, apparently with b in role B, then b has previously been running the protocol.

Weak agreement A protocol guarantees to an agent a in role A weak agreement with another agent b if, whenever agent a completes a run of the protocol, apparently with b in role B, then b has previously been running the protocol, apparently with a.

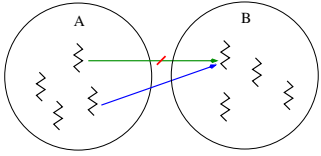
[1] Terminology and notation slightly adapted to our setting.

A Hierarchy of Authentication Specifications (2)

[Gavin Lowe, 1997]

Non-injective agreement A protocol guarantees to an agent a in role A non-injective agreement with an agent b in role B on a message M if, whenever a completes a run of the protocol, apparently with b in role B, then b has previously been running the protocol, apparently with a, and b was acting in role B in his run, and the two principals agreed on the message M.

Injective agreement is non-injective agreement where additionally each run of agent a in role A corresponds to a unique run of agent b.



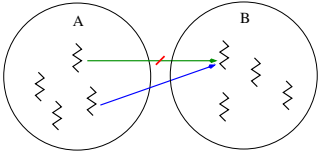
Other versions including recentness: insist that B's run was recent (e.g., within t time units).

A Hierarchy of Authentication Specifications (2)

[Gavin Lowe, 1997]

Non-injective agreement A protocol guarantees to an agent a in role A non-injective agreement with an agent b in role B on a message M if, whenever a completes a run of the protocol, apparently with b in role B, then b has previously been running the protocol, apparently with a, and b was acting in role B in his run, and the two principals agreed on the message M.

Injective agreement is non-injective agreement where additionally each run of agent a in role A corresponds to a unique run of agent b.



Other versions including recentness: insist that B's run was recent (e.g., within t time units).

These are complex properties. What do they mean and can we formalize them?

Role Instrumentation for Authentication

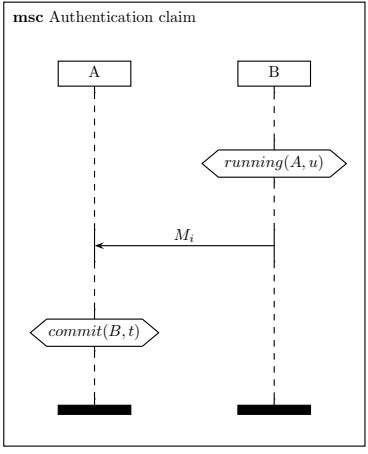
We use two claims to express that role A authenticates role B on t:

In role A:

- Insert a commit claim event Claim_commit(A, B, t).
- Position: after A can construct t. Typically, at end of A's role.

In role B:

- Insert a running claim event Claim_running(B, A, u).
- Term u is B's view of t.
- Position: after B can construct u and causally preceding Claim_commit(A, B, t).



Formalizing Authentication

Definition (Non-injective agreement)

The property $Agreement_{NI}(R_1, R_2, t)$ consists of all traces satisfying

$$\begin{aligned} \forall a\ b\ t\ i.\quad & \mathrm{Claim\_commit}(a, b, \langle R_1, R_2, t\rangle)@i \\ & \Rightarrow (\exists j.\mathrm{Claim\_running}(b, a, \langle R_1, R_2, t\rangle)@j) \\ & \quad \vee (\exists X\ r.\mathrm{Rev}(X)@r \wedge \mathrm{Honest}(X)@i) \end{aligned}$$

- If a commit claim is made with honest agents a and b, then the peer b must be running with the same parameter t, or the adversary has compromised at least one of the two agents.

Formalizing Authentication

Definition (Non-injective agreement)

The property $Agreement_{NI}(R_1, R_2, t)$ consists of all traces satisfying

$$\begin{aligned} \forall a\ b\ t\ i.\quad & \mathtt{Claim\_commit}(a, b, \langle R_1, R_2, t\rangle)@i \\ & \Rightarrow (\exists j.\mathtt{Claim\_running}(b, a, \langle R_1, R_2, t\rangle)@j) \\ & \quad \vee (\exists X\ r.\mathrm{Rev}(X)@r \wedge \mathrm{Honest}(X)@i) \end{aligned}$$

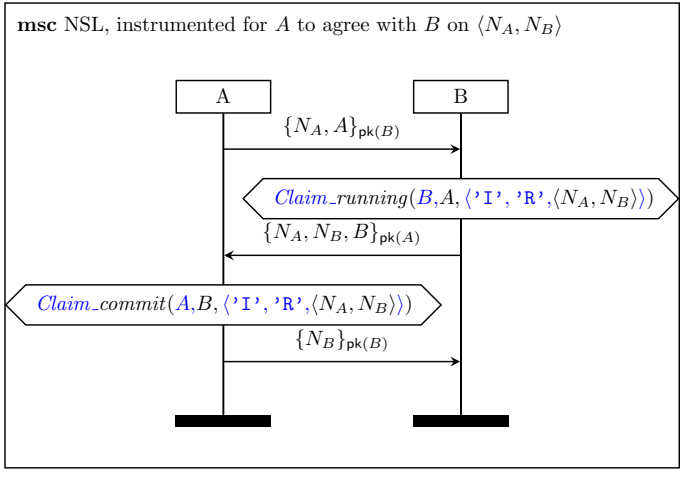
- If a commit claim is made with honest agents a and b, then the peer b must be running with the same parameter t, or the adversary has compromised at least one of the two agents.

Faithfulness What about the ordering of the claims in the trace?

This seems to hold even if the running claim follows the commit claim!?! But as the property must hold for ALL traces, we can always consider the prefix of the trace ending with the commit claim, thus implicitly the running claim must happen before.

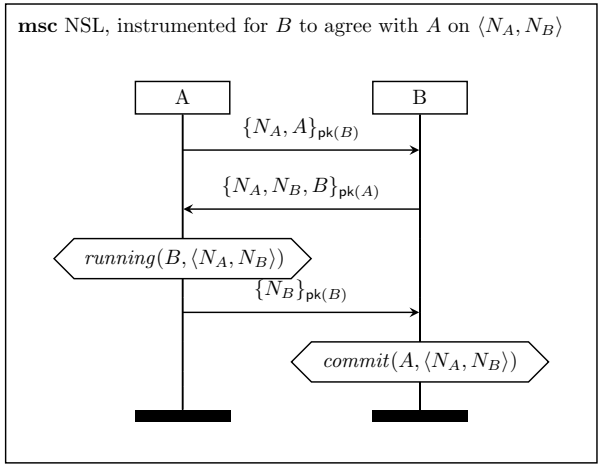
We use role names R_1, R_2 (instead of A, B) to better distinguish them from the agent names a, b. R_1, R_2 will usually be fixed to `'I'` and `'R'`.

Example: NSL Protocol (1/2)



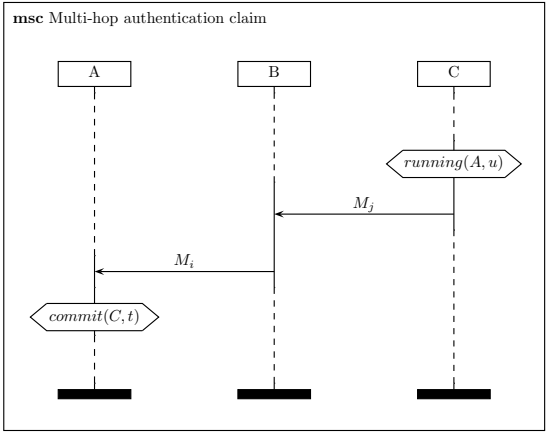
The parts in blue will be omitted in future slides, as they are implicitly given by the message sequence chart.

Example: NSL Protocol (2/2)



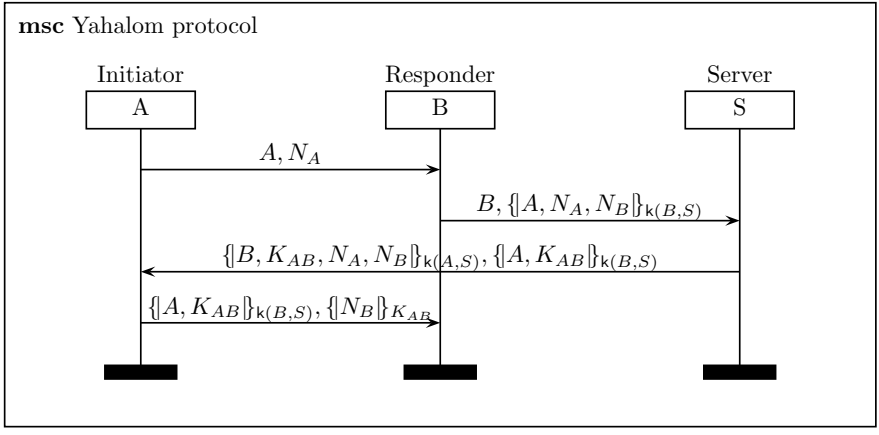
Both claims omit the "Claim_" prefix, the executing agent name, and the `'R','I'` as explained.

Role Instrumentation for Authentication (cont.)

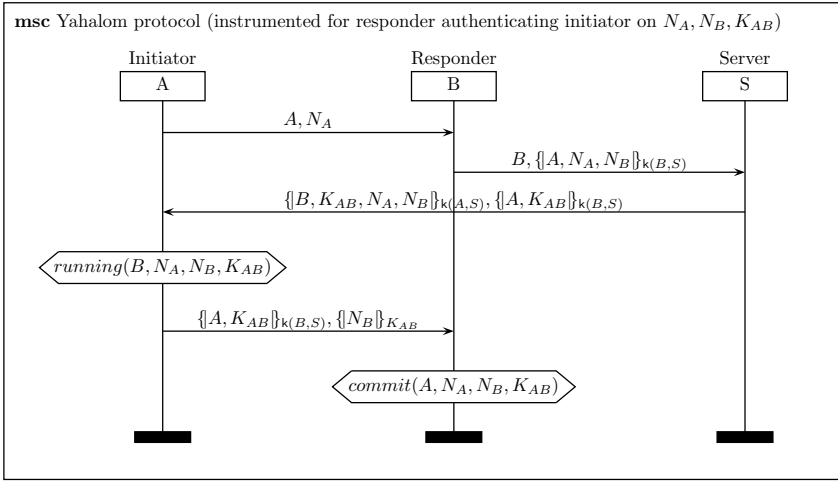


Event causality in multi-hop authentication claims: The *running* event must causally precede the *commit* event and the messages t and u must be known at the position of the claim event in the respective role.

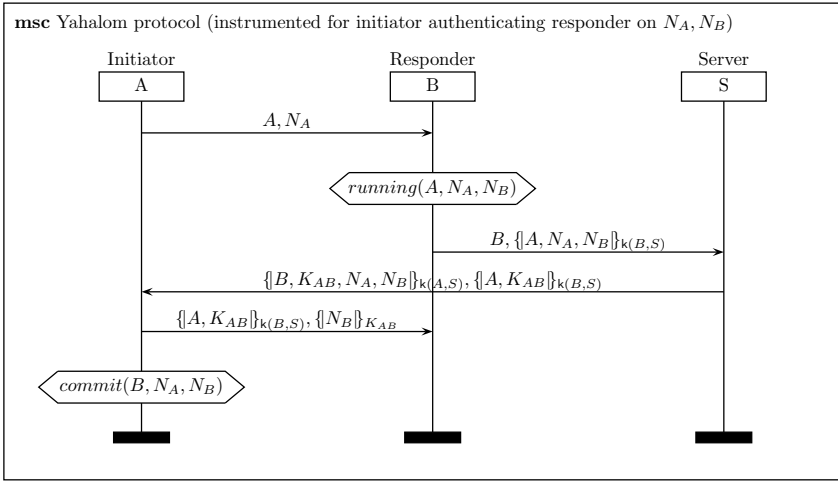
Example: Yahalom Protocol (1/3)



Example: Yahalom Protocol (2/3)



Example: Yahalom Protocol (3/3)



Note: agreement for A on K_{AB} is not possible, since B gets K_{AB} after A.

Formalizing Authentication

Definition (Injective agreement)

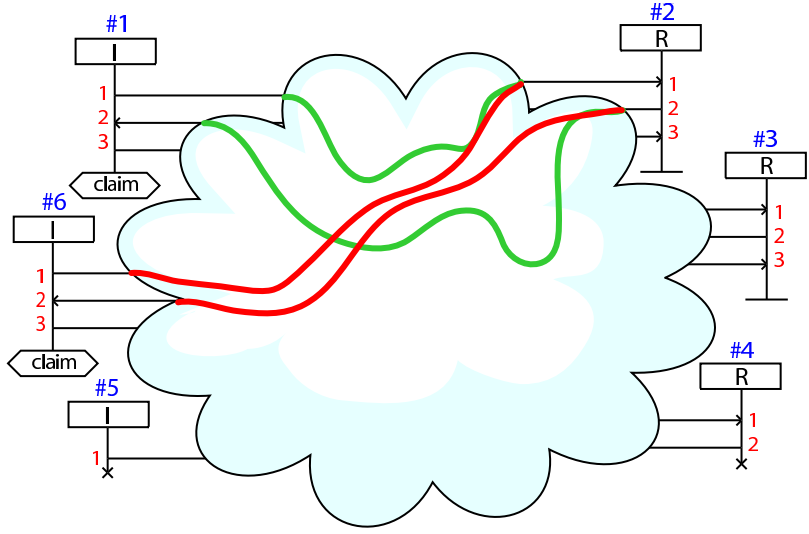
The property $\mathit{Agreement}(R_1, R_2, t)$ consists of all traces satisfying:

$$\begin{aligned}\forall a\ b\ t\ i.\quad & \texttt{Claim\_commit}(a, b, \langle R_1, R_2, t\rangle)@i \\ & \Rightarrow (\exists j.\texttt{Claim\_running}(b, a, \langle R_1, R_2, t\rangle)@j \\ & \qquad \wedge \neg(\exists a_2 b_2 i_2.\texttt{Claim\_commit}(a_2, b_2, \langle R_1, R_2, t\rangle)@i_2 \wedge \neg(i_2 = i))) \\ & \quad \vee(\exists X\ r.\mathrm{Rev}(X)@r \wedge \mathrm{Honest}(X)@i)\end{aligned}$$

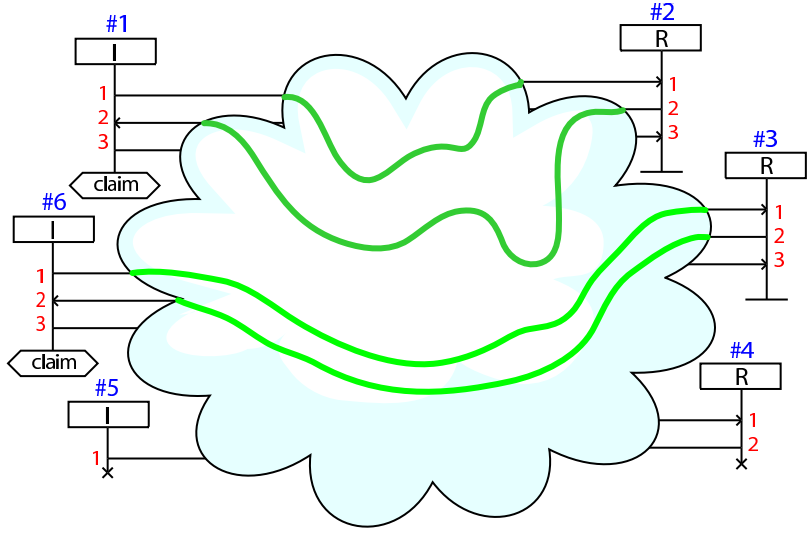
Remarks

- For each commit by a in role R_1 on the trace, there is a unique matching b executing role R_2.

Failed Injective Authentication

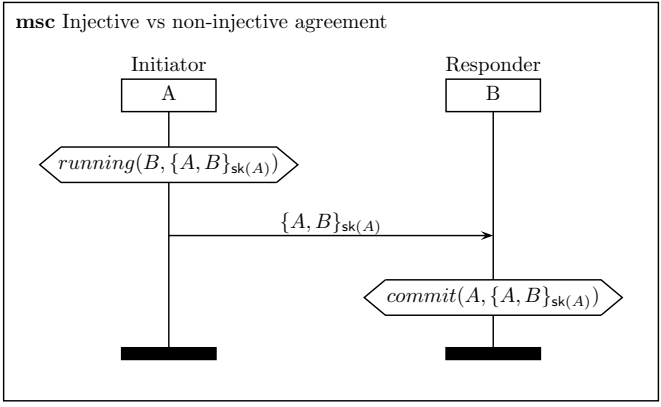


Successful Injective Authentication



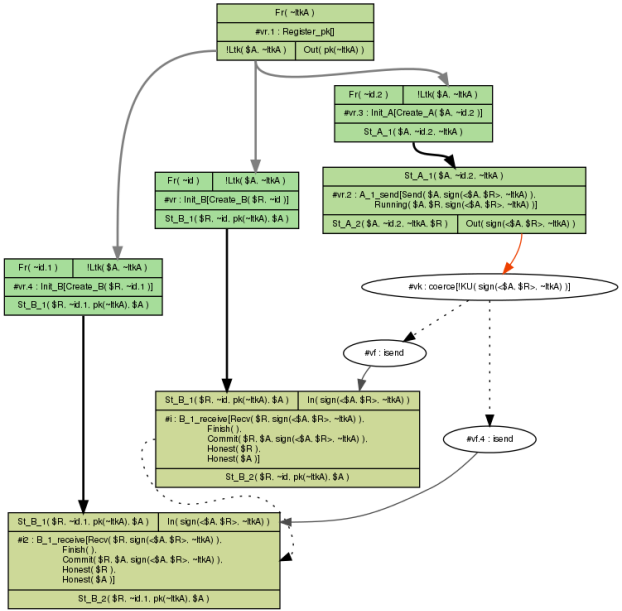
Injective vs Non-injective Agreement

Separating Example



- Non-injective agreement holds.
- Injective agreement fails, since the adversary can replay message to several threads in responder role B.

Injective Agreement Counter-example



Formalizing Authentication

Weaker Variants

Definition (Weak agreement)

A trace *tr* satisfies the property *WeakAgreement* iff

$$\begin{aligned} \forall a\ b\ i.\quad & \mathtt{Claim\_commit}(a, b, \langle\rangle)@i \\ & \Rightarrow (\exists j.\ \mathtt{Claim\_running}(b, a, \langle\rangle)@j) \\ & \quad \vee (\exists X\ r.\ \mathrm{Rev}(X)@r \wedge \mathrm{Honest}(X)@i) \end{aligned}$$

It is sufficient that the agents agree they are communicating, it is not required that they play the right roles. Note also the empty list of data $\langle\rangle$ that is agreed upon, i.e., none.

Formalizing Authentication

Weaker Variants

Definition (Aliveness)

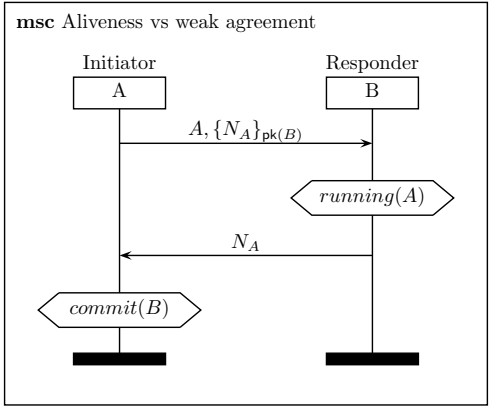
A trace tr satisfies the property *Alive* iff

$$\begin{aligned}\forall a\ b\ i.\quad & \mathrm{Claim\_commit}(a, b, \langle\rangle)@i \\ & \Rightarrow (\exists id\ R\ j.\ \mathrm{Create}(b, id, R)@j) \\ & \quad \vee (\exists X\ r.\ \mathrm{Rev}(X)@r \wedge \mathrm{Honest}(X)@i)\end{aligned}$$

It is neither required that the agent b, believed to instantiate role B by agent a, really plays role B, nor that he believes to be talking to a.

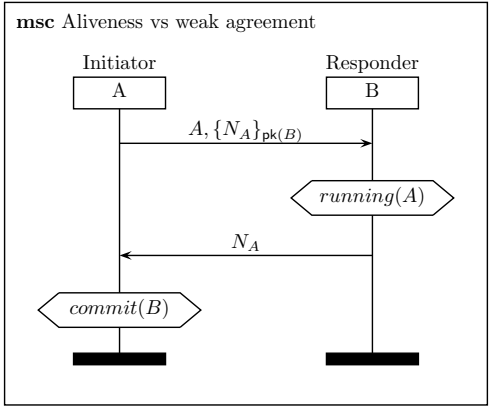
Aliveness vs Weak Agreement

Separating Example



Aliveness vs Weak Agreement

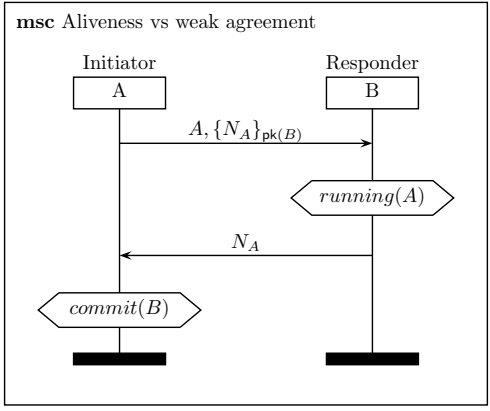
Separating Example



- Aliveness holds: only B can have decrypted the fresh nonce N_A.

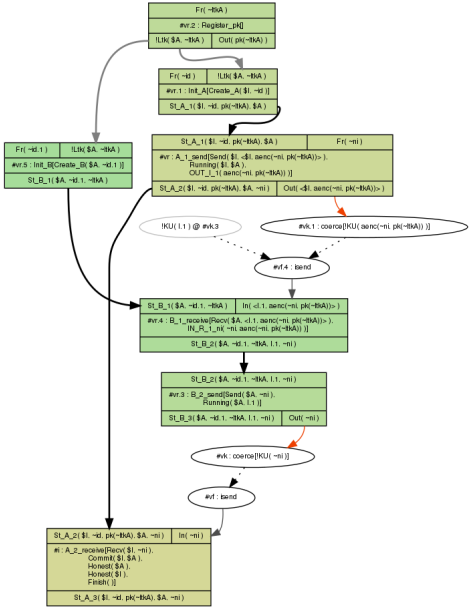
Aliveness vs Weak Agreement

Separating Example

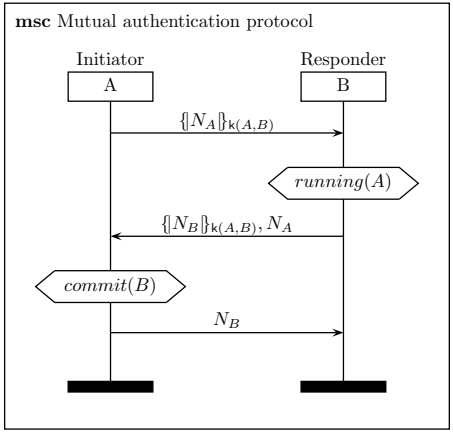


- Aliveness holds: only B can have decrypted the fresh nonce N_A.
- Weak agreement fails, since adversary may modify unprotected identity A to C in first message so that B thinks he is talking to C.

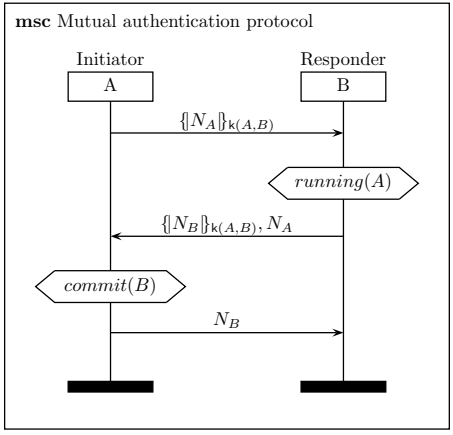
Weak Agreement Counter-example



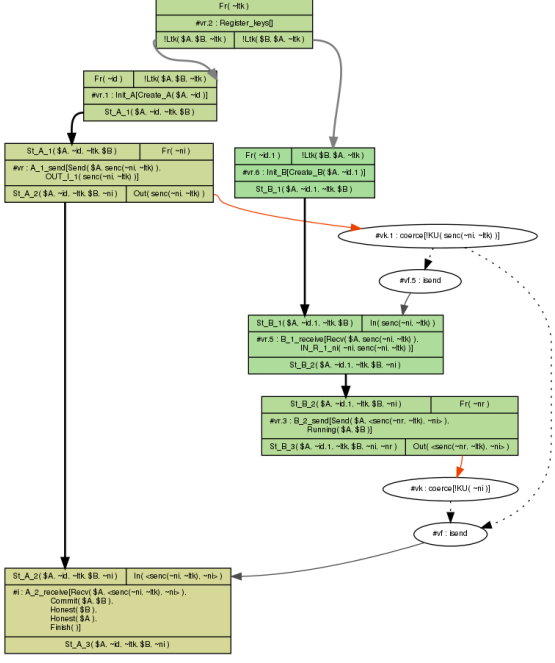
When Even Aliveness Fails ...



When Even Aliveness Fails ...



- Attack: A may play protocol in both roles, communicating with a non-existing B.
- Hence, aliveness fails.


Fr(~ltk)
#vr.2 : Register_keys[]
!Ltk($A, $B, ~ltk)
!Ltk($B, $A, ~ltk)
Fr(~id)
!Ltk($A, $B, ~ltk)
#vr.1 : Init_A[Create_A($A, ~id)]
St_A_1($A, ~id, ~ltk, $B)
St_A_1($A, ~id, ~ltk, $B)
Fr(~ni)
#vr : A_1_send[Send($A, senc(~ni, ~ltk)),
OUT_1_1(senc(~ni, ~ltk))]
St_A_2($A, ~id, ~ltk, $B, ~ni)
Out(senc(~ni, ~ltk))
Fr(~id.1)
!Ltk($B, $A, ~ltk)
#vr.6 : Init_B[Create_B($A, ~id.1)]
St_B_1($A, ~id.1, ~ltk, $B)
#vk.1 : coerce[!KU(senc(~ni, ~ltk))]
#vf.5 : isend
St_B_1($A, ~id.1, ~ltk, $B)
In(senc(~ni, ~ltk))
#vr.5 : B_1_receive[Recv($A, senc(~ni, ~ltk)),
IN_B_1_1(~ni, senc(~ni, ~ltk))]
St_B_2($A, ~id.1, ~ltk, $B, ~ni)
St_B_2($A, ~id.1, ~ltk, $B, ~ni)
Fr(~nr)
#vr.3 : B_2_send[Send($A, <senc(~nr, ~ltk), ~nb>),
Running($A, $B)]
St_B_3($A, ~id.1, ~ltk, $B, ~ni, ~nr)
Out(<senc(~nr, ~ltk), ~nb>)
#vk : coerce[!KU(~ni)]
#vf : isend
St_A_2($A, ~id, ~ltk, $B, ~ni)
In(<senc(~ni, ~ltk), ~nb>)
#i : A_2_receive[Recv($A, <senc(~ni, ~ltk), ~nb>),
Commit($A, $B),
Honest($B),
Honest($A),
Finish( )]
St_A_3($A, ~id, ~ltk, $B, ~ni)

Conclusions

We have seen...

- Security protocols are critical and their design is highly nontrivial
- Formal Methods provide a way to specify and reason about them
  - ★ thereby finding bugs or producing proofs
- Tamarin provides a method with an associated tool
  - ★ based on multiset rewriting, equations, and first-order specifications
  - ★ as well as a powerful inference engine
- Extremely effective in practice (see associated papers)

Appendix: Labeled Multiset Rewriting

General Case

Definition (Labeled multiset rewriting step)

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

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

where

- $lin(l)$ denotes the multiset of linear facts in l, and
- $per(l)$ denotes the set of persistent facts in l.

- The multiplicity of the linear facts matters (multiset inclusion).
- The multiplicity of the persistent facts is immaterial (set inclusion).
- Only the linear facts are removed from l, the persistent ones remain.