# EITP20 – Secure Systems Engineering

## Analysis and Design of Security Protocols (Part II)

Mohsen Toorani

Department of Electrical and Information Technology

February 18, 2020

# Outline

# Multiset Rewriting

# Definitions: Multisets

## Multiset

A multiset $m$ over a set $X$ is a set of elements, each imbued with a multiplicity

$$m : X \to \mathbb{N}$$

where $m(x)$ denotes the multiplicity of $x$.

- Let $\subseteq^\sharp$ denote multiset inclusion, $\cup^\sharp$ denote multiset union, $\setminus^\sharp$ denote multiset difference, and $X^\sharp$ denote the set of finite multisets with elements from X.
- Example: For $A = [0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 3] = [0, 1, 1, 2, 2, 2]$ and $B = [0 \mapsto 3, 1 \mapsto 1, 2 \mapsto 2] = [0, 0, 0, 1, 2, 2]$, we have: $A \cup^\sharp B = [0 \mapsto 4, 1 \mapsto 3, 2 \mapsto 5] = [0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2]$. $A \setminus^\sharp B = [1 \mapsto 1, 2 \mapsto 1] = [1, 2]$.

# Definitions: Signature, Facts, and Multiset Rewriting

## Signature

An unsorted signature $\Sigma$ is a set of function symbols, each having an arity $k \geq 0$. Function symbols of arity 0 are called constants.

## Facts

Let $\Sigma_{fact}$ denote an unsorted signature of fact symbols, each with an arity $k \geq 0$. Then, $F(t_1, \ldots, t_k)$ is called a fact in which $F \in \Sigma_{fact}$ with arity $k$ and $t_1, \ldots, t_k \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{N})$ where $\mathcal{T}_\Sigma$ denotes *term algebra* over $\Sigma$.

## Labeled Multiset Rewriting

A labeled multiset rewriting rule is a triple $l \xrightarrow{a} r$ where l and r are multisets of facts, called state facts, and a is a multiset of facts, called action facts or events.

# Definitions: Executions and Trace

## Executions

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: $S_0 = \emptyset$
2. Corresponds to a transition sequence, i.e.
   $\forall i : S_{i-1}, (l_i \xrightarrow{a_i} r_i), S_i \in steps(R)$
3. Fresh names are unique, i.e.
   $\forall i, j, n : (l_i \xrightarrow{a_i} r_i) = (l_j \xrightarrow{a_j} r_j) = ([] \to [Fr(n)]) \Rightarrow i = j.$

## 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 multisets of its action labels, i.e., $a_1, a_2, \ldots, a_k$

# Modeling Security Protocols using Multiset Rewriting

# Modeling Security Protocols using MSR

Different types of rules in security protocols:

- **Adversary rules** determine which messages the adversary can derive from his knowledge.
- **Fresh rule** generates unique fresh values that can be used as nonces or thread identifiers.
- **Infrastructure rules** formalize generation of cryptographic keys. They can be used to model a public-key infrastructure (PKI).
- **Protocol rules** formalize the roles of the protocol. They define sending and receiving of messages, and use agent state facts to keep track of the role's activities.

# Adversary in Dolev-Yao model

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

A Dolev-Yao adversary can

- construct messages using the inference rules
- eavesdrop on messages
- delay or block messages
- forge and inject messages
- employ malicious agents in the system

Informally, can do anything except breaking cryptography!

# Dolev-Yao Deduction

## Adversary Knowledge

We denote the adversarial knowledge of a term $t$ by a fact K(t).

## Adversary Knowledge Derivation

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

$$\frac{Fr(x)}{K(x)} \qquad \frac{Out(x)}{K(x)} \qquad \frac{K(x)}{In(x)} \qquad \frac{K(t_1), \ldots, K(t_k)}{K(f(t_1, \ldots, t_k))} \; \forall f \in \Sigma(\text{k-ary})$$

## Adversary Knowledge Derivation in rewrite rules

$$[Fr(x)] \rightarrow [K(x)] \qquad [Out(x)] \rightarrow [K(x)] \qquad [K(x)] \xrightarrow{K(x)} [In(x)]$$
$$[K(x_1), \ldots, K(x_k)] \rightarrow [K(f(x_1, \ldots, x_k))] \; \forall f \in \Sigma(\text{k-ary})$$

In the third case, the adversary derives a message and sends it via *In*.

# Adversary Rules

## Adversary Knowledge Derivation

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

$$\frac{Fr(x)}{K(x)} \qquad \frac{Out(x)}{K(x)} \qquad \frac{K(x)}{In(x)} \qquad \frac{K(t_1), \ldots, K(t_k)}{K(f(t_1, \ldots, t_k))} \ \forall f \in \Sigma(\text{k-ary})$$

- 1st rule models that the adversary can use fresh values.
- 2nd and 3rd rules model the interface with the protocol:
  - *Out* facts mark messages sent by the protocol.
  - *In* facts mark messages received by the protocol.
- 4th rule models the adversary's inference capabilities.

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

# Fresh Rule

## Fresh Rule

Fresh Rule generates fresh terms/facts. It does not have any precondition and is the only rule allowed to create fresh facts:

$$[] \rightarrow [Fr(N)]$$

Nonce $N$ will be fresh and unique.

Fresh terms represent randomness being used. They are assumed unguessable and unique (e.g. nonce).

# Infrastructure Rule

## Key Generation for PKI

Generates long-term public and private keys:

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

- Ltk: Long-term key
- A's long-term private key will be fresh value $sk$, generated by $Fr$.
- Fact $Ltk(A, sk)$ registers $sk$ as A's long-term private key.
- Fact $Pk(A, pk(sk))$ registers $pk(sk)$ as A's long-term public key.
- Fact $Out(pk(sk))$ publishes A's public key $pk(sk)$.

# Protocol Rules

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

## Agent state facts

An agent state fact for role $R$ is a fact

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

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

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

# Protocol rules: Communications

- Messages are sent and received via Out and In facts, respectively.
- Any protocol rule with Out and In fact has also a matching Send and Recv action, respectively.

Examples:

- Receive rule:
  $[St\_I\_2(A, id, k), In(m)] \xrightarrow{Recv(A,m)} [St\_I\_3(A, id, k, m)]$
- Send rule:
  $[St\_I\_3(A, id, k, m)] \xrightarrow{Send(A,\{m\}_k)} [St\_I\_4(A, id, k, m), Out(\{m\}_k)]$

# Protocol Rules: Formal definition

## Protocol Rule

A multiset-rewriting rule $l \xrightarrow{a} r$ is a protocol rule if following conditions are satisfied (except for initialization rules):

1. Only *In*, *Fr*, and agent state facts occur in $l$.
2. Only *Out* and agent state facts occur in $r$.
3. Either *In* or *Out* facts occur in the rule, but never both.
4. Exactly one agent state fact occurs in each of $l$ and $r$. If the fact $St\_R\_s(A, id, k_1, \ldots, k_n)$ occurs in $l$ then the fact $St\_R\_s'(A, id, k_1', \ldots, k_m')$ occurs in $r$ where $s' = s + 1$.
5. Every variable in $r$ must occur in $l$.

# Well-formedness

Protocol rules must be well-formed to be executable.

## Well-formedness

A protocol rule $l \xrightarrow{a} r$ with agent state facts

$$St\_R\_s(A, id, k_1, \ldots, k_n) \in l \text{ and } St\_R\_s'(A, id, k'_1, \ldots, k'_m) \in r$$

is well-formed if all terms in $\{k'_1, \ldots, k'_m\} \cup \{t | Out(t) \in r\}$ are derivable from those terms in

$$PV \cup \{k_1, \ldots, k_n\} \cup \{u | Fr(u) \in l\} \cup \{v | In(v) \in l\}$$

in which $PV$ denotes the public values.

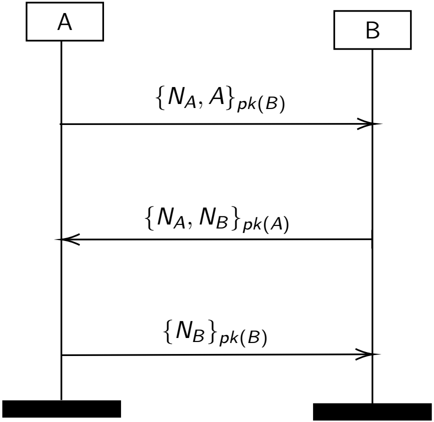
# Well-formedness: Examples

1. $[St\_I\_2(A, id, B, m), Ltk(B, skB)] \xrightarrow{Send(A,\{m\}_{pk(skB)})}$
   $[St\_I\_5(A, id, B, m), Out(\{m\}_{pk(skB)})]$
   Not a protocol rule: $2 + 1 = 3$ and *Ltk* occurred on the left-hand side.

2. $[St\_I\_2(A, id, B), In(\{m\}_{pk(skA)})] \xrightarrow{Recv(A,\{m\}_{pk(skA)})}$
   $[St\_I\_3(A, id, m)]$
   ill-formed: A does not have knowledge of private key *skA* to extract *m*.

3. $[St\_R\_1(A, id, B, skA), In(\{m\}_{pk(skA)})] \xrightarrow{Recv(A,\{m\}_{pk(skA)})}$
   $[St\_R\_2(A, id, B, skA, m)]$
   well-formed.

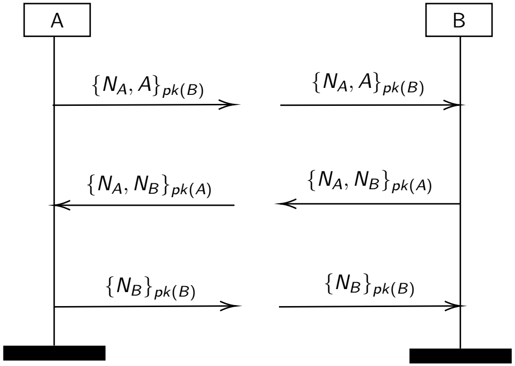
# Example: Writing protocol rules for NSPK Protocol



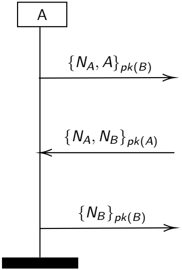

Message sequence chart for
Needham-Schroeder Public Key (NSPK) protocol (1978)

# Example: Preparing for A & B specifications

First split the message sequence chart into single roles where chords, symbolic strands and role scripts are separated:



# Example: Role specification rules

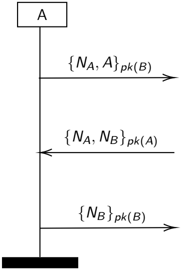



$$[St\_A\_1(A, id, skA, pkB), Fr(N_A)] \xrightarrow{Send(A,\{N_A,A\}_{pkB})} [St\_A\_2(A, id, skA, pkB, N_A), Out(\{N_A, A\}_{pkB})]$$

$$[St\_A\_2(A, id, skA, pkB, N_A), In(\{N_A, N_B\}_{pk(skA)})] \xrightarrow{Recv(A,\{N_A,N_B\}_{pk(skA)})} [St\_A\_3(A, id, skA, pkB, N_A, N_B)]$$

$$[St\_A\_3(A, id, skA, pkB, N_A, N_B)] \xrightarrow{Send(A,\{N_B\}_{pkB})} [St\_A\_4(A, id, skA, pkB, N_A, N_B), Out(\{N_B\})_{pkB}]$$

# Example: Initialization rule




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

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

The thread knowledge is also initialized (in this example with $skA, B, pkB$).

# Security Property Specification

# Protocol instrumentation

- An approach to formalize security properties independent of protocols.
- We insert special claim events into the protocol roles as $Claim\_claimtype(A, t)$ where *claimtype* indicates the type of claim, $A$ is the executing agent, and $t$ is a message term.
  Example: $Claim\_secret(A, N_A)$ claims that $N_A$ is a secret for agent A and not known to the adversary.
- Claim events are part of the protocol rules as actions, and used to record facts or claims in the protocol trace.
- Claim events cannot be observed, modified, or generated by the adversary.

# Security Property Specification

## Frequently used events

| | |
|---|---|
| $Claim\_claimtype(A, t)$ | Claim event |
| $Send(A, t)$, $Recv(A, t)$, $Create(A, id, R)$ | Protocol events |
| $Honest(A)$, $Rev(A)$ | Honesty & reveal events |
| $K(t)$ | Adversary knowledge |

## Property specification language

Security properties are formulated in first-order logic over predicates:

| | |
|---|---|
| $F@i$ | timestamped event |
| $t = u$ | term equality |
| $i = j$ | timepoint equality |
| $i < j$ | timepoint inequality |

Note: Predicate F@i holds on trace $tr = a_1, \ldots, a_n$ if $F \in a_i$.

# Formalizing Secrecy

# Secrecy

## Secrecy

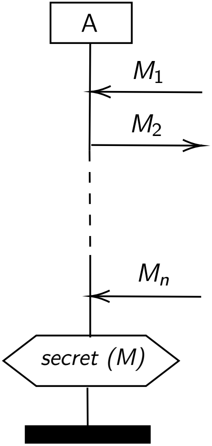
A term t is secret for an agent A in role R iff whenever A executes R and believes to be communicating with honest agents, t will not be inferable from the adversary's knowledge.

- Informally: Adversary cannot discover information that is intended to be secret. Secrecy is a local property.
- Secrecy can only be considered when all alleged communication partners are honest.

# Role Instrumentation for Secrecy

For claiming that message $M$ used in the role remains secret:

- Insert claim event $Claim\_secret(A, M)$ into role A at the end of the role.




Secrecy claim

* We simply use $claimtype(t)$ inside hexagons instead of $Claim\_claimtype(A, t)$.

# Compromised & Honest Agents

## Compromised Agent

A compromised agent is under full adversarial control and shares its long-term secrets with the adversary.

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

- *Rev* (reveal event) is used in property specifications.
- *Out* state fact publishes *skA* so the adversary can perform all send and receive steps of a compromised agent.

## Honest Agent

An agent A is honest in a trace *tr* if $Rev(A) \notin tr$.

- *Honest* (honesty event) is used in property specifications for all parties that are expected to be honest.

# Formalizing Secrecy

## Weak Secrecy

The weak secrecy property consists of all traces *tr* satisfying

$$\forall\ A\ M\ i.\ (Claim\_secret(A, M)@i) \Rightarrow \neg(\exists j.\ K(M)@j)$$

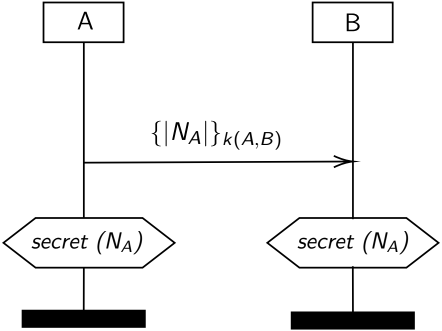
- Above definition works if both A and B are honest, or if the adversary is passive (does not participate in the protocol).
- It is trivially broken if A or B is a compromised agent.

## Secrecy

The secrecy property consists of all traces *tr* satisfying

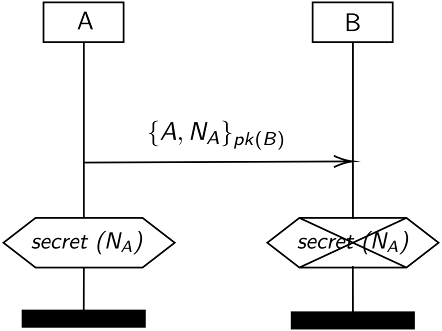
$$\begin{aligned}&\forall\ A\ M\ i.\ (Claim\_secret(A, M)@i)\\&\qquad \Rightarrow \neg(\exists j.K(M)@j) \vee (\exists\ B\ k.\ Rev(B)@k \wedge Honest(B)@i)\end{aligned}$$

- Above definition guarantees the secrecy of M unless the adversary has compromised an agent that is required to be honest.




- The secrecy holds for both A and B.
- We simply omit obvious $Honest(A)$ and $Honest(B)$ annotations in message sequence charts for 2-party protocols.

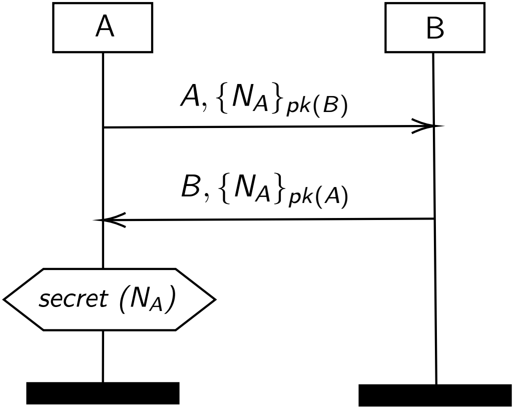
# Secrecy: Example 2




- The secrecy holds for A (because she knows that only B can decrypt and extract $N_A$).
- The secrecy does not hold for B (because she does not know who encrypted the message).

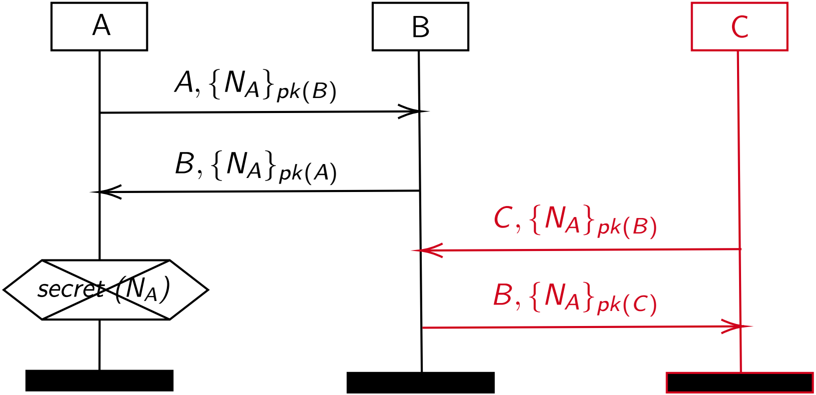
# Secrecy: Example 3

Find an attack on the secrecy claim in the following protocol:



# Secrecy: Example 3

Answer:



# Formalizing Forward Secrecy

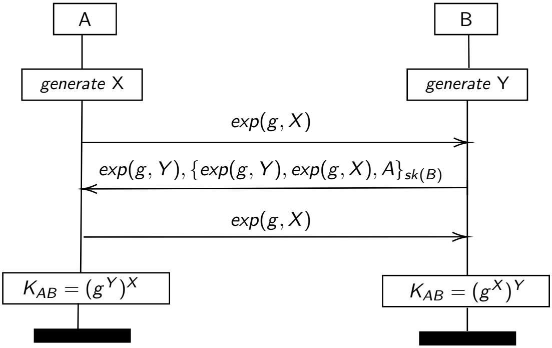
## Forward Secrecy

The forward secrecy property consists of all traces *tr* satisfying

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

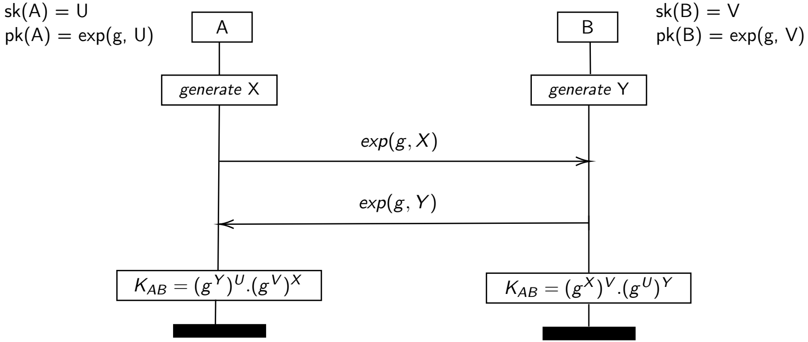
It guarantees the secrecy of $M$ unless the adversary has previously compromised an agent that is required to be honest.

# Forward secrecy: Example 1




- Modified station-to-station protocol: Digital signature is used to authenticate Diffie-Hellman public keys $exp(g, X)$ and $exp(g, Y)$.
- It provides forward secrecy: Even if long-term signing keys are compromised, the adversary cannot compute the session key $K_{AB} = exp(exp(g, X), Y)$.

# Forward secrecy: Example 2




- MTI A(0) Protocol due to Matsumoto-Takashima-Imai: Message exchange as in basic Diffie-Hellman but it combines long-term and ephemeral keys.
- It does not provide forward secrecy: Assuming that private keys $U$ and $V$ are compromised, the adversary can construct the session key $K_{AB} = g^{XV+YU}$ as $(g^X)^V.(g^Y)^U$ from the exchanged messages and long-term private keys.

# Formalizing Authentication

# Formalizing Authentication

- There is no unique definition of authentication:
  - weak / non-injective / injective agreements, weak/strong authentication, ping authentication, aliveness, synchronization, matching histories, etc.
- Lowe (1997) defined the following properties:
  - **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 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*.
  - **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**: In addition to being non-injective, each run of agent a in role A should correspond to a unique run of agent b.

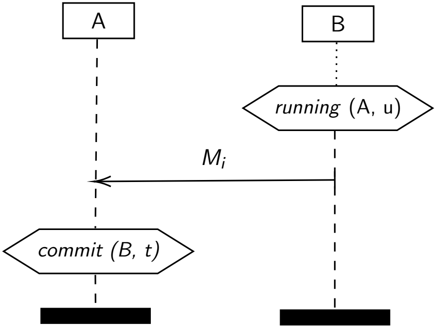
For showing that role A authenticates role B on t, we use two claims:

**In role A:**

- Insert a commit claim event $Claim\_commit(A, B, t)$.
- After A can construct t (typically at the end of role A).

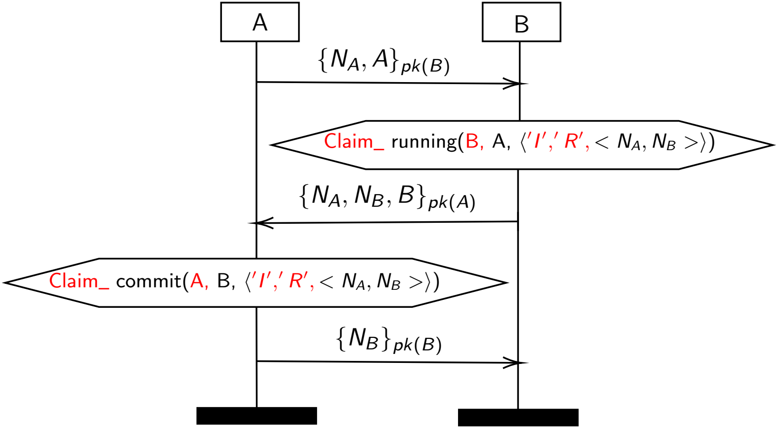
**In role B:**

- Insert a running claim event $Claim\_running(B, A, u)$.
- After B can construct u (causally before $Claim\_commit(A, B, t)$).
- Term u is B's view of t.




Authentication claim

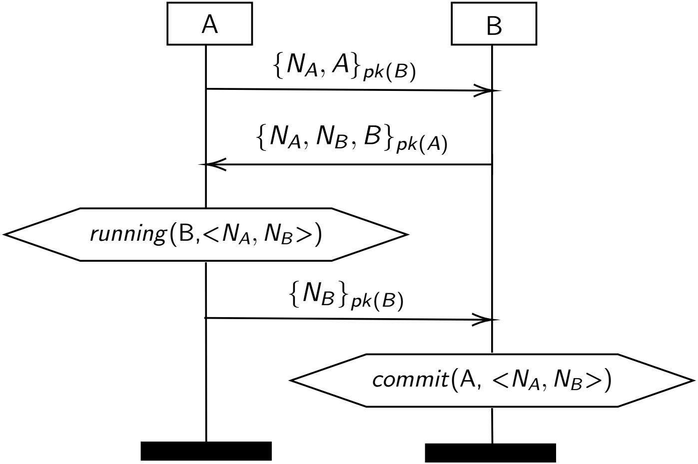
# Example: Role instrument on NSL protocol




NSL protocol when instrumented for A to agree with B on $\langle N_A, N_B \rangle$

- NSL: Needham-Shroeder-Lowe
- We simply omit red parts from message sequence charts in next slides.

# Example: Role instrument on NSL protocol




NSL protocol when instrumented for B to agree with A on $<N_A, N_B>$

# Formalizing Authentication: Aliveness

## Aliveness

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

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

- This is the weakest variant.
- Agent b that believed to instantiate role B by agent a is not required to really play role B or believe that he is talking to a!

# Formalizing Authentication: Week agreement

## Weak agreement

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

$$\begin{aligned}
&\forall\ a\ b\ i.\ Claim_commit(a, b, \langle\rangle)@i \\
&\qquad \Rightarrow (\exists j.\ Claim_running(b, a, \langle\rangle)@j) \\
&\qquad\quad \vee(\exists X\ \ r.Rev(X)@r \wedge Honest(X)@i)
\end{aligned}$$

- For having week agreement, it is sufficient that the agents agree they are communicating.
- It is not required that they play the right roles.
- List of data that should be agreed upon is empty $\langle\rangle$.

# Formalizing Authentication: Non-injective agreement

## Non-injective agreement

Property $\color{red}{Agreement_{NI}}('I','R',t)$ consists of all traces satisfying

$$\begin{aligned}\forall\ a\ b\ t\ i.\ & Claim\_commit(a, b, \langle 'I','R', t\rangle)@i \\ & \Rightarrow (\exists j. Claim\_running(b, a, \langle 'I','R', t\rangle)@j) \\ & \quad \vee(\exists X r. Rev(X)@r \wedge Honest(X)@i)\end{aligned}$$

- Whenever a commit claim is made with honest agents a and b, the peer b must be running with the same parameter $t$, or the adversary has compromised at least one of the two agents.
- We used role names 'I' and 'R' (instead of A and B) to better distinguish them from the agent names a and b.

# Formalizing Authentication: Injective agreement

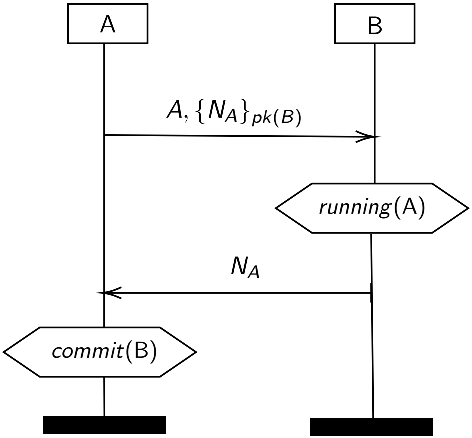
## Injective agreement

Property *Agreement*$('I','R',t)$ consists of all traces satisfying

$$
\begin{aligned}
\forall\ a\ b\ t\ i.\ & Claim\_commit(a, b, \langle 'I', 'R', t\rangle)@i \\
& \Rightarrow (\exists j.\ Claim\_running(b, a, \langle 'I', 'R', t\rangle)@j \\
& \quad \wedge\neg(\exists a_2 b_2 i_2.\ Claim\_commit(a_2, b_2, \langle 'I', 'R', t\rangle)@i_2 \wedge \neg(i_2 = i))) \\
& \quad \vee(\exists X\ r.Rev(X)@r \wedge Honest(X)@i)
\end{aligned}
$$

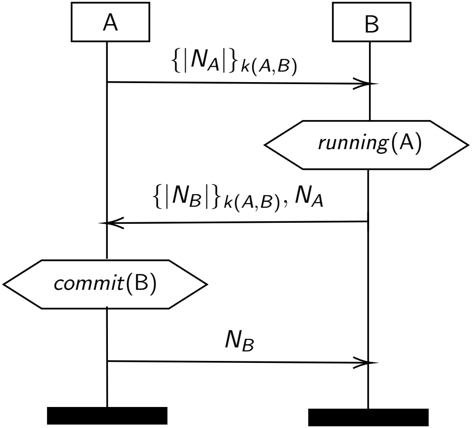
- For each commit by a in role 'I' on the trace, there is a unique matching b which executes role 'R'.

# Example: Aliveness vs Weak Agreement



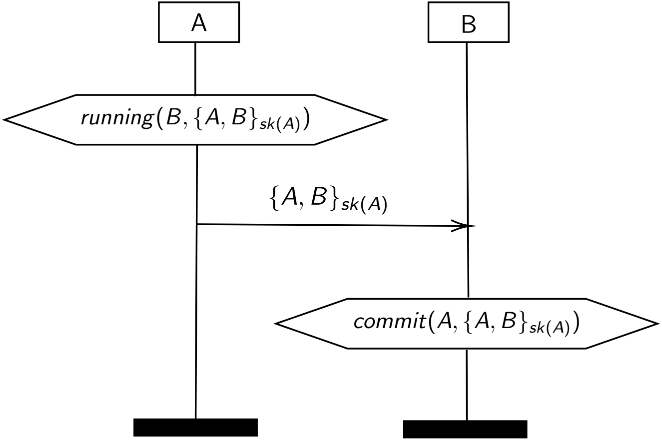

- **Aliveness** holds: only B can decrypt the fresh nonce $N_A$.
- **Weak agreement** fails: Adversary modifies plain identity A to C in the first message and B thinks he is talking to C.

# Example: when everything fails




- The protocol is supposed to provide mutual authentication.
- But even the **aliveness** fails due to a reflection attack: A can complete the protocol run even without B's participation!

# Example: Injective vs non-injective agreement




- **Non-injective agreement** holds but **injective agreement** fails!
- Adversary can replay message to several threads in responder role B.

# Tamarin

# Tamarin

- *Tamarin prover* is a verification tool for security protocols.
- It supports both falsification and unbounded verification of security protocols in the symbolic model.

# Equations in Tamarin

Tamarin supports

- 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, etc.

Example in Tamarin Syntax:

```
functions: h/1, senc/2, sdec/2
equations: sdec(senc(m,k),k) = m
builtins: diffie-hellman, bilinear-pairing, multiset
/* Other convenient builtins:
hashing, asymmetric-encryption, symmetric-encryption,
signing, revealing-signing */
```

# Multiset rewriting in Tamarin

In Tamarin, protocols are modeled using rewrite rules operating on multisets of facts $l \xrightarrow{a} r$.

Example:

rule 1: $\xrightarrow{Init()} A('4'), C('2')$

rule 2: $A(x) \xrightarrow{Step(x)} B(x)$

In Tamarin syntax:

```
rule 1: [ ] --[ Init() ]-> [ A('4'), C('2') ]
rule 2: [ A(x) ] --[ Step(x) ]-> [ B(x) ]

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

# Fresh and Public Terms in Tamarin

## Fresh terms

Agents generate fresh terms using fresh facts, denoted by Fr.

In Tamarin, fresh variables are prefixed with $\sim$ (e.g. $\sim r$).

## Public terms

Public terms are terms that are known to all participants of a protocol. These include all agent names and all constants.

In Tamarin, public variables are prefixed with \$ (e.g. \$Y).

# Communication and Persistent Facts in Tamarin

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

Example (Input and Output)

```
rule 4: [ Key(x), In(y) ] --> [ Out( senc(y,x) ) ]
```

Facts can be linear or persistent:

- Linear facts can only be consumed once. By default, facts are linear.
- Persistent facts can be consumed infinitely/often.

In Tamarin, persistent facts are marked with !

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